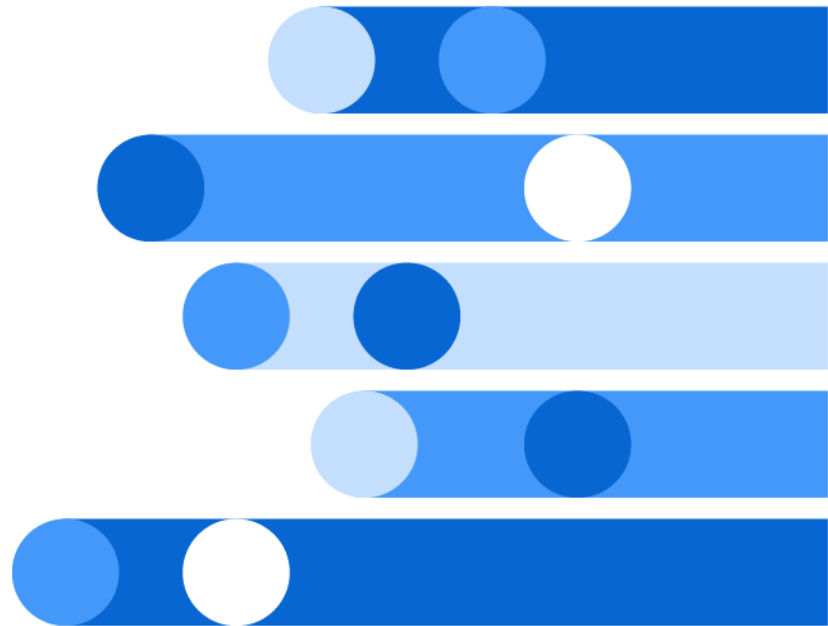




Base SAS[®] Procedures Guide

2021.1.6 - 2026.09*



* This document might apply to additional versions of the software. Open this document in [SAS Help Center](#) and click on the version in the banner to see all available versions.



The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2021. *Base SAS® Procedures Guide*. Cary, NC: SAS Institute Inc.

Base SAS® Procedures Guide

Copyright © 2021, SAS Institute Inc., Cary, NC, USA

All Rights Reserved. Produced in the United States of America.

For a hard copy book: No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc.

For a web download or e-book: Your use of this publication shall be governed by the terms established by the vendor at the time you acquire this publication.

The scanning, uploading, and distribution of this book via the Internet or any other means without the permission of the publisher is illegal and punishable by law. Please purchase only authorized electronic editions and do not participate in or encourage electronic piracy of copyrighted materials. Your support of others' rights is appreciated.

U.S. Government License Rights; Restricted Rights: The Software and its documentation is commercial computer software developed at private expense and is provided with RESTRICTED RIGHTS to the United States Government. Use, duplication, or disclosure of the Software by the United States Government is subject to the license terms of this Agreement pursuant to, as applicable, FAR 12.212, DFAR 227.7202-1(a), DFAR 227.7202-3(a), and DFAR 227.7202-4, and, to the extent required under U.S. federal law, the minimum restricted rights as set out in FAR 52.227-19 (DEC 2007). If FAR 52.227-19 is applicable, this provision serves as notice under clause (c) thereof and no other notice is required to be affixed to the Software or documentation. The Government's rights in Software and documentation shall be only those set forth in this Agreement.

SAS Institute Inc., SAS Campus Drive, Cary, NC 27513-2414

September 2026

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.

Other brand and product names are trademarks of their respective companies.

v_002-P1:proc

Contents

<i>Syntax Conventions for the SAS Language</i>	<i>xiii</i>
--	-------------

PART 1 Concepts 1

Chapter 1 / Choosing the Right Procedure	3
Functional Categories of Base SAS Procedures	3
Report-Writing Procedures	5
Statistical Procedures	6
Utility Procedures	9
Brief Descriptions of Base SAS Procedures	12
Chapter 2 / Fundamental Concepts for Using Base SAS Procedures	19
Language Concepts	20
Procedure Concepts	24
Output Delivery System	71
Chapter 3 / In-Database Processing of Base Procedures	73
Base Procedures That Are Enhanced for In-Database Processing	73
Chapter 4 / CAS Processing of Base Procedures	75
Procedures That Use CAS Actions	75
When CAS Processing Cannot Be Used	77
BY-Group Processing	77
Filtering Observations	78
Related Documents	78
Chapter 5 / Base SAS Procedures Documented in Other Publications	81
Base SAS Procedures Documented in Other Publications	81
Chapter 6 / Base SAS Procedures That Are No Longer Supported	85
Base SAS Procedures That Are Not Supported in SAS Viya Platform	85

PART 2 Procedures 87

Chapter 7 / ACCELERATOR Procedure	91
Overview: ACCELERATOR Procedure	91
Concepts: ACCELERATOR Procedure	92
Syntax: ACCELERATOR Procedure	99

Examples: ACCELERATOR Procedure	110
Chapter 8 / APPEND Procedure	125
Overview: APPEND Procedure	125
Syntax: APPEND Procedure	126
Usage: APPEND Procedure	135
Examples: APPEND Procedure	136
Chapter 9 / CATALOG Procedure	149
Overview: CATALOG Procedure	150
Concepts: CATALOG Procedure	150
Syntax: CATALOG Procedure	150
Usage: CATALOG Procedure	163
Results: CATALOG Procedure	168
Examples: CATALOG Procedure	169
Chapter 10 / CIMPORT Procedure	179
Overview: CIMPORT Procedure	179
Syntax: CIMPORT Procedure	181
Usage: CIMPORT Procedure	193
Examples: CIMPORT Procedure	201
Chapter 11 / COMPARE Procedure	209
Overview: COMPARE Procedure	210
Concepts: COMPARE Procedure	212
Syntax: COMPARE Procedure	216
Usage: COMPARE Procedure	241
Results: COMPARE Procedure	244
Examples: COMPARE Procedure	262
Chapter 12 / CONTENTS Procedure	289
Overview: CONTENTS Procedure	289
Concepts: CONTENTS Procedure	290
Syntax: CONTENTS Procedure	292
Usage: CONTENTS Procedure	297
Examples: CONTENTS Procedure	298
Chapter 13 / COPY Procedure	309
Overview: COPY Procedure	309
Syntax: COPY Procedure	311
Usage: COPY Procedure	328
Examples: COPY Procedure	335
Chapter 14 / CPORT Procedure	341
Overview: CPORT Procedure	341
Syntax: CPORT Procedure	342
Usage: CPORT Procedure	354
Examples: CPORT Procedure	357
Chapter 15 / DATASETS Procedure	365
Overview: DATASETS Procedure	366
Concepts: DATASETS Procedure	368

Syntax: DATASETS Procedure	378
Usage: DATASETS Procedure	467
Results: DATASETS Procedure	474
Examples: DATASETS Procedure	492
Chapter 16 / DATEKEYS Procedure	527
Overview: DATEKEYS Procedure	528
Concepts: DATEKEYS Procedure	528
Syntax: DATEKEYS Procedure	532
Usage: DATEKEYS Procedure	545
Examples: DATEKEYS Procedure	561
References	583
Chapter 17 / DELETE Procedure	585
Overview: DELETE Procedure	585
Concepts: DELETE Procedure	586
Syntax: DELETE Procedure	586
Examples: DELETE Procedure	590
Chapter 18 / DS2 Procedure	597
Overview: DS2 Procedure	598
Concepts: DS2 Procedure	598
Syntax: DS2 Procedure	600
Usage: DS2 Procedure	610
Examples: DS2 Procedure	620
Chapter 19 / DSTODS2 Procedure	631
Overview: DSTODS2 Procedure	631
Concepts: DSTODS2 Procedure	632
Syntax: DSTODS2 Procedure	636
Examples: DSTODS2 Procedure	637
Chapter 20 / EXPORT Procedure	643
Overview: EXPORT Procedure	643
Syntax: EXPORT Procedure	645
Examples: EXPORT Procedure	654
Chapter 21 / FCMP Procedure	669
Overview: FCMP Procedure	670
Concepts: FCMP Procedure	671
Syntax: FCMP Procedure	677
Usage: FCMP Procedure	698
Examples: FCMP Procedure	731
References	745
Chapter 22 / FCMP Special Functions and Call Routines	747
Overview of Special Functions and CALL Routines	748
Functions and CALL Routines by Category	748
Dictionary	750
Chapter 23 / FEDSQL Procedure	793
Overview: FEDSQL Procedure	794

Concepts: FEDSQL Procedure	795
Syntax: FEDSQL Procedure	797
Usage: FEDSQL Procedure	806
Examples: FEDSQL Procedure	816
Chapter 24 / FMTC2ITM Procedure	837
Overview: FMTC2ITM Procedure	837
Syntax: FMTC2ITM Procedure	838
Examples: FMTC2ITM Procedure	841
Chapter 25 / FONTREG Procedure	845
Overview: FONTREG Procedure	845
Concepts: FONTREG Procedure	846
Syntax: FONTREG Procedure	850
Examples: FONTREG Procedure	858
Chapter 26 / FORMAT Procedure	863
Overview: FORMAT Procedure	864
Concepts: FORMAT Procedure	866
Syntax: FORMAT Procedure	872
Usage: FORMAT Procedure	918
Results: FORMAT Procedure	923
Examples: FORMAT Procedure	931
Chapter 27 / GATEWAY Procedure	993
Overview: GATEWAY Procedure	994
Concepts: GATEWAY Procedure	994
Syntax: GATEWAY Procedure	995
Usage: GATEWAY Procedure	1004
Chapter 28 / GROOVY Procedure	1009
Overview: GROOVY Procedure	1009
Syntax: GROOVY Procedure	1010
Usage: GROOVY Procedure	1018
Examples: GROOVY Procedure	1020
Chapter 29 / HADOOP Procedure	1023
Overview: HADOOP Procedure	1023
Syntax: HADOOP Procedure	1024
Usage: HADOOP Procedure	1036
Examples: HADOOP Procedure	1038
Chapter 30 / HTTP Procedure	1045
Overview: HTTP Procedure	1046
Syntax: HTTP Procedure	1046
Usage: HTTP Procedure	1060
Examples: HTTP Procedure	1066
Chapter 31 / IMPORT Procedure	1085
Overview: IMPORT Procedure	1085
Syntax: IMPORT Procedure	1090
Examples: IMPORT Procedure	1105

Chapter 32 / JAVAINFO Procedure	1121
Overview: JAVAINFO Procedure	1121
Syntax: JAVAINFO Procedure	1121
Chapter 33 / JSON Procedure	1123
Overview: JSON Procedure	1124
Concepts: JSON Procedure	1124
Syntax: JSON Procedure	1129
Usage: JSON Procedure	1142
Examples: JSON Procedure	1146
Chapter 34 / LUA Procedure	1163
Overview: LUA Procedure	1164
Concepts: LUA Procedure	1165
Syntax: LUA Procedure	1180
Usage: LUA Procedure	1184
Examples: LUA Procedure	1186
Chapter 35 / MEANS Procedure	1227
Overview: MEANS Procedure	1228
Concepts: MEANS Procedure	1231
Syntax: MEANS Procedure	1239
Usage: MEANS Procedure	1293
Results: MEANS Procedure	1296
Examples: MEANS Procedure	1299
References	1341
Chapter 36 / MIGRATE Procedure	1343
Overview: MIGRATE Procedure	1344
Concepts: MIGRATE Procedure	1344
Syntax: MIGRATE Procedure	1347
Usage: MIGRATE Procedure	1350
Examples: MIGRATE Procedure	1355
Chapter 37 / OPTIONS Procedure	1363
Overview: OPTIONS Procedure	1363
Syntax: OPTIONS Procedure	1364
Usage: OPTIONS Procedure	1370
Results: OPTIONS Procedure	1378
Examples: OPTIONS Procedure	1379
Chapter 38 / OPTLOAD Procedure	1385
Overview: OPTLOAD Procedure	1385
Syntax: OPTLOAD Procedure	1385
Examples: OPTLOAD Procedure	1387
Chapter 39 / OPTSAVE Procedure	1391
Overview: OPTSAVE Procedure	1391
Syntax: OPTSAVE Procedure	1392
Usage: OPTSAVE Procedure	1393
Examples: OPTSAVE Procedure	1395

Chapter 40 / PRINT Procedure	1397
Overview: PRINT Procedure	1398
Syntax: PRINT Procedure	1400
Usage: PRINT Procedure	1436
Results: PRINT Procedure	1447
Examples: PRINT Procedure	1451
Chapter 41 / PRINTTO Procedure	1497
Overview: PRINTTO Procedure	1497
Syntax: PRINTTO Procedure	1498
Usage: PRINTTO Procedure	1503
Examples: PRINTTO Procedure	1505
Chapter 42 / PROTO Procedure	1519
Overview: PROTO Procedure	1520
Concepts: PROTO Procedure	1520
Syntax: PROTO Procedure	1528
Usage: PROTO Procedure	1534
Examples: PROTO Procedure	1544
Chapter 43 / PRTDEF Procedure	1547
Overview: PRTDEF Procedure	1547
Syntax: PRTDEF Procedure	1548
Usage: PRTDEF Procedure	1550
Examples: PRTDEF Procedure	1556
Chapter 44 / PRTEXP Procedure	1567
Overview: PRTEXP Procedure	1567
Syntax: PRTEXP Procedure	1568
Examples: PRTEXP Procedure	1570
Chapter 45 / PWENCODE Procedure	1573
Overview: PWENCODE Procedure	1573
Concepts: PWENCODE Procedure	1574
Syntax: PWENCODE Procedure	1576
Examples: PWENCODE Procedure	1578
Chapter 46 / PYTHON Procedure	1583
Overview: PYTHON Procedure	1584
Concepts: PYTHON Procedure	1584
Syntax: PYTHON Procedure	1588
Usage: PYTHON Procedure	1592
Examples: PYTHON Procedure	1599
Chapter 47 / QDEVICE Procedure	1617
Overview: QDEVICE Procedure	1618
Syntax: QDEVICE Procedure	1618
Usage: QDEVICE Procedure	1636
Examples: QDEVICE Procedure	1648
Chapter 48 / R Procedure	1663
Overview: R Procedure	1664
Concepts: R Procedure	1665

Syntax: R Procedure	1667
Usage: R Procedure	1669
Examples: R Procedure	1674
Chapter 49 / RANK Procedure	1701
Overview: RANK Procedure	1702
Concepts: RANK Procedure	1703
Syntax: RANK Procedure	1708
Results: RANK Procedure	1722
Examples: RANK Procedure	1722
References	1731
Chapter 50 / REGISTRY Procedure	1733
Overview: REGISTRY Procedure	1733
Syntax: REGISTRY Procedure	1734
Usage: REGISTRY Procedure	1740
Examples: REGISTRY Procedure	1743
Chapter 51 / REPORT Procedure	1751
Overview: REPORT Procedure	1752
Concepts: REPORT Procedure	1758
Syntax: REPORT Procedure	1774
Usage: REPORT Procedure	1833
Results: REPORT Procedure	1850
Examples: REPORT Procedure	1862
Chapter 52 / S3 Procedure	1923
Overview: S3 Procedure	1924
Concepts: S3 Procedure	1924
Syntax: S3 Procedure	1929
Examples: S3 Procedure	1948
Chapter 53 / SCAPROC Procedure	1957
Overview: SCAPROC Procedure	1957
Concepts: SCAPROC Procedure	1958
Syntax: SCAPROC Procedure	1959
Results: SCAPROC Procedure	1962
Examples: SCAPROC Procedure	1966
Chapter 54 / SCOREACCEL Procedure	1971
Overview: SCOREACCEL Procedure	1972
Concepts: SCOREACCEL Procedure	1973
Syntax: SCOREACCEL Procedure	1977
Examples: SCOREACCEL Procedure	2013
Chapter 55 / SORT Procedure	2039
Overview: SORT Procedure	2040
Concepts: SORT Procedure	2042
Syntax: SORT Procedure	2049
Usage: SORT Procedure	2076
Results: SORT Procedure	2079
Examples: SORT Procedure	2080

Chapter 56 / SQL Procedure	2099
A Brief Overview	2099
Example: Using the SQL Procedure	2100
Chapter 57 / SQOOP Procedure	2101
Overview: SQOOP Procedure	2101
Syntax: SQOOP Procedure	2102
Usage: SQOOP Procedure	2105
Examples: SQOOP Procedure	2107
Chapter 58 / STANDARD Procedure	2109
Overview: STANDARD Procedure	2110
Syntax: STANDARD Procedure	2112
Usage: STANDARD Procedure	2129
Results: STANDARD Procedure	2130
Examples: STANDARD Procedure	2131
Chapter 59 / STREAM Procedure	2139
Overview: STREAM Procedure	2139
Concepts: STREAM Procedure	2140
Syntax: STREAM Procedure	2142
Usage: STREAM Procedure	2145
Chapter 60 / SUMMARY Procedure	2153
Overview: SUMMARY Procedure	2153
Syntax: SUMMARY Procedure	2154
Chapter 61 / TABULATE Procedure	2207
Overview: TABULATE Procedure	2208
Concepts: TABULATE Procedure	2211
Syntax: TABULATE Procedure	2214
Usage: TABULATE Procedure	2271
Results: TABULATE Procedure	2295
Examples: TABULATE Procedure	2306
References	2369
Chapter 62 / TRANSPOSE Procedure	2371
Overview: TRANSPOSE Procedure	2372
Syntax: TRANSPOSE Procedure	2375
Results: TRANSPOSE Procedure	2393
Examples: TRANSPOSE Procedure	2395
Chapter 63 / XSL Procedure	2411
Overview: XSL Procedure	2411
Syntax: XSL Procedure	2412
Examples: XSL Procedure	2415

PART 3 **Appendixes** **2423**

Appendix 1 / SAS Elementary Statistics Procedures	2425
Overview of SAS Elementary Statistics Procedures	2426
Keywords and Formulas	2426
Statistical Background	2436
Appendix 2 / Raw Data and DATA Steps for Base SAS Procedures	2469
Overview of Raw Data and DATA Steps for Base SAS Procedures	2470
CARSURVEY	2471
CENSUS	2472
CHARITY	2473
CONTROL Library	2475
CUSTOMER_RESPONSE	2506
DJIA	2508
EDUCATION	2509
EMPDATA	2510
ENERGY	2512
EXP Library	2513
EXPREV	2514
GROC	2516
MATCH_11	2517
PROCLIB.DELAY	2519
PROCLIB.EMP95	2520
PROCLIB.EMP96	2521
PROCLIB.INTERNAT	2522
PROCLIB.LAKES	2522
PROCLIB.MARCH	2523
PROCLIB.PAYLIST2	2524
PROCLIB.PAYROLL	2525
PROCLIB.PAYROLL2	2528
PROCLIB.SCHEDULE	2528
PROCLIB.STAFF	2531
PROCLIB.STAFF2	2534
PROCLIB.SUPERV	2535
RADIO	2535
SALES	2548
Appendix 3 / UNICODE License	2549
UNICODE LICENSE V3	2549

Syntax Conventions for the SAS Language

Overview of Syntax Conventions for the SAS Language

SAS uses standard conventions in the documentation of syntax for SAS language elements. These conventions enable you to easily identify the components of SAS syntax. The conventions can be divided into these parts:

- syntax components
- style conventions
- special characters
- references to SAS libraries and external files

Syntax Components

The components of the syntax for most language elements include a keyword and arguments. For some language elements, only a keyword is necessary. For other language elements, the keyword is followed by an equal sign (=). The syntax for arguments has multiple forms in order to demonstrate the syntax of multiple arguments, with and without punctuation.

keyword

specifies the name of the SAS language element that you use when you write your program. Keyword is a literal that is usually the first word in the syntax. In a CALL routine, the first two words are keywords.

In these examples of SAS syntax, the keywords are bold:

CHAR (*string, position*)

CALL RANBIN (*seed, n, p, x*);

ALTER (*alter-password*)

BEST *w*.

REMOVE <*data-set-name*>

In this example, the first two words of the CALL routine are the keywords:

CALL RANBIN(*seed, n, p, x*)

The syntax of some SAS statements consists of a single keyword without arguments:

DO;

... SAS code ...

END;

Some system options require that one of two keyword values be specified:

DUPLEX | NODUPLEX

Some procedure statements have multiple keywords throughout the statement syntax:

CREATE <UNIQUE> **INDEX** *index-name* **ON** *table-name* (*column-1* <
column-2, ...>)

argument

specifies a numeric or character constant, variable, or expression. Arguments follow the keyword or an equal sign after the keyword. The arguments are used by SAS to process the language element. Arguments can be required or optional. In the syntax, optional arguments are enclosed in angle brackets (< >).

In this example, *string* and *position* follow the keyword CHAR. These arguments are required arguments for the CHAR function:

CHAR (*string, position*)

Each argument has a value. In this example of SAS code, the argument *string* has a value of 'summer', and the argument *position* has a value of 4:

```
x=char('summer', 4);
```

In this example, *string* and *substring* are required arguments, whereas *modifiers* and *startpos* are optional.

FIND(*string, substring* <, *modifiers*> <, *startpos*>

argument(s)

specifies that one argument is required and that multiple arguments are allowed. Separate arguments with a space. Punctuation, such as a comma (,) is not required between arguments.

The MISSING statement is an example of this form of multiple arguments:

MISSING *character(s)*;

<LITERAL_ARGUMENT> *argument-1* <<LITERAL_ARGUMENT> *argument-2 ...* >

specifies that one argument is required and that a literal argument can be associated with the argument. You can specify multiple literals and argument

pairs. No punctuation is required between the literal and argument pairs. The ellipsis (...) indicates that additional literals and arguments are allowed.

The BY statement is an example of this argument:

BY <DESCENDING> *variable-1* <<DESCENDING> *variable-2* ...>;

argument-1 <*options*> <*argument-2* <*options*> ...>

specifies that one argument is required and that one or more options can be associated with the argument. You can specify multiple arguments and associated options. No punctuation is required between the argument and the option. The ellipsis (...) indicates that additional arguments with an associated option are allowed.

The FORMAT procedure PICTURE statement is an example of this form of multiple arguments:

PICTURE name <(format-options)>
<value-range-set-1 <(picture-1-options)>
<value-range-set-2 <(picture-2-options)> ...>;

argument-1=value-1 <*argument-2*=value-2 ...>

specifies that the argument must be assigned a value and that you can specify multiple arguments. The ellipsis (...) indicates that additional arguments are allowed. No punctuation is required between arguments.

The LABEL statement is an example of this form of multiple arguments:

LABEL *variable-1*=label-1 <*variable-2*=label-2 ...>;

argument-1 <, *argument-2*, ...>

specifies that one argument is required and that you can specify multiple arguments that are separated by a comma or other punctuation. The ellipsis (...) indicates a continuation of the arguments, separated by a comma. Both forms are used in the SAS documentation.

Here are examples of this form of multiple arguments:

AUTHPROVIDERDOMAIN (provider-1:domain-1 <, provider-2:domain-2, ...>
INTO :macro-variable-specification-1 <, :macro-variable-specification-2, ...>

Note: In most cases, example code in SAS documentation is written in lowercase with a monospace font. You can use uppercase, lowercase, or mixed case in the code that you write.

Style Conventions

The style conventions that are used in documenting SAS syntax include uppercase bold, uppercase, and italic:

UPPERCASE BOLD

identifies SAS keywords such as the names of functions or statements. In this example, the keyword ERROR is written in uppercase bold:

ERROR <message>;

UPPERCASE

identifies arguments that are literals.

In this example of the CMPMODEL= system option, the literals include BOTH, CATALOG, and XML:

CMPMODEL=BOTH | CATALOG | XML |

italic

identifies arguments or values that you supply. Items in italic represent user-supplied values that are either one of the following:

- nonliteral arguments. In this example of the LINK statement, the argument *label* is a user-supplied value and therefore appears in italic:

LINK *label*;

- nonliteral values that are assigned to an argument.

In this example of the FORMAT statement, the argument DEFAULT is assigned the variable *default-format*:

FORMAT *variable(s)* <format> <DEFAULT = *default-format*>;

Special Characters

The syntax of SAS language elements can contain the following special characters:

=

an equal sign identifies a value for a literal in some language elements such as system options.

In this example of the MAPS system option, the equal sign sets the value of MAPS:

MAPS=*location-of-maps*

< >

angle brackets identify optional arguments. A required argument is not enclosed in angle brackets.

In this example of the CAT function, at least one item is required:

CAT (*item-1* <, *item-2*, ...>)

|

a vertical bar indicates that you can choose one value from a group of values. Values that are separated by the vertical bar are mutually exclusive.

In this example of the CMPMODEL= system option, you can choose only one of the arguments:

CMPMODEL=BOTH | CATALOG | XML

...

an ellipsis indicates that the argument can be repeated. If an argument and the ellipsis are enclosed in angle brackets, then the argument is optional. The repeated argument must contain punctuation if it appears before or after the argument.

In this example of the CAT function, multiple *item* arguments are allowed, and they must be separated by a comma:

CAT (*item-1* <, *item-2*, ...>)

'value' or "value"

indicates that an argument that is enclosed in single or double quotation marks must have a value that is also enclosed in single or double quotation marks.

In this example of the FOOTNOTE statement, the argument *text* is enclosed in quotation marks:

FOOTNOTE <*n*> <*ods-format-options* 'text' | "text">;

;

a semicolon indicates the end of a statement or CALL routine.

In this example, each statement ends with a semicolon:

```
data namegame;
  length color name $8;
  color = 'black';
  name = 'jack';
  game = trim(color) || name;
run;
```

References to SAS Libraries and External Files

Many SAS statements and other language elements refer to SAS libraries and external files. You can choose whether to make the reference through a logical name (a libref or fileref) or use the physical filename enclosed in quotation marks.

If you use a logical name, you typically have a choice of using a SAS statement (LIBNAME or FILENAME) or the operating environment's control language to make the reference. Several methods of referring to SAS libraries and external files are available, and some of these methods depend on your operating environment.

In the examples that use external files, SAS documentation uses the italicized phrase *file-specification*. In the examples that use SAS libraries, SAS documentation uses the italicized phrase *SAS-library* enclosed in quotation marks:

```
infile file-specification obs = 100;  
libname libref 'SAS-library';
```

PART 1

Concepts

Chapter 1	
<i>Choosing the Right Procedure</i>	3
Chapter 2	
<i>Fundamental Concepts for Using Base SAS Procedures</i>	19
Chapter 3	
<i>In-Database Processing of Base Procedures</i>	73
Chapter 4	
<i>CAS Processing of Base Procedures</i>	75
Chapter 5	
<i>Base SAS Procedures Documented in Other Publications</i>	81
Chapter 6	
<i>Base SAS Procedures That Are No Longer Supported</i>	85

Choosing the Right Procedure

Functional Categories of Base SAS Procedures	3
Report Writing	3
Statistics	4
Utilities	4
Report-Writing Procedures	5
Statistical Procedures	6
Available Statistical Procedures	6
Efficiency Issues	8
Additional Information about the Statistical Procedures	9
Utility Procedures	9
Brief Descriptions of Base SAS Procedures	12

Functional Categories of Base SAS Procedures

Report Writing

These procedures display useful information, such as data listings (detail reports), summary reports, calendars, letters, labels, multipanel reports, and graphical reports.

FREQ ¹	QDEVICE ¹	SUMMARY ¹
MEANS ¹	REPORT ¹	TABULATE ¹
PRINT	SQL ¹	

¹ These procedures produce reports and compute statistics.

Statistics

These procedures compute elementary statistical measures that include descriptive statistics based on moments, quantiles, confidence intervals, frequency counts, crosstabulations, correlations, and distribution tests. They also rank and standardize data.

CORR	REPORT	TABULATE
FREQ	SQL	UNIVARIATE
MEANS	STANDARD	
RANK	SUMMARY	

Utilities

These procedures perform the following basic utility operations:

- create, edit, sort, and transpose data sets
- create and restore transport data sets
- create user-defined formats
- provide basic file maintenance such as copy, append, and compare data sets
- deletes permanent or temporary data files from disk or tape

APPEND	FCMP	OPTSAVE
CATALOG	FEDSQL	PRINTTO
CIMPORT	FONTREG	PWENCODE
COMPARE	GROOVY	REGISTRY
CONTENTS	HADOOP	SCAPROC
COPY	IMPORT ³	SORT
CPORT	JAVAINFO	SOURCE

DATASETS	JSON	SQL ¹
DELETE	MIGRATE	TEMPLATE
DS2	OPTIONS	TRANSPPOSE ¹
EXPORT ³	OPTLOAD	TRANSTAB ²

¹ See the SAS documentation for your operating environment for a description of this procedure.

² For a description of this procedure, see the *SAS Output Delivery System: User's Guide*.

³ For a description of this procedure, see the *SAS/ACCESS Interface to PC Files: Reference for the SAS Viya Platform*.

Report-Writing Procedures

The following table lists report-writing procedures according to the type of report.

Table 1.1 Report-Writing Procedures by Task

Report Type	Procedure	Description
Detail reports	PRINT	Produces data listings quickly; can supply titles, footnotes, and column sums.
	REPORT	Offers more control and customization than PROC PRINT; can produce both column and row sums; has DATA step computation abilities.
	SQL	Combines Structured Query Language and SAS features such as formats; can manipulate data and create a SAS data set in the same step that creates the report; can produce column and row statistics; does not offer as much control over output as PROC PRINT and PROC REPORT.
Summary reports	MEANS or SUMMARY	Computes descriptive statistics for numeric variables; can produce a printed report and create an output data set.
	PRINT	Produces only one summary report; can sum the BY variables.
	QDEVICE	Produces reports about graphics devices and universal printers.

Report Type	Procedure	Description
	REPORT	Combines features of the PRINT, MEANS, and TABULATE procedures with features of the DATA step in a single report-writing tool that can produce a variety of reports; can also create an output data set.
	SQL	Computes descriptive statistics for one or more SAS data sets or DBMS tables; can produce a printed report or create a SAS data set.
	TABULATE	Produces descriptive statistics in a tabular format; can produce stub-and-banner reports (multidimensional tables with descriptive statistics); can also create an output data set.
Miscellaneous highly formatted reports		
Multipanel reports (telephone book listings)	REPORT	Produces multipanel reports.

Statistical Procedures

Available Statistical Procedures

The following table lists statistical procedures according to task. [Table A1.1 on page 2427](#) lists the most common statistics and the procedures that compute them.

Table 1.2 *Elementary Statistical Procedures by Task*

Report type	Procedure	Description
Descriptive statistics	CORR	Computes simple descriptive statistics.
	MEANS or SUMMARY	Computes descriptive statistics; can produce printed output and output data sets. By default, PROC MEANS produces printed output, and PROC SUMMARY creates an output data set.

Report type	Procedure	Description
	REPORT	Computes most of the same statistics as PROC TABULATE; allows customization of format.
	SQL	Computes descriptive statistics for data in one or more DBMS tables; can produce a printed report or create a SAS data set.
	TABULATE	Produces tabular reports for descriptive statistics; can create an output data set.
	UNIVARIATE	Computes the broadest set of descriptive statistics; can create an output data set.
Frequency and crosstabulation tables	FREQ	Produces one-way to n -way tables; reports frequency counts; computes chi-square tests; computes test and measures of association and agreement for two-way to n -way crosstabulation tables; can compute exact tests and asymptotic tests; can create output data sets.
	TABULATE	Produces one-way and two-way crosstabulation tables; can create an output data set.
	UNIVARIATE	Produces one-way frequency tables.
Correlation analysis	CORR	Computes Pearson's, Spearman's, and Kendall's correlations and partial correlations. Also, computes Hoeffding's measures of dependence (D) and Cronbach's coefficient alpha.
Distribution analysis	UNIVARIATE	Computes tests for location and tests for normality.
	FREQ	Computes a test for the binomial proportion for one-way tables; computes a goodness-of-fit test for one-way tables; computes a chi-square test of equal distribution for two-way tables.
Robust estimation	UNIVARIATE	Computes robust estimates of scale, trimmed means, and Winsorized means.
Data transformation		
Computing ranks	RANK	Computes ranks for one or more numeric variables across the observations of a SAS data set and creates an output data set; can produce normal scores or other rank scores.

Report type	Procedure	Description
Standardizing data	STANDARD	Creates an output data set that contains variables that are standardized to a given mean and standard deviation.
Low-resolution graphics ¹		
	UNIVARIATE	Produces descriptive plots such as stem-and-leaf plot, box plots, and normal probability plots.

¹ To produce high-resolution graphical reports, use SAS/GRAPH software.

Efficiency Issues

Quantiles

For a large sample size n , the calculation of quantiles, including the median, requires computing time proportional to $n\log(n)$. Therefore, a procedure, such as UNIVARIATE, that automatically calculates quantiles might require more time than other data summarization procedures. Furthermore, because data is held in memory, the procedure also requires more storage space to perform the computations. By default, the report procedures PROC MEANS, PROC SUMMARY, and PROC TABULATE require less memory because they do not automatically compute quantiles. These procedures also provide an option to use a new fixed-memory, quantiles estimation method that is usually less memory-intensive. For more information, see [“Quantiles” on page 1295](#) in the PROC MEANS documentation.

Computing Statistics for Groups of Observations

To compute statistics for several groups of observations, you can use any of the previous procedures with a BY statement to specify BY-group variables. However, BY-group processing requires that you previously sort or index the data set, which for very large data sets might require substantial computer resources. A more efficient way to compute statistics within groups without sorting is to use a CLASS statement with one of the following procedures: MEANS, SUMMARY, or TABULATE.

Additional Information about the Statistical Procedures

Appendix 1, “SAS Elementary Statistics Procedures,” on page 2425 lists standard keywords, statistical notation, and formulas for the statistics that Base SAS procedures compute frequently. The sections on the individual statistical procedures discuss the statistical concepts that are useful to interpret a procedure output.

Utility Procedures

The following table groups utility procedures according to task.

Table 1.3 *Utility Procedures by Task*

Tasks	Procedure	Description
Supply information	COMPARE	Compares the contents of two SAS data sets.
	CONTENTS	Describes the contents of a SAS library or specific library members.
	JAVAINFO	Conveys diagnostic information about the Java environment that SAS is using.
	OPTIONS	Lists the current values of all SAS system options.
	SCAPROC	Implements the SAS Code Analyzer, which captures information about input, output, and the use of macro symbols from a SAS job while it is running.
	SQL	Supplies information through dictionary tables on an individual SAS data set as well as all SAS files active in the current SAS session. Dictionary tables can also provide information about macros, titles, indexes, external files, or SAS system options.
Manage SAS system options	OPTIONS	Lists the current values of all SAS system options.

Tasks	Procedure	Description
	OPTLOAD	Reads SAS system option settings that are stored in the SAS registry or a SAS data set.
	OPTSAVE	Saves SAS system option settings to the SAS registry or a SAS data set.
	FONTREG	Adds system fonts to the SAS registry.
	FORMAT	Creates user-defined formats to display and print data.
	PRINTTO	Routes procedure output to a file, a SAS catalog entry, or a printer; can also redirect the SAS log to a file.
	TEMPLATE ²	Customizes ODS output.
Create, browse, and edit data	FCMP	Enables creation, testing, and storage of SAS functions and subroutines before they are used in other SAS procedures.
	SQL	Creates SAS data sets using Structured Query Language and SAS features.
	FORMAT	Creates user-defined informats to read data and user-defined formats to display data.
	SORT	Sorts SAS data sets by one or more variables.
	SQL	Sorts SAS data sets by one or more variables.
	STREAM	Enables you to process an input stream that consists of arbitrary text that can contain SAS macro specifications. It can expand macro code and store it in a file.
	TRANSPOSE	Transforms SAS data sets so that observations become variables and variables become observations.
	TRANTAB ³	Creates, edits, and displays customized translation tables.
	XSL	Transforms an XML document into another format, such as HTML, text, or another XML document type.

Tasks	Procedure	Description
Manage SAS files	APPEND	Appends one SAS data set to the end of another.
	CATALOG	Manages SAS catalog entries.
	CIMPORT	Restores a transport sequential file that PROC CPORT creates (usually in another operating environment) to its original form as a SAS catalog, a SAS data set, or a SAS library.
	CONVERT ¹	Converts BMDP system files, OSIRIS system files, and SPSS portable files to SAS data sets.
	COPY	Copies a SAS library or specific members of the library.
	CPORT	Converts a SAS catalog, a SAS data set, or a SAS library to a transport sequential file that PROC CIMPORT can restore (usually in another operating environment) to its original form.
	DATASETS	Manages SAS files.
	DELETE	Deletes permanent or temporary SAS files.
	EXPORT ⁴	Reads data from a SAS data set and writes them to an external data source.
	IMPORT ⁴	Reads data from an external data source and writes them to a SAS data set.
	JSON	Reads data from a SAS data set and writes it to an external file in JSON representation.
	MIGRATE	Migrates members in a SAS library forward to the most current release of SAS.
	PROTO	Enables registration, in batch mode, of external functions that are written in the C or C++ programming languages.
	REGISTRY	Imports registry information to the USER portion of the SAS registry.
	RELEASE ¹	Releases unused space at the end of a disk data set under the z/OS environment.

Tasks	Procedure	Description
	SOURCE ¹	Provides an easy way to back up and process source library data sets.
	SQL	Concatenates SAS data sets.
	DS2	Submits DS2 language statements from a Base SAS session.
	FEDSQL	Submits FedSQL language statements from a Base SAS session.
	GROOVY	Enables SAS code to execute Groovy code on the Java Virtual Machine (JVM).
	HADOOP	Enables SAS to run Apache Hadoop code against Hadoop data.
	PWENCODE	Encodes passwords for use in SAS programs.

¹ See the SAS documentation for your operating environment for a description of these procedures.

² For a description of this procedure, see the *SAS Output Delivery System: User's Guide*.

³ For a description of this procedure, see the *SAS National Language Support (NLS): Reference Guide*.

⁴ For a description of this procedure, see the *SAS/ACCESS Interface to PC Files: Reference for the SAS Viya Platform*.

Brief Descriptions of Base SAS Procedures

APPEND procedure

adds observations from one SAS data set to the end of another SAS data set.

CATALOG procedure

manages entries in SAS catalogs. PROC CATALOG is an interactive, non-windowing procedure that enables you to display the contents of a catalog; copy an entire catalog or specific entries in a catalog; and rename, exchange, or delete entries in a catalog.

CIMPORT procedure

restores a transport file created by the CPORT procedure to its original form (a SAS library, catalog, or data set) in the format appropriate to the operating environment. Coupled with the CPORT procedure, PROC CIMPORT enables you to move SAS libraries, catalogs, and data sets from one operating environment to another.

COMPARE procedure

compares the contents of two SAS data sets. You can also use PROC COMPARE to compare the values of different variables within a single data set. PROC COMPARE produces a variety of reports on the comparisons that it performs.

CONTENTS procedure

prints descriptions of the contents of one or more files in a SAS library.

COPY procedure

copies an entire SAS library or specific members of the library. You can limit processing to specific types of library members.

CORR procedure

computes Pearson product-moment and weighted product-moment correlation coefficients between variables and descriptive statistics for these variables. In addition, PROC CORR can compute three nonparametric measures of association (Spearman's rank-order correlation, Kendall's tau-b, and Hoeffding's measure of dependence, D), partial correlations (Pearson's partial correlation, Spearman's partial rank-order correlation, and Kendall's partial tau-b), and Cronbach's coefficient alpha. For more information, see *Base SAS Procedures Guide: Statistical Procedures*.

CPORT procedure

writes SAS libraries, data sets, and catalogs in a special format called a transport file. Coupled with the CIMPORT procedure, PROC CPORT enables you to move SAS libraries, data sets, and catalogs from one operating environment to another.

DATASETS procedure

lists, copies, renames, and deletes SAS files and SAS generation groups; manages indexes; and appends SAS data sets in a SAS library. The procedure provides all the capabilities of the APPEND, CONTENTS, and COPY procedures. You can also modify variables within data sets; manage data set attributes, such as labels and passwords; or Create and Delete integrity constraints.

DELETE procedure

deletes SAS files from the disk or tape on which it is stored.

DOCUMENT procedure

manipulates procedure output that is stored in ODS documents. PROC DOCUMENT enables a user to browse and edit output objects and hierarchies, and to replay them to any supported ODS output format. For more information, see *SAS Output Delivery System: User's Guide*.

DS2 procedure

enables you to submit DS2 language statements from a Base SAS session.

DSTODS2 procedure

enables you to translate a subset of your SAS DATA step code into DS2 code. Then you can revise your program to take advantage of DS2 features and submit your program using PROC DS2.

EXPORT procedure

reads data from a SAS data set and writes it to an external data source.

FCMP procedure

enables you to create, test, and store SAS functions and subroutines before you use them in other SAS procedures. PROC FCMP accepts slight variations of

DATA step statements. Most features of the SAS programming language can be used in functions and subroutines that are processed by PROC FCMP.

FEDSQL procedure

enables you to submit FedSQL language statements from a Base SAS session.

FMTC2ITM procedure

converts one or more format catalogs into a single item store that can be made available to CAS.

FONTREG procedure

adds system fonts to the SAS registry.

FORMAT procedure

creates user-defined informats and formats for character or numeric variables. PROC FORMAT also prints the contents of a format library, creates a control data set to write other informats or formats, and reads a control data set to create informats or formats.

FREQ procedure

produces one-way to n -way frequency tables and reports frequency counts. PROC FREQ can compute chi-square tests for one-way to n -way tables; for tests and measures of association and of agreement for two-way to n -way crosstabulation tables; risks and risk difference for 2×2 tables; trends tests; and Cochran-Mantel-Haenszel statistics. You can also create output data sets. For more information, see *Base SAS Procedures Guide: Statistical Procedures*.

GROOVY procedure

enables SAS code to execute Groovy code on the Java Virtual Machine (JVM).

HADOOP procedure

enables SAS to run Apache Hadoop code against Hadoop data.

HTTP procedure

issues Hypertext Transfer Protocol (HTTP) requests.

IMPORT procedure

reads data from an external data source and writes them to a SAS data set.

JAVAINFO procedure

conveys diagnostic information to the user about the Java environment that SAS is using. The diagnostic information can be used to confirm that the SAS Java environment has been configured correctly and can be helpful when reporting problems to SAS technical support.

JSON procedure

reads data from a SAS data set and writes it to an external file in JSON representation.

LUA procedure

enables you to run statements from the Lua programming language within SAS code. You can submit Lua statements from an external Lua script, or enter Lua statements directly in SAS code.

MEANS procedure

computes descriptive statistics for numeric variables across all observations and within groups of observations. You can also create an output data set that contains specific statistics and identifies minimum and maximum values for groups of observations.

MIGRATE procedure

migrates members in a SAS library forward to the most current release of SAS. The migration must occur within the same engine family; for example, V6, V7, or V8 can migrate to V9, but V6TAPE must migrate to V9TAPE.

OPTIONS procedure

lists the current values of all SAS system options.

OPTLOAD procedure

reads SAS system option settings from the SAS registry or a SAS data set, and puts them into effect.

OPTSAVE procedure

saves SAS system option settings to the SAS registry or a SAS data set.

PRINT procedure

prints the observations in a SAS data set, using all or some of the variables. PROC PRINT can also print totals and subtotals for numeric variables.

PRINTTO procedure

defines destinations for SAS procedure output and the SAS log.

PROTO procedure

enables you to register, in batch mode, external functions that are written in the C or C++ programming languages. You can use these functions in SAS as well as in C-language structures and types. After these functions are registered in PROC PROTO, they can be called from any SAS function or subroutine that is declared in the FCMP procedure. After registration, they can also be called from any SAS function, subroutine, or method block that is declared in the COMPILE procedure.

PWENCODE procedure

encodes passwords for use in SAS programs.

QDEVICE procedure

produces reports about graphics devices and universal printers.

RANK procedure

computes ranks for one or more numeric variables across the observations of a SAS data set. The ranks are written to a new SAS data set. Alternatively, PROC RANK produces normal scores or other rank scores.

REGISTRY procedure

imports registry information into the USER portion of the SAS registry.

REPORT procedure

combines features of the PRINT, MEANS, and TABULATE procedures with features of the DATA step in a single report-writing tool that can produce both detail and summary reports.

SCAPROC procedure

implements the SAS Code Analyzer, which captures information about input, output, and the use of macro symbols from a SAS job while it is running.

SCOREACCEL procedure

provides an interface to the CAS server for DATA step and DS2 model publishing and scoring.

SORT procedure

sorts observations in a SAS data set by one or more variables. PROC SORT stores the resulting sorted observations in a new SAS data set or replaces the original data set.

SOURCE procedure

provides an easy way to back up and process source library data sets. For more information, see the SAS documentation for your operating environment.

SQL procedure

implements a subset of the Structured Query Language (SQL) for use in SAS. SQL is a standardized, widely used language that retrieves and updates data in SAS data sets, SQL views, and DBMS tables, as well as views based on those tables. PROC SQL can also create tables and views, summaries, statistics, and reports and perform utility functions such as sorting and concatenating. For more information, see [SAS SQL Procedure User's Guide](#).

SQOOP procedure

allows access to Apache Sqoop using options to allow data transfer between a database and HDFS.

STANDARD procedure

standardizes some or all of the variables in a SAS data set to a given mean and standard deviation and produces a new SAS data set that contains the standardized values.

STREAM procedure

enables you to process an input stream that consists of arbitrary text that can contain SAS macro specifications. It can expand macro code and store it in a file.

SUMMARY procedure

computes descriptive statistics for the variables in a SAS data set across all observations and within groups of observations, and writes the results to a new SAS data set.

TABULATE procedure

displays descriptive statistics in tabular form. The value in each table cell is calculated from the variables and statistics that define the pages, rows, and columns of the table. The statistic associated with each cell is calculated on values from all observations in that category. You can write the results to a SAS data set.

TEMPLATE procedure

customizes ODS output for an entire SAS job or a single ODS output object. For more information, see *SAS Output Delivery System: User's Guide*.

TRANSPPOSE procedure

transposes a data set that changes observations into variables and vice versa.

TRANTAB procedure

creates, edits, and displays customized translation tables. For more information, see *SAS National Language Support (NLS): Reference Guide*.

UNIVARIATE procedure

computes descriptive statistics (including quantiles), confidence intervals, and robust estimates for numeric variables. Provides detail on the distribution of numeric variables, which include tests for normality, plots to illustrate the

distribution, frequency tables, and tests of location. For more information, see *Base SAS Procedures Guide: Statistical Procedures*.

XSL procedure

transforms an XML document into another format, such as HTML, text, or another XML document type.

Fundamental Concepts for Using Base SAS Procedures

Language Concepts	20
Temporary and Permanent SAS Data Sets	20
SAS System Options	21
Data Set Options	22
Statements with the Same Function in Multiple Procedures	23
Global Statements	23
AES Encryption	24
Procedure Concepts	24
Input Data Sets	24
Threaded Processing for Base SAS Procedures	24
Controlling the Order of Data Values	25
RUN-Group Processing	55
Creating Titles That Contain BY-Group Information	55
Shortcuts for Specifying Lists of Variable Names	61
Formatted Values	62
Processing All the Data Sets in a Library	68
Statistic Descriptions	68
Computational Requirements for Statistics	70
Output Delivery System	71

Language Concepts

Temporary and Permanent SAS Data Sets

Naming SAS Data Sets

SAS data sets can have a one-level name or a two-level name. Typically, names of temporary SAS data sets have only one level and are stored in the WORK library. The WORK library is defined automatically at the beginning of the SAS session and is deleted automatically at the end of the SAS session. Procedures assume that SAS data sets that are specified with a one-level name are to be read from or written to the WORK library. To indicate otherwise, you specify a USER library. For more information, see [“USER Library” on page 21](#). For example, the following PROC PRINT steps are equivalent. The second PROC PRINT step assumes that the DEBATE data set is in the WORK library.

```
proc print data=work.debate;  
run;  
  
proc print data=debate;  
run;
```

The SAS system options WORK=, WORKINIT, and WORKTERM affect how you work with temporary and permanent libraries. For more information, see *SAS System Options: Reference*.

Typically, two-level names represent permanent SAS data sets. A two-level name takes the form *libref.SAS-data-set*. The *libref* is a name that is temporarily associated with a SAS library. A SAS library is a collection of one or more SAS files, including SAS data sets, that are referenced and stored as a unit. In directory-based environments, a SAS library is a group of SAS files that are stored in the same folder or directory. A LIBNAME statement associates the libref with the SAS library. In the following PROC PRINT step, PROCLIB is the libref and EMP is the SAS data set within the library:

```
libname proclib 'SAS-library';  
proc print data=proclib.emp;  
run;
```

USER Library

You can use one-level names for permanent SAS data sets by specifying a USER library. You can assign a USER library with a LIBNAME statement or with the SAS system option USER=. After you specify a USER library, the procedure assumes that data sets with one-level names are in the USER library instead of the WORK library. For example, the following PROC PRINT step assumes that DEBATE is in the USER library:

```
options user='SAS-library';  
proc print data=debate;  
run;
```

Note: If you have a USER library defined, then you can still use the WORK library by specifying WORK.SAS-*data-set*.

SAS System Options

Some SAS system option settings affect procedure output. The SAS system options listed below are the options that you are most likely to use with SAS procedures:

- BYLINE | NOBYLINE
- DATE | NODATE
- DETAILS | NODETAILS
- FMterr | NOFMterr
- FORMCHAR=
- LABEL | NOLABEL
- LINESIZE=
- NUMBER | NONUMBER
- PAGENO=
- PAGESIZE=
- REPLACE | NOREPLACE
- SOURCE | NOSOURCE

For a complete description of SAS system options, see the *SAS System Options: Reference*.

Data Set Options

Most of the procedures that read data sets or create output data sets accept data set options. SAS data set options appear in parentheses after the data set specification. Here is an example:

```
proc print data=stocks(obs=25 pw=green);
```

The individual procedure chapters contain reminders that you can use data set options where it is appropriate.

SAS data set options are as follows:

ALTER=	OBS=
BUFNO=	OBSBUF=
BUFSIZE=	OUTREP=
CNTLLEV=	POINTOBS=
COMPRESS=	PW=
DLDMGACTION=	PWREQ=
DROP=	READ=
ENCODING=	RENAME=
ENCRYPT=	REPEMPTY=
FILECLOSE=	REPLACE=
FIRSTOBS=	REUSE=
GENMAX=	SORTEDBY=
GENNUM=	SPILL=
IDXNAME=	TOBSNO=
IDXWHERE=	TYPE=
IN=	WHERE=
INDEX=	WHEREUP=

KEEP=

WRITE=

LABEL=

For a complete description of SAS data set options, see the *SAS Data Set Options: Reference*.

Statements with the Same Function in Multiple Procedures

Several Base SAS statements have the same function in a number of Base SAS procedures. Some of the statements are fully documented in the *SAS DATA Step Statements: Reference*, and others are documented in the specific procedure.

Global Statements

You can use these global statements anywhere in SAS programs except after a DATALINES, CARDS, or PARMCARDS statement:

<i>comment</i>	ODS
DM	OPTIONS
ENDSAS	PAGE
FILENAME	RUN
FOOTNOTE	%RUN
%INCLUDE	SASFILE
LIBNAME	SKIP
%LIST	TITLE
LOCK	

For information about all the above statements except for the ODS statement, see [SAS Global Statements: Reference](#). For information about the ODS statement, see [“Output Delivery System” on page 71](#) and *SAS Output Delivery System: User's Guide*.

AES Encryption

SAS supports a stronger AES key generation algorithm specified with the ENCRYPT=AES2 data set option. This stronger algorithm meets newer standards requested by some SAS customers. The same key value passphrase specified by the ENCRYPTKEY= data set option can be used with either algorithm.

To access a data set that is created with AES encryption, you have to supply the encryption key value with the ENCRYPTKEY= option. If you omit the ENCRYPTKEY= key value when accessing an AES secured data set, a dialog box appears and prompts you to add the ENCRYPTKEY= key value. For more information, see [“Encryption” in SAS Programmer’s Guide: Essentials](#) and [“ENCRYPTKEY= Data Set Option” in SAS Data Set Options: Reference](#).

Procedure Concepts

Input Data Sets

Many Base SAS procedures require an input SAS data set. You specify the input SAS data set by using the DATA= option in the procedure statement, as in this example:

```
proc print data=emp;
```

If you omit the DATA= option, the procedure uses the value of the SAS system option _LAST_. The default of _LAST_ is the most recently created SAS data set in the current SAS job or session. _LAST_ is described in detail in the *SAS Data Set Options: Reference*.

Threaded Processing for Base SAS Procedures

Threaded processing enables multiple pieces of executable code to run simultaneously. Many SAS procedures, including several SAS/STAT and High-Performance Analytics (HPA) procedures, support threaded processing. However, not all SAS procedures support threaded processing. See the documentation for the procedures that you are using to determine whether they support threaded processing.

Calculated statistics can vary slightly, depending on the order in which observations are processed. Such variations are due to numerical errors that are

introduced by floating-point arithmetic, the results of which should be considered approximate and not exact. The order of observation processing can be affected by nondeterministic effects of multithreaded or parallel processing. The order of processing can also be affected by inconsistent or nondeterministic ordering of observations that are produced by a data source, such as a DBMS that delivers query results through an ACCESS engine. For more information, see [“Threading in Base SAS” in *SAS Programmer’s Guide: Essentials*](#)

The following Base SAS procedures support threaded processing:

- [Chapter 18, “DS2 Procedure,” on page 597](#)
- [Chapter 23, “FEDSQL Procedure,” on page 793](#)
- [Chapter 35, “MEANS Procedure,” on page 1227](#)
- [Chapter 51, “REPORT Procedure,” on page 1751](#)
- [Chapter 52, “S3 Procedure,” on page 1923](#)
- [Chapter 55, “SORT Procedure,” on page 2039](#)
- [Chapter 60, “SUMMARY Procedure,” on page 2153](#)
- [Chapter 61, “TABULATE Procedure,” on page 2207](#)
- [“SQL Procedure” in *SAS SQL Procedure User’s Guide*](#)

See Also

System Options

- [“CPUCOUNT= System Option” in *SAS System Options: Reference*](#)
- [“THREADS System Option” in *SAS System Options: Reference*](#)

Other Documentation

- [Support for Parallel Processing](#)
- [SAS Scalable Performance Data Server: User’s Guide](#)

Controlling the Order of Data Values

Ordering of Data Values

Procedures apply an ordering scheme for data values and certain conditions affect how procedures order data:

- operating-environment-specific collating sequences
- the BY statement

- a single classification variable
- multiple classification variables
- formats
- the ORDER= option
- supplemental ordering options

Examples for ordering data use the Sasuser.Houses data set.

```
data sasuser.houses;
input style $ 1-9 sqfeet 10-13 bedrooms 15 baths 17-19 street $ 21-36 price 38-44;
datalines;
RANCH      1250 2 1.0 Sheppart Avenue    64000
SPLIT      1190 1 1.0 Rand Street        65850
CONDO      1400 2 1.5 Market Street      80050
TWOSTORY   1810 4 3.0 Garris Street      107250
RANCH      1500 3 3.0 Kemble Avenue      86650
SPLIT      1615 4 3.0 West Drive         94450
SPLIT      1305 3 1.5 Graham Avenue     73650
CONDO      1390 3 2.5 Hampshire Avenue  79650
TWOSTORY   1040 2 1.0 Sanders Road       55850
CONDO      2105 4 2.5 Jeans Avenue      127150
RANCH      1535 3 3.0 State Highway      89100
TWOSTORY   1240 2 1.0 Fairbanks Circle   69250
RANCH       720 1 1.0 Nicholson Drive    34550
TWOSTORY   1745 4 2.5 Highland Road     102950
CONDO      1860 2 2.0 Arcata Avenue     110700
run;

proc datasets lib=sasuser memtype=data;
    modify houses;
    attrib price format=dollar8.;
run;

proc print data=sasuser.houses;
run;
```


Figure 2.1 The Sasuser.Houses Data Set

Obs	style	sqfeet	bedrooms	baths	street	price
1	RANCH	1250	2	1.0	Sheppard Avenue	\$64,000
2	SPLIT	1190	1	1.0	Rand Street	\$65,850
3	CONDO	1400	2	1.5	Market Street	\$80,050
4	TWOSTORY	1810	4	3.0	Garris Street	\$107,250
5	RANCH	1500	3	3.0	Kemble Avenue	\$86,650
6	SPLIT	1615	4	3.0	West Drive	\$94,450
7	SPLIT	1305	3	1.5	Graham Avenue	\$73,650
8	CONDO	1390	3	2.5	Hampshire Avenue	\$79,350
9	TWOSTORY	1040	2	1.0	Sanders Road	\$55,850
10	CONDO	2105	4	2.5	Jeans Avenue	\$127,150
11	RANCH	1535	3	3.0	State Highway	\$89,100
12	TWOSTORY	1240	2	1.0	Fairbanks Circle	\$69,250
13	RANCH	720	1	1.0	Nicholson Drive	\$34,550
14	TWOSTORY	1745	4	2.5	Highland Road	\$102,950
15	CONDO	1860	2	2.0	Arcata Avenue	\$110,700

Order Data Using the BY Statement

To order data by one or more variables, you must first use the SORT procedure. After the data is sorted, you use the same variables in a procedure BY statement to indicate how the data is sorted. The procedure subsets the data into BY groups based on the BY variables.

```
proc sort data=sasuser.houses out=houses;
  by style;
run;
proc freq data=houses;
  by style;
  tables bedrooms /nopercnt;
run;
```

Here are the first two BY groups ordered by the Style variable for the house styles:

Figure 2.2 The First of Two BY Groups in Ascending Order

The FREQ Procedure

style=CONDO

bedrooms	Frequency	Cumulative Frequency
2	2	2
3	1	3
4	1	4

Figure 2.3 The Second of Two BY Groups in Ascending Order

The FREQ Procedure

style=RANCH

bedrooms	Frequency	Cumulative Frequency
1	1	1
2	1	2
3	2	4

By default, the SORT procedure orders BY groups in ascending order. To reverse the order, you use the DESCENDING option in the BY statement in PROC SORT and in subsequent procedures that process the data set.

```
proc sort data=sasuser.houses out=houses;
    by descending style;
run;
proc freq data=houses;
    by descending style;
    tables bedrooms /nopercnt;
run;
```

Figure 2.4 First of Two BY Groups in Descending Order

The FREQ Procedure

style=TWOSTORY

bedrooms	Frequency	Cumulative Frequency
2	2	2
4	2	4

Figure 2.5 Second of Two BY Groups in Descending Order

The FREQ Procedure

style=SPLIT

bedrooms	Frequency	Cumulative Frequency
1	1	1
3	1	2
4	1	3

The BY statement has another option, NOTSORTED. The NOTSORTED option is useful when you want to list a variable in a BY statement whose values are grouped together, but are not sorted in either ascending or descending order.

Order Data Using a Single Classification Variable

Classification variables organize data into groups that are meaningful for analysis. The values are grouped and ordered after all of the data values in a data set or BY group have been read by the procedure.

The following table lists some of the procedures and their statements that define classification variables:

Procedure	Statement
FREQ	TABLES
MEANS	CLASS

Procedure	Statement
REPORT	DEFINE use options GROUP, ORDER, or ACROSS
SUMMARY	CLASS
TABULATE	CLASS

For most procedures, the default ordering scheme is ascending for single classification variables with no formats or ordering options:

```
proc means data=sasuser.houses nway mean;
  class style;
  var sqfeet;
run;
```

Figure 2.6 The Default Ascending Order for a Single Classification Variable

The MEANS Procedure		
Analysis Variable : sqfeet Square footage		
Style of homes	N Obs	Mean
CONDO	4	1688.75
RANCH	4	1251.25
SPLIT	3	1370.00
TWOSTORY	4	1458.75

For information about the default data ordering behavior for PROC REPORT, see [“Order Data Using the ORDER= Option” on page 42](#).

Order Data Using Multiple Classification Variables

Only one variable can be considered the higher-order grouping variable when you use multiple classification variables to create subgroups of data. All other variables are used to create subgroups of the immediate preceding higher-order variable.

Here is a list of statements, by procedure, that specifies the higher-order variable. The first variable in the procedure statement is the higher-order variable. In all examples in the table, A is the higher-order variable.

Procedure	Statement That Specifies the Higher-Order Variable	Example
FREQ	TABLES	<pre>proc freq; tables A * B * C * D; run;</pre>
MEANS	CLASS	<pre>proc means; class A B C D; run;</pre>
REPORT	COLUMN	<pre>proc report; column A B C D; define C / group; define A / group; define B / group; define D / group; run;</pre>
SUMMARY	CLASS	<pre>proc summary; class A B C D; run;</pre>
TABULATE	TABLE	<pre>proc tabulate; class D C B A; table A, B, C * D; run;</pre>

The ordering scheme is determined first by developing a main order for each class variable, for the entire data set or BY group. The main order is then applied to each subgroup of the hierarchy. The order does not change from one class subgroup to the next.

Consider this example:

```
proc tabulate data=sasuser.houses format=3. noseps;
  class style bedrooms;
  table style*bedrooms, n / rts=23;
run;
```

The main order for Style and Bedrooms is determined separately. Style is the higher-order class variable. Bedrooms forms subgroups of each value of Style. The order for Bedrooms is not determined again for each value of Style. Instead, the order is taken from the main order that is determined before generating subgroups.

When no ORDER= option is specified in the CLASS statement, the data is ordered by using unformatted values, which results in the same order as the SORT procedure. The order for Style is ascending alphabetically, and the order for Bedrooms is ascending numerically:

Figure 2.7 Order Using No ORDER= Option

		N
style	bedrooms	
CONDO	2	2
	3	1
	4	1
RANCH	1	1
	2	1
	3	2
SPLIT	1	1
	3	1
	4	1
TWO STORY	2	2
	4	2

Consider this example where the order also considers frequency counts:

```
proc tabulate data=sasuser.houses format=3. noseps order=freq;
  class style bedrooms;
  table style*bedrooms, n / rts=23;
run;
```

For PROC TABULATE, if the frequency count is the same for multiple variables, then the order uses the order in which the data was read by the procedure. When PROC TABULATE reads the data set, the frequency count for Ranch is 4, Split is 3, Condo is 4, and TwoStory is 4. Therefore, the order of the higher-order variable, Style, is Ranch, Condo, TwoStory, and Split.

The order for bedrooms is then determined. Two bedrooms has a count of 5, four bedrooms has a count of 4, three bedrooms has a count of 4, and one bedroom has a count of 2. The order for bedrooms is 2, 4, 3, 1. Here is the output:

Figure 2.8 *The Order of Values Using the Order*

		N
Style of homes	Number of bedrooms	
RANCH	2	1
	3	2
	1	1
CONDO	2	2
	4	1
	3	1
TWO STORY	2	2
	4	2
SPLIT	4	1
	3	1
	1	1

The order becomes obvious when you add the PRINTMISS option to the TABLE statement to show the frequency count:

Figure 2.9 The Order with Frequency Counts

		N
Style of homes	Number of bedrooms	
RANCH	2	1
	4	.
	3	2
	1	1
CONDO	2	2
	4	1
	3	1
	1	.
TWO STORY	2	2
	4	2
	3	.
	1	.
SPLIT	2	.
	4	1
	3	1
	1	1

Order Data Using Formats: Procedure Default Behavior

The FREQ, MEANS, SUMMARY, and TABULATE procedures default to order class variable values in ascending order by the actual value in the SAS data set and not their formatted values.

The REPORT procedure default order is ascending order based on the formatted values of the order, group, or across variable.

The following table summarizes the default ordering schemes for procedures when formats are applied to classification variables:

Procedure	Variable Type	Format Specified	Ordered By
FREQ	Numeric or Character	Yes or No	Actual value

Procedure	Variable Type	Format Specified	Ordered By
MEANS	Numeric or Character	Yes or No	Actual value
REPORT	Numeric	Yes or No	Formatted value ¹
	Character	Yes	Formatted value
		No	Actual value
SUMMARY	Numeric or Character	Yes or No	Actual value
TABULATE	Numeric or Character	Yes or No	Actual value

¹ BESTw.d is the default format if no format is assigned.

Order Data Using Formats: Using the Lowest Actual Value

When you format multiple values of a class variable and use ORDER=INTERNAL, the ordering of data uses the actual values from the data set and not formatted values. The value that applies to a particular class level is the lowest actual value that was encountered in the data set for the format range. This occurs whether you are producing printed output or an output data set.

In this example, GROUP B appears before GROUP A because the lowest actual value encountered for GROUP B is 1. For GROUP A, the lowest actual value is 3. The lowest possible actual value that could be in GROUP A is 0, but 0 does not exist in the data.

```
proc format;
    value numf 0,3,4='GROUP A'
              1,2='GROUP B';

proc report data=sasuser.houses;
    column bedrooms;
    define bedrooms / group format=numf. order=internal;
run;
```

Figure 2.10 Ordering by the Lowest Value in a Format Range

The SAS System

bedrooms
GROUP B
GROUP A

Another situation that uses the lowest actual value occurs when a format contains groups or ranges that are independent from one another.

In this example, for DEPT=PET, the value OTHER appears last in the sequence. It appears first for DEPT=PLANT. This is because the ordering sequence for ID is determined before subgroups are created. ID=199 and ID=299 have the same format, OTHER. Because only the formatted value OTHER was established in the ordering for ID, OTHER is ordered first for DEPT=PLANT.

```
data sample;

    length dept $ 5;
    input dept id;
    datalines;
PET    100
PET    110
PET    120
PET    199
PLANT  200
PLANT  210
PLANT  220
PLANT  299
;

proc format;
    value idfmt
        100='CAT'
        110='DOG'
        120='FISH'
        199='OTHER'
        200='CACTUS'
        210='IVY'
        220='FERN'
        299='OTHER';

proc tabulate data=sample noseps;
    class dept id;
    table dept*id, n;
    format id idfmt.;
run;
```

Figure 2.11 Ordering by the Lowest Actual Value

The SAS System

		N
dept	id	
PET	CAT	1
	DOG	1
	FISH	1
	OTHER	1
PLANT	OTHER	1
	CACTUS	1
	IVY	1
	FERN	1

Order Data Using Formats: Missing Values

If missing values exist in the data and in a format range, they can have one of two possible effects on the output, depending on the procedure and options that you specify:

- 1 The entire format group is treated as invalid.
- 2 The entire format range has the lowest internal value, possibly resulting in the format range appearing first in an ordering scheme.

Most procedures that use class variables provide the **MISSING** option, which enables you to specify whether missing values are to be considered valid class levels. **PROC FREQ** has the option **MISSPRINT**, which displays missing class levels but does not use them in calculating statistics.

The **MEANS**, **REPORT**, **SUMMARY**, and **TABULATE** procedures classify missing values as valid or invalid before formats are applied. If you do not specify the **MISSING** option, and you use a format range that groups nonmissing class levels with missing class levels, then the nonmissing class levels are treated as valid, whereas the missing class levels are not valid.

PROC FREQ applies formats before classifying missing values as valid or invalid. If **MISSING** or **MISSPRINT** is not used and a format range groups nonmissing class levels with missing class levels, then both the nonmissing and missing class levels are considered invalid. The following example demonstrates the difference in effect between **PROC FREQ** and **PROC REPORT**:

```
proc format;
  value bedfmt 1='ONE' 2='TWO' other='OTHER';
```

```
data houses;
    set sasuser.houses end=last;
    output;
    if last then do;
        bedrooms=.;
        output;
    end;
    format bedrooms bedfmt.;
run;

proc print data=houses;
    title "PROC PRINT";
    title2 "WORK.HOUSES";
    var bedrooms;
    format bedrooms;
run;

proc freq data=houses;
    title1 "PROC FREQ";
    title2 "Without MISSING Specified";
    tables bedrooms / nocum nopercnt;
run;

proc report data=houses;
    title1 "PROC REPORT";
    title2 "Without MISSING Specified";
    column bedrooms n;
    define bedrooms /group width=8;
run;
```

Output 2.1 Order of Data Compared in PROC FREQ and PROC REPORT

**PROC PRINT
WORK.HOUSES**

Obs	bedrooms
1	2
2	1
3	2
4	4
5	3
6	4
7	3
8	3
9	2
10	4
11	3
12	2
13	1
14	4
15	2
16	.

**PROC FREQ
Without MISSING Specified**

The FREQ Procedure

Number of bedrooms	
bedrooms	Frequency
ONE	2
TWO	5
Frequency Missing = 9	

PROC REPORT Without MISSING Specified

Number of bedrooms	n
ONE	2
OTHER	8
TWO	5

PROC FREQ does not include the formatted class level OTHER, where PROC REPORT does. This is because PROC FREQ applied the BEDFMT. format to the Bedrooms variable before classifying missing values as invalid. PROC REPORT classifies missing values as invalid before applying the format. This allows the nonmissing class levels that would normally be grouped with the missing class levels to be treated as valid.

You can verify this by observing the frequency counts. A total of 16 observations reside in the Work.Houses data set. PROC FREQ reports nine invalid class levels. PROC REPORT treats only one class level as invalid. If the lowest internal value in a format group is a missing value, then the entire group is treated as missing. Because missing values are considered the lowest possible internal value for either numeric or character variables, missing values cause an entire format group to have the lowest internal value. Missing values rank first when ordering by internal values. The following code demonstrates this by adding the MISSING option to the previous example:

```
proc freq data=houses;
    title1 "PROC FREQ";
    title2 "With MISSING Specified";
    tables bedrooms / nocum nopercnt missing ;
run;

proc report data=houses missing;
    title1 "PROC REPORT";
    title2 "With MISSING Specified";
    column bedrooms n;
    define bedrooms / group width=8;
run;
```

Output 2.2 *PROC FREQ and PROC REPORT with MISSING Specified***PROC FREQ**
With MISSING Specified**The FREQ Procedure**

bedrooms	Frequency
OTHER	9
ONE	2
TWO	5

PROC REPORT
With MISSING Specified

bedrooms	n
ONE	2
OTHER	9
TWO	5

In the results for PROC FREQ, the group OTHER is ordered first in the sequence. In the results for PROC REPORT, OTHER is ordered second. By default, PROC FREQ orders by internal values and PROC REPORT orders by formatted values. Note that the frequency count is the same for both procedures. In PROC FREQ, OTHER corresponds to the missing frequency in the Work.Houses data set. For PROC REPORT, the frequency increases by one to account for the single missing value that was not included in Work.Houses.

Order Data Using Formats: BY Variables

If a format is applied to a variable in a BY statement, then the values of the BY variable are grouped unless the internal values are not sequenced in the format range:

```
proc sort data=sasuser.houses out=houses;
    by bedrooms;
run;

proc format;
    value numf 3='GROUP A' 1,2,4='GROUP B';
run;

proc report data=houses;
    by bedrooms;
```

```
format bedrooms numf.;
column price;
run;
```

Output 2.3 Ordering with a Formatted BY Variable



Order Data Using the ORDER= Option

The ORDER= option enables you to choose the ordering scheme that the procedure uses. The ORDER= option can have the values INTERNAL or UNFORMATTED, FORMATTED, DATA, and FREQ.

Note: The MEANS, REPORT, SUMMARY, and TABULATE procedures accept both the INTERNAL and UNFORMATTED as a value for the ORDER= option to order unformatted data.

You can use the option with these procedures:

Procedure	SAS Product	Default Value
CATMOD	SAS/STAT	INTERNAL
FREQ	Base SAS	INTERNAL

Procedure	SAS Product	Default Value
GLM	SAS/STAT	FORMATTED
LIFEREG	SAS/STAT	FORMATTED
LOGISTIC	SAS/STAT	FORMATTED
MEANS	Base SAS	UNFORMATTED
PROBIT	SAS/STAT	FORMATTED
REPORT	Base SAS	FORMATTED
SUMMARY	Base SAS	UNFORMATTED
TABULATE	Base SAS	UNFORMATTED

The following topics explain each value of ORDER=.

Order Data Using the ORDER= INTERNAL Option

When you specify ORDER=INTERNAL with no other ordering options specified, it causes class variable values to be listed in ascending order by their actual, unformatted values:

```
proc format;
    value numf 1='ONE' 2='TWO' 3='THREE' 4='FOUR';
run;
proc report data=sasuser.houses;
    column bedrooms;
    define bedrooms /group format=numf8. order=internal;
run;
```

Output 2.4 Output When ORDER=INTERNAL

The SAS System	
Number of bedrooms	
	ONE
	TWO
	THREE
	FOUR

The values for bedroom are ONE, TWO, THREE, and FOUR, which correspond to the numbers 1, 2, 3, 4. If you do not specify ORDER=INTERNAL, the values are listed in alphabetical order by the formatted value: FOUR, ONE, THREE, TWO.

Order Data Using the ORDER= FORMATTED Option

When you specify ORDER=FORMATTED with no other ordering options specified causes class variable values to be listed in alphabetical order by their formatted values:

```
proc format;
    value numf 1='ONE' 2='TWO' 3='THREE' 4='FOUR';
run;
proc freq data=sasuser.houses order=formatted;
    tables bedrooms / nopercnt;
    format bedrooms numf8. ;
run;
```

Output 2.5 Output When ORDER=FORMATTED

The SAS System		
The FREQ Procedure		
Number of bedrooms		
bedrooms	Frequency	Cumulative Frequency
FOUR	4	4
ONE	2	6
THREE	4	10
TWO	5	15

If ORDER=FORMATTED is specified and a class variable does not have a format associated with it, then the output is listed by its internal, or unformatted values.

In a special case, as the default behavior for PROC REPORT is ORDER=FORMATTED, this setting might cause problems because PROC REPORT uses BESTw. as the default format for numeric values. Here is an example:

```
proc report data=sasuser.houses;
    title1 "ORDER=FORMATTED";
    title2 "(default)";
    title4 "FORMAT=BEST9.";
    title5 "(default)";
    column baths;
    define baths / group;
run;
```

Here, BEST9. is the default format because a format is not specified. Since the default setting for PROC REPORT is ORDER=FORMATTED, the values are sorted using the formatted values. This can be complicated by character comparisons that sometimes give misleading results. The values that are being compared are "1", "2", "3", "1.5", and "2.5". The single-digit values have leading blanks. Since a blank character sorts before a number or a period (.), the values "1", "2", and "3" sort before the values 1.5 or 2.5.

Output 2.6 PROC REPORT Default Formatting When Values Have Leading Blanks

ORDER=FORMATTED (default)	
FORMAT=BEST9. (default)	
Number of bathrooms	
	1
	2
	3
	1.5
	2.5

This problem is corrected in the next output by using the 3.1 format, causing no leading blanks to appear in the comparison:

```
proc report data=sasuser.houses;
  title1 "ORDER=FORMATTED";
  title2 "(default)";
  title4 "FORMAT=3.1";
  title5 "(specified)";
  column baths;
  define baths / group format=3.1;
run;
```

Output 2.7 PROC REPORT Formatted When Values Have No Leading Blanks

ORDER=FORMATTED
(default)

FORMAT=3.1
(specified)

Number of bathrooms	
	1.0
	1.5
	2.0
	2.5
	3.0

The problem is also corrected in the following output because the internal values and not the formatted values are used to determine the order:

```
proc report data=sasuser.houses nowd;
  title1 "ORDER =INTERNAL";
  title2 "(specified)";
  title4 "FORMAT=BEST9.";
  title5 "(default)";
  column baths;
  define baths / group order=internal;
run;
```

Output 2.8 PROC FORMAT Output Based on Internal Values

ORDER =INTERNAL
(specified)

FORMAT=BEST9.
(default)

Number of bathrooms	
	1
	1.5
	2
	2.5
	3

Order Data Using the ORDER= DATA Option

ORDER=DATA specifies that the order is set according to how the data is initially read by the procedure. ORDER=DATA can be complicated depending on the use of BY statements or multiple classification variables. Here is a simple case:

```
proc tabulate data=sasuser.houses order=data format=3. noseps;
  class style;
  table style, n;
run;
```

Output 2.9 PROC TABULATE with ORDER=DATA

	N
Style of homes	
RANCH	4
SPLIT	3
CONDO	4
TWOSTORY	4

The output order for Style is neither ascending nor descending. RANCH appears first because it is the first value that is encountered in the input data set. SPLIT appears second because it is encountered second, and so on. If you use a BY statement, then the order is reset at the beginning of each new BY group, as if a new data set were being processed.

When you use multiple classification variables, the order is determined independently for each classification variable across the entire data set or BY group. The ordering scheme is developed first using the order of the highest order variable and then the next highest, and so on. Here is an example:

```
proc tabulate data=sasuser.houses order=data format=3. noseps;
  class style bedrooms;
  table style*bedrooms, n;
run;
```

Output 2.10 PROC TABULATE Using Two Classification Variables

		N
Style of homes	Number of bedrooms	
RANCH	2	1
	1	1
	3	2
SPLIT	1	1
	4	1
	3	1
CONDO	2	2
	4	1
	3	1
TWO STORY	2	2
	4	2

To compare, the order for Style is RANCH, SPLIT, CONDO, TWO STORY. The order for Bedrooms is 2, 1, 4, 3. This is not clearly apparent in the output because no value of STYLE has all four values of Bedrooms. The order of values is established independently for each variable and not according to the subgroups. You can verify this by adding the PRINTMISS option to the TABLE statement or by comparing the order of Style and Bedrooms in the Sasuser.Houses data set. See [“Order Data Using Multiple Classification Variables” on page 30](#).

Order Data Using the ORDER= FREQ Option

ORDER=FREQ specifies that classification variables are to be ordered by the frequencies of each of their values. With the exception of PROC FREQ, missing class levels are ordered by their frequency counts just as nonmissing class levels are. PROC FREQ always lists missing class levels first, regardless of their frequency counts. All Base SAS procedures, except for PROC REPORT, list the frequencies in descending order. PROC REPORT, by default, lists the frequencies in ascending order. If you use the PROC REPORT option DESCENDING in conjunction with ORDER=FREQ, then the class levels are ordered by their descending frequency counts.

If two class levels have the same frequency, a secondary ordering algorithm is used. All Base SAS procedures, except for PROC FREQ, use ORDER=DATA as a secondary ordering method.

If duplicate frequency counts occur with PROC FREQ and ORDER=FREQ has been specified, then PROC FREQ uses ORDER=FORMATTED as a secondary ordering method. If in PROC FREQ a format has not been applied, then the tie is broken

using the ORDER=INTERNAL method. If you use PROC REPORT with ORDER=FREQ and the DESCENDING option, and a tie occurs, then the ORDER=DATA method is used, but the levels are listed in reverse order of occurrence in the data.

The following table summarizes the behavior across the Base SAS procedures that use the ORDER= option:

Procedure	Order Direction	Format	Order Value In a Tie
FREQ	Descending	No	ORDER=INTERNAL
		Yes	ORDER=FORMATTED
MEANS	Descending	Yes or No	ORDER=DATA
REPORT	Ascending	Yes or No	ORDER=DATA
	Descending	Yes or No	ORDER=DATA ¹
SUMMARY	Descending	Yes or No	ORDER=DATA
TABULATE	Descending	Yes or No	ORDER=DATA

¹ If you specify the DESCENDING option, then both the primary method, ORDER=FREQ, and the secondary method, ORDER=DATA, list levels in descending order.

Here are some examples when ORDER=FREQ is specified with the TABULATE, FREQ, and REPORT procedures. First we have PROC TABULATE and PROC FREQ output without a format specified:

```
proc format;
  value bedfmt 1='ONE' 2='TWO' 3='THREE' 4='FOUR';

proc tabulate data=sasuser.houses order=freq noseps format=3.;
  title1 "PROC TABULATE";
  title2 "Without Format";
  class bedrooms;
  table bedrooms, n;
run;

proc freq data=sasuser.houses order=freq;
  title1 "PROC FREQ";
  title2 "Without Format";
  tables bedrooms / nocum nopercnt;
run;
```

Both outputs list data values in descending order of the frequency counts.

Output 2.11 ORDER=FREQ Examples

PROC TABULATE
Without Format

	N
Number of bedrooms	
2	5
4	4
3	4
1	2

PROC FREQ
Without Format

The FREQ Procedure

Number of bedrooms	
bedrooms	Frequency
2	5
3	4
4	4
1	2

PROC REPORT, when no format is specified, lists the output in ascending order:

```
proc report data=sasuser.houses;
  title1 "PROC REPORT";
  title2 "Without Format";
  title3 "Without DESCENDING";
  column bedrooms n;
  define bedrooms / group order=freq;
run;
```


Output 2.12 PROC REPORT with No Format Specified

PROC REPORT
Without Format
Without DESCENDING

Number of bedrooms	n
1	2
4	4
3	4
2	5

For PROC REPORT to list output in descending order, you must specify DESCENDING in the DEFINE statement:

```
proc report data=sasuser.houses;
  title1 "PROC REPORT";
  title2 "Without Format";
  title3 "With DESCENDING";
  column bedrooms n;
  define bedrooms / group order=freq descending;
run;
```

PROC REPORT
Without Format
With DESCENDING

Number of bedrooms	n
2	5
4	4
3	4
1	2

PROC TABULATE and PROC REPORT use the ORDER=DATA method to handle values that are the same.

```
proc tabulate data=sasuser.houses order=freq noseps format=3.;
  title1 "PROC TABULATE";
  title2 "With Format";
  class bedrooms;
  table bedrooms, n;
  format bedrooms bedfmt.;
run;

proc report data=sasuser.houses;
  title1 "PROC REPORT";
  title2 "With Format";
  title3 "Without DESCENDING";
```

```

column bedrooms n;
define bedrooms / group order=freq format=bedfmt8.;
run;

```

Output 2.13 *Descending Output Examples*

**PROC TABULATE
With Format**

	N
Number of bedrooms	
TWO	5
FOUR	4
THREE	4
ONE	2

**PROC REPORT
With Format
Without DESCENDING**

Number of bedrooms	n
ONE	2
FOUR	4
THREE	4
TWO	5

PROC FREQ uses ORDER=FORMATTED when a format is specified and values are the same.

```

proc freq data=sasuser.houses order=freq;
  title1 "PROC FREQ";
  title2 "With Format";
  tables bedrooms / nocum nopercnt;
  format bedrooms bedfmt.;
run;

```

Output 2.14 Descending Output Using PROC FREQ with a Format

PROC FREQ
With Format

The FREQ Procedure

Number of bedrooms	
bedrooms	Frequency
TWO	5
FOUR	4
THREE	4
ONE	2

If you do not specify a format, PROC FREQ uses ORDER=INTERNAL.

In PROC REPORT when DESCENDING is applied, data values are listed in descending order of frequency counts and the data values that had the same frequency are listed in reverse order of occurrence. In other words, ORDER=DATA is used to handle values that are the same, and the data values are listed in reverse order.

Output 2.15 Output Using PROC REPORT with a Format and the DESCENDING Option

PROC REPORT
With Format
With DESCENDING

Number of bedrooms	n
TWO	5
FOUR	4
THREE	4
ONE	2

As with ORDER=DATA, when you add a BY statement, the order for classification variables is established as if each BY group were a separate data set. When you use multiple classification variables with ORDER=FREQ, the order is determined independently for each classification variable. The overall ordering scheme is then applied first by using the order of the highest order variable and then the next-highest order variable, and so on.

In this example, consider the frequencies for Baths:

```
proc tabulate data=sasuser.houses order=freq format=3. noseps;
```

```

class baths;
table baths, n;
run;

```

Output 2.16 The Frequency of Baths Using PROC TABULATE

	N
Number of bathrooms	
1	5
3	4
2.5	3
1.5	2
2	1

If you make Bedrooms the highest order variable and Baths the next-highest order variable, you expect the rows to be ordered by the descending frequency of values first in Bedrooms, then in Baths.

```

proc tabulate data=sasuser.houses order=freq format=3. noseps;
  class baths bedrooms;
  table bedrooms*baths, n;
run;

```

The N statistic does not list the frequencies of the combinations in descending order. Instead, the order is set first by the descending frequencies of BEDROOMS and then by BATHS. You can verify this by comparing the order of each variable against the orders used in [Output 2.11 on page 50](#). The ordering sequence for BEDROOMS is 2, 4, 3, 1. The ordering sequence for BATHS is 1, 3, 2.5, 1.5, 2.

Output 2.17 PROC TABULATE with Bedrooms as the High Order Variable

		N
Number of bedrooms	Number of bathrooms	
2	1	3
	1.5	1
	2	1
4	3	2
	2.5	2
3	3	2
	2.5	1
	1.5	1
1	1	2

RUN-Group Processing

RUN-group processing enables you to submit a PROC step with a RUN statement without ending the procedure. You can continue to use the procedure without issuing another PROC statement. To end the procedure, use a RUN CANCEL or a QUIT statement. Several Base SAS procedures support RUN-group processing:

CATALOG DATASETS DS2 FEDSQL TRANTAB

See the section on the individual procedure for more information.

Note: PROC SQL executes each query automatically. Neither the RUN nor RUN CANCEL statement has any effect.

Creating Titles That Contain BY-Group Information

BY-Group Processing

BY-group processing uses a BY statement to process observations that are ordered, grouped, or indexed according to the values of one or more variables. By default, when you use BY-group processing in a procedure step, a BY line identifies each group. This section explains how to create titles that serve as customized BY lines.

Suppressing the Default BY Line

When you insert BY-group processing information into a title, you usually want to suppress the default BY line. To suppress it, use the SAS system option NOBYLINE.

Note: You must use the NOBYLINE option if you insert BY-group information into titles for the following procedures:

- MEANS
- PRINT
- STANDARD
- SUMMARY

If you use the BY statement with the NOBYLINE option, then these procedures always start a new page for each BY group. This behavior prevents multiple BY

groups from appearing on a single page and ensures that the information in the titles matches the report on the pages.

Inserting BY-Group Information into a Title

The general form for inserting BY-group information into a title is as follows:

#BY-specification<.suffix>

BY-specification is one of the following specifications:

BYVAL*n* | BYVAL(*BY-variable*)

places the value of the specified BY variable in the title. You specify the BY variable with one of the following options:

n

is the *n*th BY variable in the BY statement.

BY-variable

is the name of the BY variable whose value you want to insert in the title.

BYVAR*n* | BYVAR(*BY-variable*)

places the label or the name (if no label exists) of the specified BY variable in the title. You designate the BY variable with one of the following options:

n

is the *n*th BY variable in the BY statement.

BY-variable

is the name of the BY variable whose value you want to insert in the title.

BYLINE

inserts the complete default BY line into the title.

suffix supplies text to place immediately after the BY-group information that you insert in the title. No space appears between the BY-group information and the suffix.

Example: Inserting a Value from Each BY Variable into the Title

This example demonstrates these actions:

- 1 creates a data set, GROC, that contains data for stores from four regions. Each store has four departments. See [“GROC” on page 2516](#) for the DATA step that creates the data set.
- 2 sorts the data by Region and Department.
- 3 uses the SAS system option NOBYLINE to suppress the BY line that normally appears in output that is produced with BY-group processing.

- 4 uses PROC REPORT to chart sales by Region and Department. In the first TITLE statement, #BYVAL2 inserts the value of the second BY variable, Department, into the title. In the second TITLE statement, #BYVAL(Region) inserts the value of Region into the title. The first period after Region indicates that a suffix follows. The second period is the suffix.
- 5 uses the SAS system option BYLINE to return to the creation of the default BY line with BY-group processing.

```

data groc;
    input Region $9. Manager $ Department $ Sales;
    datalines;
Southeast    Hayes    Paper    250
Southeast    Hayes    Produce  100
Southeast    Hayes    Canned  120
Southeast    Hayes    Meat    80
...more lines of data...
Northeast    Fuller    Paper    200
Northeast    Fuller    Produce  300
Northeast    Fuller    Canned  420
Northeast    Fuller    Meat    125
;

proc sort data=groc;
    by region department;
run;
options nobyline nodate pageno=1
        linesize=64 pagesize=20;
proc report data=groc;
    by region department;
    title1 'This chart shows #byval2 sales';
    title2 'in the #byval(region)..';
run;
options byline;

```

This partial output shows two BY groups with customized BY lines:

**This report shows Canned sales
in the Northeast.**

Region	Manager	Department	Sales
Northeast	Fuller	Canned	420

**This report shows Meat sales
in the Northeast.**

Region	Manager	Department	Sales
Northeast	Fuller	Meat	125

**This report shows Paper sales
in the Northeast.**

Region	Manager	Department	Sales
Northeast	Fuller	Paper	200

...more...

Example: Inserting the Name of a BY Variable into a Title

This example inserts the name of a BY variable and the value of a BY variable into the title. The program does these actions.

- 1 uses the SAS system option NOBYLINE to suppress the BY line that normally appears in output that is produced with BY-group processing.
- 2 uses PROC REPORT to chart sales by Region. In the first TITLE statement, #BYVAR(Region) inserts the name of the variable Region into the title. (If Region had a label, #BYVAR would use the label instead of the name.) The suffix al is appended to the label. In the second TITLE statement, #BYVAL1 inserts the value of the first BY variable, Region, into the title.

- 3 uses the SAS system option BYLINE to return to the creation of the default BY line with BY-group processing.

```

options nobyline nodate pageno=1
      linesize=64 pagesize=20;      1
proc report data=groc;              2
  by region;
  title1 '#byvar(region).al Analysis';
  title2 'for the #byval1';
run;
options byline;                      3

```

This partial output shows one BY group with a customized BY line:

Regional Analysis for the Northeast

Region	Manager	Department	Sales
Northeast	Fuller	Canned	420
Northeast	Fuller	Meat	125
Northeast	Fuller	Paper	200
Northeast	Fuller	Produce	300

Regional Analysis for the Southeast

Region	Manager	Department	Sales
Southeast	Hayes	Canned	120
Southeast	Hayes	Meat	80
Southeast	Hayes	Paper	250
Southeast	Hayes	Produce	100

Example: Inserting the Complete BY Line into a Title

This example inserts the complete BY line into the title. The program does these actions:

- 1 uses the SAS system option NOBYLINE to suppress the BY line that normally appears in output that is produced with BY-group processing.
- 2 uses PROC REPORT to chart sales by Region and Department. In the TITLE statement, #BYLINE inserts the complete BY line into the title.
- 3 uses the SAS system option BYLINE to return to the creation of the default BY line with BY-group processing.

```
options nobyline nodate pageno=1  
      linesize=64 pagesize=20; 1  
proc report data=groc;          2  
  by region department;  
  vbar manager / type=sum sumvar=sales;  
  title 'Information for #byline';  
run;  
options byline;                 3
```

This partial output shows two BY groups with customized BY lines:

Information for Region=Northeast Department=Canned

Region	Manager	Department	Sales
Northeast	Fuller	Canned	420

Information for Region=Northeast Department=Meat

Region	Manager	Department	Sales
Northeast	Fuller	Meat	125

Information for Region=Northeast Department=Paper

Region	Manager	Department	Sales
Northeast	Fuller	Paper	200

Information for Region=Northeast Department=Produce

Region	Manager	Department	Sales
Northeast	Fuller	Produce	300

...more observations...

Error Processing of BY-Group Specifications

SAS does not issue error or warning messages for incorrect #BYVAL, #BYVAR, or #BYLINE specifications. Instead, the text of the item becomes part of the title.

Shortcuts for Specifying Lists of Variable Names

Several statements in procedures allow multiple variable names. You can use these shortcut notations instead of specifying each variable name:

Notation	Meaning
x1-xn	Specifies variables X1 through Xn. The numbers must be consecutive.
x:	Specifies all variables that begin with the letter X.

Notation	Meaning
x--a	Specifies all variables between X and A, inclusive. This notation uses the position of the variables in the data set.
x-numeric-a	Specifies all numeric variables between X and A, inclusive. This notation uses the position of the variables in the data set.
x-character-a	Specifies all character variables between X and A, inclusive. This notation uses the position of the variables in the data set.
numeric	Specifies all numeric variables.
character	Specifies all character variables.
all	Specifies all variables.

Note: You cannot use shortcuts to list variable names in the INDEX CREATE statement in PROC DATASETS.

For more information, see the *SAS Programmer's Guide: Essentials*.

Formatted Values

Using Formatted Values

Typically, when you print or group variable values, Base SAS procedures use the formatted values. This section contains examples of how Base SAS procedures use formatted values.

Example: Printing the Formatted Values for a Data Set

The following example prints the formatted values of the data set PROCLIB.PAYROLL. (For details about the DATA step that creates this data set, see [“PROCLIB.PAYROLL” on page 2525](#).) In PROCLIB.PAYROLL, the variable Jobcode indicates the job and level of the employee. For example, **TA1** indicates that the employee is at the beginning level for a ticket agent.

```

options nodate pageno=1
      linesize=64 pagesize=40;
proc print data=proclib.payroll(obs=10)
      noobs;
      title 'PROCLIB.PAYROLL';
      title2 'First 10 Observations Only';
run;

```

The following example is a partial printing of PROCLIB.PAYROLL:

PROCLIB.PAYROLL First 10 Observations Only					
IdNumber	Gender	Jobcode	Salary	Birth	Hired
1919	M	TA2	34376	12SEP60	04JUN87
1653	F	ME2	35108	15OCT64	09AUG90
1400	M	ME1	29769	05NOV67	16OCT90
1350	F	FA3	32886	31AUG65	29JUL90
1401	M	TA3	38822	13DEC50	17NOV85
1499	M	ME3	43025	26APR54	07JUN80
1101	M	SCP	18723	06JUN62	01OCT90
1333	M	PT2	88606	30MAR61	10FEB81
1402	M	TA2	32615	17JAN63	02DEC90
1479	F	TA3	38785	22DEC68	05OCT89

The following PROC FORMAT step creates the format \$JOBfmt., which assigns descriptive names for each job:

```

proc format;
  value $jobfmt
    'FA1'='Flight Attendant Trainee'
    'FA2'='Junior Flight Attendant'
    'FA3'='Senior Flight Attendant'
    'ME1'='Mechanic Trainee'
    'ME2'='Junior Mechanic'
    'ME3'='Senior Mechanic'
    'PT1'='Pilot Trainee'
    'PT2'='Junior Pilot'
    'PT3'='Senior Pilot'
    'TA1'='Ticket Agent Trainee'
    'TA2'='Junior Ticket Agent'
    'TA3'='Senior Ticket Agent'
    'NA1'='Junior Navigator'
    'NA2'='Senior Navigator'
    'BCK'='Baggage Checker'
    'SCP'='Skycap';
run;

```

The FORMAT statement in this PROC MEANS step temporarily associates the \$JOBfmt. format with the variable Jobcode:

```

options nodate pageno=1

```

```

linesize=64 pagesize=60;
proc means data=proclib.payroll mean max;
  class jobcode;
  var salary;
  format jobcode $jobfmt.;
  title 'Summary Statistics for';
  title2 'Each Job Code';
run;

```

PROC MEANS produces this output, which uses the \$JOBfmt. format:

Summary Statistics for Each Job Code			
The MEANS Procedure			
Analysis Variable : Salary			
Jobcode	N Obs	Mean	Maximum
Baggage Checker	9	25794.22	26896.00
Flight Attendant Trainee	11	23039.36	23979.00
Junior Flight Attendant	16	27986.88	28978.00
Senior Flight Attendant	7	32933.86	33419.00
Mechanic Trainee	8	28500.25	29769.00
Junior Mechanic	14	35576.86	36925.00
Senior Mechanic	7	42410.71	43900.00
Junior Navigator	5	42032.20	43433.00
Senior Navigator	3	52383.00	53798.00
Pilot Trainee	8	67908.00	71349.00
Junior Pilot	10	87925.20	91908.00
Senior Pilot	2	10504.50	11379.00
Skycap	7	18308.86	18833.00
Ticket Agent Trainee	9	27721.33	28880.00
Junior Ticket Agent	20	33574.95	34803.00
Senior Ticket Agent	12	39679.58	40899.00

Note: Because formats are character strings, formats for numeric variables are ignored when the values of the numeric variables are needed for mathematical calculations.

Example: Grouping or Classifying Formatted Data

If you use a formatted variable to group or classify data, then the procedure uses the formatted values. The following example creates and assigns a format, \$CODEFMT., that groups the levels of each job code into one category. PROC MEANS calculates statistics based on the groupings of the \$CODEFMT. format.

```
proc format;
  value $codefmt
    'FA1','FA2','FA3'='Flight Attendant'
    'ME1','ME2','ME3'='Mechanic'
    'PT1','PT2','PT3'='Pilot'
    'TA1','TA2','TA3'='Ticket Agent'
    'NA1','NA2'='Navigator'
    'BCK'='Baggage Checker'
    'SCP'='Skycap';

run;

options nodate pageno=1
       linesize=64 pagesize=40;
proc means data=proclib.payroll mean max;
  class jobcode;
  var salary;
  format jobcode $codefmt.;
  title 'Summary Statistics for Job Codes';
  title2 '(Using a Format that Groups the Job Codes)';
run;
```

PROC MEANS produces this output:

Summary Statistics for Job Codes (Using a Format that Groups the Job Codes)

The MEANS Procedure

Analysis Variable : Salary			
Jobcode	N Obs	Mean	Maximum
Baggage Checker	9	25794.22	26896.00
Flight Attendant	34	27404.71	33419.00
Mechanic	29	35274.24	43900.00
Navigator	8	45913.75	53798.00
Pilot	20	72176.25	91908.00
Skycap	7	18308.86	18833.00
Ticket Agent	41	34076.73	40899.00

Example: Temporarily Associating a Format with a Variable

If you want to associate a format with a variable temporarily, then you can use the **FORMAT** statement. For example, the following **PROC PRINT** step associates the **DOLLAR8.** format with the variable **Salary** for the duration of this **PROC PRINT** step only:

```
options nodate pageno=1
      linesize=64 pagesize=40;
proc print data=proclib.payroll(obs=10)
      noobs;
      format salary dollar8.;
      title 'Temporarily Associating a Format';
      title2 'with the Variable Salary';
run;
```

PROC PRINT produces this output:

Temporarily Associating a Format with the Variable Salary					
IdNumber	Gender	Jobcode	Salary	Birth	Hired
1919	M	TA2	\$34,376	12SEP60	04JUN87
1653	F	ME2	\$35,108	15OCT64	09AUG90
1400	M	ME1	\$29,769	05NOV67	16OCT90
1350	F	FA3	\$32,886	31AUG65	29JUL90
1401	M	TA3	\$38,822	13DEC50	17NOV85
1499	M	ME3	\$43,025	26APR54	07JUN80
1101	M	SCP	\$18,723	06JUN62	01OCT90
1333	M	PT2	\$88,606	30MAR61	10FEB81
1402	M	TA2	\$32,615	17JAN63	02DEC90
1479	F	TA3	\$38,785	22DEC68	05OCT89

Example: Temporarily Dissociating a Format from a Variable

If a variable has a permanent format that you do not want a procedure to use, then temporarily dissociate the format from the variable by using a **FORMAT** statement.

In this example, the **FORMAT** statement in the **DATA** step permanently associates the **\$YRFMT.** variable with the variable **Year**. Thus, when you use the variable in a **PROC** step, the procedure uses the formatted values. The **PROC MEANS** step,

however, contains a FORMAT statement that dissociates the \$YRFMT. format from Year for this PROC MEANS step only. PROC MEANS uses the stored value for Year in the output.

```
proc format;
  value $yrfmt  '1'='Freshman'
                '2'='Sophomore'
                '3'='Junior'
                '4'='Senior';

run;
data debate;
  input Name $ Gender $ Year $ GPA @@;
  format year $yrfmt.;
  datalines;
Capiccio m 1 3.598 Tucker    m 1 3.901
Bagwell  f 2 3.722 Berry     m 2 3.198
Metcalf  m 2 3.342 Gold      f 3 3.609
Gray     f 3 3.177 Syme      f 3 3.883
Baglione f 4 4.000 Carr      m 4 3.750
Hall     m 4 3.574 Lewis     m 4 3.421
;

options nodate pageno=1
        linesize=64 pagesize=40;
proc means data=debate mean maxdec=2;
  class year;
  format year;
  title 'Average GPA';
run;
```

PROC MEANS produces this output, which does not use the YRFMT. format:

Average GPA		
The MEANS Procedure		
Analysis Variable : GPA		
Year	N Obs	Mean
1	2	3.75
2	3	3.42
3	3	3.56
4	4	3.69

Formats and BY-Group Processing

When a procedure processes a data set, it checks to determine whether a format is assigned to the BY variable. If it is, then the procedure adds observations to the current BY groups until the formatted value changes. If *nonconsecutive* internal values of the BY variables have the same formatted value, then the values are grouped into different BY groups. Therefore, two BY groups are created with the same formatted value. Also, if different and *consecutive* internal values of the BY variables have the same formatted value, then they are included in the same BY group.

Formats and Error Checking

If SAS cannot find a format, then it stops processing and prints an error message in the SAS log. You can suppress this behavior with the SAS system option NOFMterr. If you use NOFMterr, and SAS cannot find the format, then SAS uses a default format and continues processing. Typically, for the default, SAS uses the BEST w. format for numeric variables and the \$w. format for character variables.

Note: To ensure that SAS can find user-written formats, use the SAS system option FMTSEARCH=. How to store formats is described in [“Storing Informats and Formats” on page 868](#).

Processing All the Data Sets in a Library

You can use the SAS Macro Facility to run the same procedure on every data set in a library. The macro facility is part of the Base SAS software.

“[Example 10: Printing All the Data Sets in a SAS Library](#)” on page 1490 shows how to print all the data sets in a library. You can use the same macro definition to perform any procedure on all the data sets in a library. Simply replace the PROC PRINT piece of the program with the appropriate procedure code.

Statistic Descriptions

The following table identifies common descriptive statistics that are available in several Base SAS procedures. For more detailed information about available statistics and theoretical information, see [“Keywords and Formulas” on page 2426](#).

Table 2.1 Common Descriptive Statistics That Base SAS Procedures Calculate

Statistic	Description	Procedures
Confidence intervals		FREQ, MEANS or SUMMARY, TABULATE, UNIVARIATE
CSS	Corrected sum of squares	CORR, MEANS or SUMMARY, REPORT, SQL, TABULATE, UNIVARIATE
CV	Coefficient of variation	MEANS or SUMMARY, REPORT, SQL, TABULATE, UNIVARIATE
Goodness-of-fit tests		FREQ, UNIVARIATE
KURTOSIS	Kurtosis	MEANS or SUMMARY, TABULATE, UNIVARIATE
MAX	Largest (maximum) value	CORR, MEANS or SUMMARY, REPORT, SQL, TABULATE, UNIVARIATE
MEAN	Mean	CORR, MEANS or SUMMARY, REPORT, SQL, TABULATE, UNIVARIATE
MEDIAN	Median (50 th percentile)	CORR (for nonparametric correlation measures), MEANS or SUMMARY, TABULATE, UNIVARIATE
MIN	Smallest (minimum) value	CORR, MEANS or SUMMARY, REPORT, SQL, TABULATE, UNIVARIATE
MODE	Most frequent value (if not unique, the smallest mode is used)	UNIVARIATE
N	Number of observations on which calculations are based	CORR, FREQ, MEANS or SUMMARY, REPORT, SQL, TABULATE, UNIVARIATE
NMISS	Number of missing values	FREQ, MEANS or SUMMARY, REPORT, SQL, TABULATE, UNIVARIATE
NOBS	Number of observations	MEANS or SUMMARY, UNIVARIATE
PCTN	Percentage of a cell or row frequency to a total frequency	REPORT, TABULATE

Statistic	Description	Procedures
PCTSUM	Percentage of a cell or row sum to a total sum	REPORT, TABULATE
Pearson correlation		CORR
Percentiles		FREQ, MEANS or SUMMARY, REPORT, TABULATE, UNIVARIATE
RANGE	Range	CORR, MEANS or SUMMARY, REPORT, SQL, TABULATE, UNIVARIATE
Robust statistics	Trimmed means, Winsorized means	UNIVARIATE
SKEWNESS	Skewness	MEANS or SUMMARY, TABULATE, UNIVARIATE
Spearman correlation		CORR
STD	Standard deviation	CORR, MEANS or SUMMARY, REPORT, SQL, TABULATE, UNIVARIATE
STDERR	Standard error of the mean	MEANS or SUMMARY, REPORT, SQL, TABULATE, UNIVARIATE
SUM	sum	CORR, MEANS or SUMMARY, REPORT, SQL, TABULATE, UNIVARIATE
SUMWGT	Sum of weights	CORR, MEANS or SUMMARY, REPORT, SQL, TABULATE, UNIVARIATE
Tests of location		UNIVARIATE
USS	Uncorrected sum of squares	CORR, MEANS or SUMMARY, REPORT, SQL, TABULATE, UNIVARIATE
VAR	Variance	CORR, MEANS or SUMMARY, REPORT, SQL, TABULATE, UNIVARIATE

Computational Requirements for Statistics

The following computational requirements are for the statistics that are listed in [Table 2.1 on page 69](#). They do not describe recommended sample sizes.

- N and NMISS do not require any nonmissing observations.
- SUM, MEAN, MAX, MIN, RANGE, USS, and CSS require at least one nonmissing observation.
- VAR, STD, STDERR, and CV require at least two observations.
- CV requires that MEAN is not equal to zero.

Statistics are reported as missing if they cannot be computed.

Output Delivery System

The Output Delivery System (ODS) gives you greater flexibility in generating, storing, and reproducing SAS procedure and DATA step output, with a wide range of formatting options. ODS provides formatting functionality that is not available from individual procedures or from the DATA step alone. ODS overcomes these limitations and enables you to format your output more easily.

- Some commonly used procedures do not produce output data sets.

For more information about the Output Delivery System, see the *SAS Output Delivery System: User's Guide*.

In-Database Processing of Base Procedures

Base Procedures That Are Enhanced for In-Database Processing 73

Base Procedures That Are Enhanced for In-Database Processing

In-database processing has several advantages over processing within SAS. These advantages include the potentials for increased security, reduced network traffic, and faster processing. Increased security might be possible if sensitive data does not have to be extracted from the data source. Faster processing is possible for the following reasons:

- Data is manipulated locally, on the data source, using high-speed secondary storage devices instead of being transported across a relatively slow network connection.
- The data source might have more processing resources at its disposal.
- The data source might be capable of optimizing a query for execution in a highly parallel and scalable fashion.

Base SAS procedures can process data inside several data sources. For more information, see [“In-Database Procedures” in SAS In-Database Products: User’s Guide](#).

The following Base SAS procedures support in-database processing.

Table 3.1 In-Database Base Procedures

Procedure	Description
PROC COPY on page 309	Copies one or more tables from one SAS library to another.
PROC FREQ in <i>Base SAS Procedures Guide: Statistical Procedures</i>	Produces one-way to n -way tables; reports frequency counts; computes test and measures of association and agreement for two-way to n -way crosstabulation tables; can compute exact tests and asymptotic tests; can create output data sets.
PROC MEANS on page 1235	Computes descriptive statistics; can produce printed output and output data sets. By default, PROC MEANS produces printed output.
PROC RANK on page 1706	Computes ranks for one or more numeric variables across the observations of a SAS data set; can produce some rank scores.
PROC REPORT on page 1846	Combines features of the PRINT, MEANS, and TABULATE procedures with features of the DATA step in a single report-writing tool that can produce a variety of reports.
PROC SORT on page 2076	Orders SAS data set observations by the values of one or more character or numeric variables.
PROC SUMMARY on page 2153	Computes descriptive statistics; can produce a printed report and create an output data set. By default, PROC SUMMARY creates an output data set.
PROC TABULATE on page 2293	Displays descriptive statistics in tabular format, using some or all of the variables in a data set.

CAS Processing of Base Procedures

<i>Procedures That Use CAS Actions</i>	75
<i>When CAS Processing Cannot Be Used</i>	77
<i>BY-Group Processing</i>	77
<i>Filtering Observations</i>	78
<i>Related Documents</i>	78

Procedures That Use CAS Actions

The following Base SAS procedures can run CAS actions:

Procedure	Description
PROC APPEND on page 125	Adds rows from a CAS table to the end of a SAS data set, and adds rows from a SAS data set to the end of a CAS table.
PROC CONTENTS on page 289	Shows the contents of a CAS table and prints the directory of the caslib.
PROC COPY on page 309	Copies entire SAS libraries or specific members of the library.
PROC DATASETS on page 365	Manages CAS tables.
PROC DELETE on page 585	Deletes SAS data sets and CAS tables.

Procedure	Description
PROC DS2 ¹	Manipulates data with DS2 language statements.
PROC FCMP on page 669	Enables you to create, test, and store SAS functions, CALL routines, and subroutines before you use them in other SAS procedures or in DATA steps.
PROC FEDSQL ²	Manipulates data and performs reporting with FedSQL language statements.
PROC FORMAT on page 863	Creates user-defined informats to read data and user-defined formats to display data.
PROC LUA on page 1163	Enables you to run statements from the Lua programming language within SAS code.
PROC MEANS on page 1237	Computes descriptive statistics; can produce printed output and output data sets. By default, PROC MEANS produces printed output.
PROC REPORT on page 1847	Combines features of the PRINT, MEANS, and TABULATE procedures with features of the DATA step in a single report-writing tool that can produce a variety of reports.
PROC SORT on page 2039	Provides the ability to manage duplicate observations in data sets. The SORT procedure uses the CAS deduplicate action to perform the equivalent functionality of the NODUPKEY or NOUNIKEY options of PROC SORT.
PROC SCOREACCEL on page 1971	Provides an interface to the CAS server for DATA step and DS2 model publishing and scoring.
PROC SUMMARY on page 2153	Computes descriptive statistics; can produce a printed report and create an output data set. By default, PROC SUMMARY creates an output data set.

Procedure	Description
PROC TABULATE on page 2292	Displays descriptive statistics in tabular format, using some or all of the variables in a data set.
PROC TRANSPOSE on page 2375	Transforms SAS data sets so that observations become variables and variables become observations.
1 The DS2 procedure does not use the CAS LIBNAME engine to access in-memory tables. Instead, the procedure accesses tables by caslib and name. For information and limitations, see “DS2 in CAS: Concepts” in SAS DS2 Programmer’s Guide. 2 The FEDSQL procedure does not use the CAS LIBNAME engine to access in-memory tables. Instead, the procedure accesses tables by caslib and name. For information and limitations, see SAS Viya Platform: FedSQL Programming for SAS Cloud Analytic Services	

When CAS Processing Cannot Be Used

The documentation for each procedure identifies the supported aggregations and supported statements and options. When your program includes a procedure statement or option that cannot be processed with a CAS action, the observation-level data for the in-memory table is transferred to the SAS client. The procedure then runs on the transferred data in SAS on the SAS Compute Server, rather than in CAS.

By default, the CAS LIBNAME engine limits data transfer to 100 MB. If you reach the limit:

- 1 You might be able to achieve the result that you want with a SAS Visual Statistics procedure or SAS Visual Data Mining and Machine Learning procedure.
- 2 You might be able to program with CAS actions so that the summarization is performed by the server.
- 3 You can increase the limit with the CASDATA LIMIT= system option or the DATA LIMIT= LIBNAME option or data set option.

BY-Group Processing

For procedures that support a BY statement, the information is passed to the CAS server. This enables two optimizations:

- 1 The data does not need to be pre-sorted by the specified variables.

- 2 When the results are transferred by the server to the SAS client, the groups are already formed. These results can be summarized results as with PROC MEANS or they can be observation-level as is the case with PROC PRINT.

If you know in advance that you will perform BY-group processing, especially if you have large in-memory tables, you can partition the in-memory table as a further efficiency. When you partition an in-memory table with the same variables that you use for BY-group processing, you avoid the performance penalty for forming the groups each time you access the table.

Filtering Observations

Procedures that support a WHERE statement, the expression is sent to the server. The server resolves the expression and subsets the data for the analysis. This can greatly reduce processing time and data transfer from the server to the SAS client.

The same is true of the WHERE= data set option when it is used with the CAS LIBNAME engine—the expression is sent to the server to subset the data.

```
cas casauto host="cloud.example.com" port=5570;
libname mycas cas sessref=casauto;

proc casutil;
  load data=sashelp.prdsale;
quit;

/* View informational messages in the SAS log.      */
options msglevel=i;

/* The WHERE statement is processed by the server.  */
proc means data=mycas.prdsale;
  where region eq 'EAST';
run;

/* These results are identical to the preceding use */
/* of PROC MEANS, but uses a WHERE= data set option. */
proc means data=mycas.prdsale(where=(region eq 'EAST'));
run;
```

Related Documents

- [SAS Cloud Analytic Services: Fundamentals](#)
- [SAS Cloud Analytic Services: User's Guide](#)
- [SAS Viya Platform: FedSQL Programming for SAS Cloud Analytic Services](#)

- [SAS DS2 Programmer's Guide](#)

Base SAS Procedures Documented in Other Publications

Base SAS Procedures Documented in Other Publications 81

Base SAS Procedures Documented in Other Publications

Some Base SAS procedures are documented in other SAS publications. The following table lists these procedures and the publications that contain them.

Procedure	Description	Publication
CALLRFC	Executes Remote Function Calls (RFC) or RFC-compatible functions on an SAP System.	SAS/ACCESS Interface to R/3: User's Guide
CORR	Computes Pearson correlation coefficients, three nonparametric measures of association, and the probabilities associated with these statistics.	Base SAS Procedures Guide: Statistical Procedures
DOCUMENT	Enables you to rearrange, duplicate, or remove output from the results of a procedure or a database query that are in ODS documents.	SAS Output Delivery System: User's Guide

Procedure	Description	Publication
FedSQL	Specifies that the subsequent input is FedSQL statements.	SAS FedSQL Language Reference
FREQ	Produces one-way to n-way frequency and contingency (crosstabulation) tables.	Base SAS Procedures Guide: Statistical Procedures
GCHART	Produces six types of charts: block charts, horizontal and vertical bar charts, pie and donut charts, and star charts.	SAS/GRAPH: Reference
GEOCODE	Adds geographic coordinates (latitude and longitude values) and attribute values such as census blocks to an address.	SAS/GRAPH and Base SAS: Mapping Reference
GINSIDE	Compares a data set of X and Y coordinates to a map data set containing map polygons. It determines whether the X and Y coordinates for each point fall inside or outside of the map polygons. The resulting output map data set contains the points inside the map polygons. It can be used as input by the GMAP procedure in SAS/GRAPH, or the ODS Graphics SGMAP procedure in Base SAS.	SAS/GRAPH and Base SAS: Mapping Reference
GPLOT	Plots the values of two or more variables on a set of coordinate axes (X and Y).	SAS/GRAPH and Base SAS: Mapping Reference
GPROJECT	Processes map data sets by converting spherical coordinates (longitude and latitude) into Cartesian coordinates for use by the GMAP and SGMAP procedures.	SAS/GRAPH and Base SAS: Mapping Reference
GREDUCE	Processes map data sets so that they can draw simpler maps with fewer boundary points. The resulting output map data set with an added DENSITY variable can be used as an input map data set by the GMAP procedure in SAS/GRAPH, or the ODS Graphics SGMAP procedure in Base SAS.	SAS/GRAPH and Base SAS: Mapping Reference
GREMOVE	Combines unit areas defined in a map data set into larger unit areas by removing internal borders between the original unit areas. The resulting output map data set can be used as input by the GMAP procedure in SAS/GRAPH,	SAS/GRAPH and Base SAS: Mapping Reference

Procedure	Description	Publication
	or the ODS Graphics SGMAP procedure in Base SAS.	
MAPIIMPORT	Enables you to import Esri shapefiles (spatial data formats) and process the SHP files into map data sets that are made available with SAS/GRAPH or through third-party sources.	SAS/GRAPH and Base SAS: Mapping Reference
ODSLIST	Stores a report's individual components and then enables you to modify and replay the report.	SAS Output Delivery System: User's Guide
ODSTABLE	Creates your tabular output templates.	SAS Output Delivery System: User's Guide
ODSTEXT	Creates paragraphs and lists that can be customized and nested an infinite number of times.	SAS Output Delivery System: User's Guide
SGMAP	Identifies the data sets needed for map areas, map response values, and overlay plots.	SAS ODS Graphics: Procedures Guide
SGPANEL	Creates a panel of graph cells for the values of one or more classification variables.	SAS ODS Graphics: Procedures Guide
SGPIE	Identifies the data set that contains the plot variables. The statement also gives you the option to specify a description, control automatic legends, and specify whether the chart background is opaque or transparent.	SAS ODS Graphics: Procedures Guide
SGPLOT	Creates one or more plots and overlays them on a single set of axes.	SAS ODS Graphics: Procedures Guide
SGRENDER	Produces graphical output from templates that are created with the Graph Template Language (GTL).	SAS ODS Graphics: Procedures Guide
SGSCATTER	Creates a paneled graph of scatter plots for multiple combinations of variables, depending on the plot statement that you use.	SAS ODS Graphics: Procedures Guide
SQL	Implements Structured Query Language (SQL) for SAS.	SAS SQL Procedure User's Guide
TEMPLATE	Enables you to customize the appearance of your SAS output.	SAS Output Delivery System: User's Guide

Procedure	Description	Publication
TRANTAB	Creates, edits, and displays customized translation tables, and enables you to view and modify translation tables that are supplied by SAS.	SAS National Language Support (NLS): Reference Guide
UNIVARIATE	Provides a variety of descriptive measures, graphical displays, and statistical methods, which you can use to summarize, visualize, analyze, and model the statistical distributions of numeric variables.	Base SAS Procedures Guide: Statistical Procedures

For information about all SAS procedures, see [SAS Procedures by Name and Product](#). The information in *SAS Procedures by Name and Product* is arranged alphabetically by the procedures' names and by their products' names.

Base SAS Procedures That Are No Longer Supported

Base SAS Procedures That Are Not Supported in SAS Viya Platform 85

Base SAS Procedures That Are Not Supported in SAS Viya Platform

The following SAS 9.4 procedures are not supported on SAS Viya platform.

Table 6.1 *Base SAS 9.4 Procedures That Are Not Supported on SAS Viya Platform*

Base SAS Procedure
AUTHLIB
CALENDAR
CHART
DISPLAY
FORMS
FSLIST
HDMD
METALIB

Base SAS Procedure

PLOT

PMENU

PRESENV

PRODUCT_STATUS

SOAP

TIMEPLOT

PART 2

Procedures

Chapter 7	
ACCELERATOR Procedure	91
Chapter 8	
APPEND Procedure	125
Chapter 9	
CATALOG Procedure	149
Chapter 10	
CIMPORT Procedure	179
Chapter 11	
COMPARE Procedure	209
Chapter 12	
CONTENTS Procedure	289
Chapter 13	
COPY Procedure	309
Chapter 14	
CPORT Procedure	341
Chapter 15	
DATASETS Procedure	365
Chapter 16	
DATEKEYS Procedure	527
Chapter 17	
DELETE Procedure	585
Chapter 18	
DS2 Procedure	597
Chapter 19	
DSTODS2 Procedure	631

Chapter 20	
EXPORT Procedure	643
Chapter 21	
FCMP Procedure	669
Chapter 22	
FCMP Special Functions and Call Routines	747
Chapter 23	
FEDSQL Procedure	793
Chapter 24	
FMTC2ITM Procedure	837
Chapter 25	
FONTREG Procedure	845
Chapter 26	
FORMAT Procedure	863
Chapter 27	
GATEWAY Procedure	993
Chapter 28	
GROOVY Procedure	1009
Chapter 29	
HADOOP Procedure	1023
Chapter 30	
HTTP Procedure	1045
Chapter 31	
IMPORT Procedure	1085
Chapter 32	
JAVAINFO Procedure	1121
Chapter 33	
JSON Procedure	1123
Chapter 34	
LUA Procedure	1163
Chapter 35	
MEANS Procedure	1227
Chapter 36	
MIGRATE Procedure	1343
Chapter 37	
OPTIONS Procedure	1363

Chapter 38	
OPTLOAD Procedure	1385
Chapter 39	
OPTSAVE Procedure	1391
Chapter 40	
PRINT Procedure	1397
Chapter 41	
PRINTTO Procedure	1497
Chapter 42	
PROTO Procedure	1519
Chapter 43	
PRTDEF Procedure	1547
Chapter 44	
PRTEXP Procedure	1567
Chapter 45	
PWENCODE Procedure	1573
Chapter 46	
PYTHON Procedure	1583
Chapter 47	
QDEVICE Procedure	1617
Chapter 48	
R Procedure	1663
Chapter 49	
RANK Procedure	1701
Chapter 50	
REGISTRY Procedure	1733
Chapter 51	
REPORT Procedure	1751
Chapter 52	
S3 Procedure	1923
Chapter 53	
SCAPROC Procedure	1957
Chapter 54	
SCOREACCEL Procedure	1971
Chapter 55	
SORT Procedure	2039

Chapter 56	
SQL Procedure	2099
Chapter 57	
SQOOP Procedure	2101
Chapter 58	
STANDARD Procedure	2109
Chapter 59	
STREAM Procedure	2139
Chapter 60	
SUMMARY Procedure	2153
Chapter 61	
TABULATE Procedure	2207
Chapter 62	
TRANSPOSE Procedure	2371
Chapter 63	
XSL Procedure	2411

ACCELERATOR Procedure

Overview: ACCELERATOR Procedure	91
What Does the ACCELERATOR Procedure Do?	91
Concepts: ACCELERATOR Procedure	92
Specifying One-Part, Two-Part, or Three-Part Table Names	92
Feature Comparison for PROC ACCELERATOR and PROC SCOREACCEL	96
Deployment Details for PROC ACCELERATOR	98
Syntax: ACCELERATOR Procedure	99
PROC ACCELERATOR Statement	100
STARTSESSION Statement	100
STOPSESSION Statement	102
PUBLISHMODEL Statement	102
RUNMODEL Statement	105
DELETEMODEL Statement	109
Examples: ACCELERATOR Procedure	110
Example 1: Publish, Run, and Delete a Model in Azure Databricks	110
Example 2: Publish, Run, and Delete a Model in Azure Synapse	113
Example 3: Publish, Run, and Delete a Model in SingleStore	116
Example 4: Publish, Run, and Delete a Model in Teradata	118
Example 5: Use an SQL Query to Read Data for Scoring	120
Example 6: Use Scala Code to Read and Write Data for Scoring	122

Overview: ACCELERATOR Procedure

What Does the ACCELERATOR Procedure Do?

PROC ACCELERATOR is an interactive procedure that provides an interface to external data sources for model publishing and scoring.

- PROC ACCELERATOR is an advanced replacement for the SCOREACCEL procedure. For differences between the two procedures, see [“Feature Comparison for PROC ACCELERATOR and PROC SCOREACCEL”](#) on page 96.
- Models are supported in the form of DATA step code, DS2 code, or analytic stores.
- The procedure supports the following data sources:
 - Databricks on Microsoft Azure as of [2024.11](#).
 - Microsoft Azure Synapse Analytics as of [2025.12](#).
 - SingleStore as of [2025.07](#). You must license and deploy the product SAS SpeedyStore.
 - Teradata as of [2026.08](#).
- After a model is published to an external data source, running the model invokes the SAS Embedded Process.
- For Databricks or Synapse, you use PROC ACCELERATOR to start and stop the SAS Embedded Process continuous session. For SingleStore or Teradata, you do not start or stop a session.
- Models can be removed from an external data source by using the DELETEMODEL statement.
- For a list of the supported scoring models and other information, see [“In-Database Model Scoring”](#) in *SAS In-Database Products: User’s Guide*.

Concepts: ACCELERATOR Procedure

Specifying One-Part, Two-Part, or Three-Part Table Names

Overview of Table Name Rules

Data sources support either two-part or three-part table names. Depending on the data source, you specify a catalog, database, schema, or a combination of these values, in the LIBNAME assignment. PROC ACCELERATOR uses the LIBNAME values as the default values for catalog, database, or schema in the MODELTABLENAME=, INPUTTABLENAME=, and OUTPUTTABLENAME= options.

For simplicity, it is recommended that you specify a one-part name in the PROC ACCELERATOR options, and take the defaults from the LIBNAME assignment for the catalog, database, or schema values.

PROC ACCELERATOR enables you to specify a one-part, two-part, or three-part table name in the MODELTABLENAME=, INPUTTABLENAME=, and OUTPUTTABLENAME= options. You can specify a two-part or three-part name if you need to override the defaults from the LIBNAME assignment. For example, you might want to create output data in a different database than the input data. An exception is Synapse, where you can override the database in the RUNMODEL= statements but not in the PUBLISHMODEL or DELETEMODEL statements. The behavior for each data source is described in the following topics.

PROC ACCELERATOR uses the authentication information specified in the LIBNAME to connect to the data source.

If a valid table name cannot be formed, PROC ACCELERATOR writes an error to the SAS log.

For all data sources, you must surround a name with backticks (``) if the name contains periods, reserved words, or special characters. The following example requires backticks around the table name `my.table` because the name contains a period:

```
modeltablename="myschema.`my.table`"
```

Table Names for Databricks

Databricks supports three-part table names, in this form:

```
catalog.schema.table
```

In the Spark LIBNAME statement for Databricks, use the SCHEMA= option, and do not use the DATABASE= option. Starting in [2025.12](#), you can use the CATALOG= LIBNAME option to specify the Databricks Unity Catalog.

In releases prior to 2025.12, you must use JDBC driver properties to specify the catalog. If you prefer, you can continue to use JDBC driver properties to specify the catalog, or specify both catalog and schema. The ConnCatalog= and ConnSchema= properties are supported in the URI= or PROPERTIES= LIBNAME options.

If you specify the catalog and schema in the PROPERTIES= option and you also specify them in the CATALOG= and SCHEMA LIBNAME options, then the values in the PROPERTIES= option take precedence.

The following table describes the PROC ACCELERATOR behavior for Databricks.

Table 7.1 Forming Three-Part Table Names for Databricks

Value in the MODELTABLENAME=, INPUTTABLENAME=, or OUTPUTTABLENAME= Option	PROC ACCELERATOR Behavior
One-part name	Read the option value as the table name. Use the catalog from the CATALOG= LIBNAME option or from the ConnCatalog= driver property. Use the schema from the SCHEMA= LIBNAME option or from the ConnSchema= driver property.
Two-part name	Read the option value as <i>schema.table</i> . Use the catalog from the CATALOG= LIBNAME option or from the ConnCatalog= driver property.
Three-part name	Use the value as is.

Table Names for SingleStore (SAS SpeedyStore)

SingleStore supports two-part table names, in this form:

```
database.table
```

In the SingleStore LIBNAME statement, specify the database in the DATABASE= option. SingleStore does not support a schema name.

The following table describes the PROC ACCELERATOR behavior for SingleStore.

Table 7.2 Forming Two-Part Table Names for SingleStore

Value in the MODELTABLENAME=, INPUTTABLENAME=, or OUTPUTTABLENAME= Option	PROC ACCELERATOR Behavior
One-part name	Use the database from the LIBNAME statement.
Two-part name	Use the option value as is.
Three-part name	Three-part names are not supported in SingleStore. Write an error to the SAS log.

Table Names for Synapse

Microsoft Azure Synapse Analytics requires three-part table names, in this form:

```
database.schema.table
```

Use the Microsoft SQL Server LIBNAME statement for your connection. You must connect to a dedicated SQL pool in the Synapse workspace. The pool must be unpaused. Specify the schema in the SCHEMA= LIBNAME option. You must specify the Database= connection option within the NOPROMPT= option. The database value is the name of the dedicated SQL pool. Do not use the DATASRC= option.

The following table describes the PROC ACCELERATOR behavior for Synapse. When you publish or delete a model, you cannot override the database value from the LIBNAME assignment. However, when you run the model, you can specify a different database in the MODELTABLENAME=, INPUTTABLENAME=, or OUTPUTTABLENAME= value.

Table 7.3 Forming Three-Part Table Names for Synapse

Value in the MODELTABLENAME=, INPUTTABLENAME=, or OUTPUTTABLENAME= Option	PROC ACCELERATOR Behavior	
	PUBLISHMODEL or DELETEMODEL Statement	RUNMODEL Statement
One-part name	Use the schema and database from the LIBNAME statement.	Read the option value as the table name. Use the schema and database from the LIBNAME statement.
Two-part name	Use the value as <i>schema.table</i> . Use the database from the LIBNAME statement.	Read the option value as <i>schema.table</i> . Use the database from the LIBNAME statement.
Three-part name	Use the value if the database in the three-part name matches the database from the LIBNAME statement. If the database does not match, write an error to the SAS log.	Use the option value as is.

Table Names for Teradata

Teradata supports two-part table names, in this form:

```
database.table
```

In the Teradata LIBNAME statement, use the DATABASE= option, and do not use the SCHEMA= option. Although the SCHEMA= option is supported by the LIBNAME statement, it is used to view or modify another user's tables. For in-database processing, this usage is discouraged.

The following table describes the PROC ACCELERATOR behavior for Teradata.

Table 7.4 *Forming Three-Part Table Names for Teradata*

Value in the MODELTABLENAME=, INPUTTABLENAME=, or OUTPUTTABLENAME= Option	PROC ACCELERATOR Behavior
One-part name	Read the option value as the table name. Use the database from the LIBNAME statement.
Two-part name	Use the option value as is.
Three-part name	Use the option value as is. However, three-part names might cause errors.

Feature Comparison for PROC ACCELERATOR and PROC SCOREACCEL

The following table summarizes differences from [PROC SCOREACCEL](#).

Table 7.5 *Support for Tasks and Features for In-Database Scoring*

Task or Feature	PROC ACCELERATOR	PROC SCOREACCEL
Deploy the SAS Embedded Process. ¹	For Databricks or Synapse, you deploy the accelserver RPM file.	For Databricks or Synapse, you deploy the sepcorespark RPM file.
	For SingleStore, you must license and deploy the product SAS SpeedyStore, for both procedures.	

Task or Feature	PROC ACCELERATOR	PROC SCOREACCEL
	For Teradata, you deploy the sepcoretera RPM file, for both procedures.	
Start or stop the SAS Embedded Process continuous session.	For Databricks or Synapse, submit the procedure with a STARTSESSION or STOPSESSION statement.	For Databricks or Synapse, submit the CAS action sparkEmbeddedProcess.startSparkEP.startSparkEP or sparkEmbeddedProcess.startSparkEP.stopSparkEP.
	For SingleStore or Teradata, you do not start or stop a continuous session.	
Establish a connection to the data source.	Submit a LIBNAME statement or use another method to assign a SAS library.	Submit a CASLIB statement or use another method to assign a CAS library.
Reference the data source within the procedure.	Specify the libref.	Specify the casref.
Reference the input data or output data.	One-part, two-part, or three-part table names are supported.	One-part name or a two-part table names are supported.
	For Databricks or Synapse, you can read the input data and write the output data by using custom Scala code.	Scala code is not supported.
Interact with a CAS server.	There is no interaction with CAS.	PROC SCOREACCEL calls CAS actions to complete the tasks. You can use the Model Publishing and Scoring action set directly, instead of using PROC SCOREACCEL.
Deprecated features	Deprecated options are omitted.	Deprecated options are available.
Supported data sources	Azure Databricks Azure Synapse SingleStore Teradata	CAS Amazon EMR AWS Databricks Azure Databricks Azure Synapse Cloudera SingleStore

Task or Feature	PROC ACCELERATOR	PROC SCOREACCEL
	Teradata	

1 For more information, see [“Deployment Details for PROC ACCELERATOR” on page 98](#).

Deployment Details for PROC ACCELERATOR

Deployment Details for Databricks and Synapse

It is important to note the difference in deployment for PROC ACCELERATOR compared to PROC SCOREACCEL. Here are the details:

- An additional RPM file is now available for the SAS Embedded Process for Spark. The new deployment is referred to as the *accelserver* RPM file. The previous deployment is referred to as the *sepcorespark* RPM file.
 - For Databricks, the *accelserver* RPM and the *sepcorespark* RPM are supported (on separate clusters). However, the *sepcorespark* RPM is deprecated, and customers are encouraged to use the *accelserver* RPM instead.
 - For Synapse, starting with [2026.04](#), only the *accelserver* RPM is supported. In addition, a new pre-installation task is required in order to download an additional object library and create a wheel file. Then you install that wheel file when you install the *sasindb* wheel. The *sepcorespark* RPM is no longer supported.
- PROC ACCELERATOR is supported if you install the Python wheel from the *accelserver* RPM file. The Python wheel from the *sepcorespark* RPM file does not support PROC ACCELERATOR.
- PROC SCOREACCEL, the Model Publishing and Scoring action set, and the SAS Embedded Process for Spark action set are supported if you install the Python wheel from the *sepcorespark* RPM file. The Python wheel from the *accelserver* RPM file does not support PROC SCOREACCEL or the action sets.
- For Databricks and Synapse, you cannot install both RPMs on the same cluster. If you need the functionality of both RPMs, they must be installed on separate clusters.

For deployment information, see [SAS In-Database Products: Deployment and Administration Guide](#).

Deployment Details for SingleStore

The procedure supports SingleStore if you license and deploy the product SAS SpeedyStore. The procedure supports the integrated instance of SingleStore database only. The procedure does not support an external database. No additional deployment or configuration tasks are required for PROC ACCELERATOR.

For deployment information, see [SAS SpeedyStore: Administration and Configuration Guide](#).

Deployment Details for Teradata

The SAS Embedded Process for Teradata is deployed from the sepcoretera RPM. No additional deployment or configuration tasks are required for PROC ACCELERATOR. You must install SAS Embedded Process 2026.08 or later.

For deployment information, see [SAS In-Database Products: Deployment and Administration Guide](#).

Syntax: ACCELERATOR Procedure

PROC ACCELERATOR *required-argument*;

STARTSESSION *required-arguments <optional-arguments>*;

STOPSESSION;

PUBLISHMODEL *required-arguments <optional-argument>*;

RUNMODEL *required-arguments <optional-arguments>*;

DELETEMODEL *required-argument*;

Statement	Task	Example
PROC ACCELERATOR	Submit statements to start or stop a continuous session, or publish, run, or delete a model in a data source	Ex. 1, Ex. 2, Ex. 3, Ex. 4, Ex. 5, Ex. 6
STARTSESSION	Start the SAS Embedded Process continuous session	Ex. 1, Ex. 2, Ex. 6
STOPSESSION	Stop the SAS Embedded Process continuous session	Ex. 1, Ex. 2, Ex. 6
PUBLISHMODEL	Publish a model to a data source	Ex. 1, Ex. 2, Ex. 3, Ex. 4, Ex. 5, Ex. 6

Statement	Task	Example
<code>RUNMODEL</code>	Run a model in a data source	Ex. 1, Ex. 2, Ex. 3, Ex. 4, Ex. 5, Ex. 6
<code>DELETEMODEL</code>	Delete a model from a data source	Ex. 1, Ex. 2, Ex. 3, Ex. 4, Ex. 5, Ex. 6

PROC ACCELERATOR Statement

Starts or stops a continuous session, or publishes, runs, or deletes a model in a data source.

Syntax

PROC SCOREACCEL `LIBRARY=libref`;

Required Argument

LIBRARY=libref

specifies the name of the SAS library where the model is run. A libref is a name that you use to reference a SAS library. You must assign the SAS library before you can specify its libref in this option. The LIBNAME statement is a common way of assigning a SAS library. For more information, see [SAS/ACCESS for Relational Databases: Reference](#).

STARTSESSION Statement

Starts the SAS Embedded Process continuous session.

Note: Do not use the STARTSESSION and STOPSESSION statements with SingleStore or Teradata.

Syntax

STARTSESSION

`PLATFORM=DATABRICKS | SYNAPSE`
`PLATFORMENDPOINT="rest-url"`
`PLATFORMPASSWORD=endpoint-password`
`PLATFORMUSER=endpoint-user-identifier`
`<startsession-optional-arguments>;`

Required Arguments

PLATFORM=CLUDERA | DATABRICKS | SYNAPSE

specifies the target platform for model publishing and scoring. This option is required for Azure Synapse, because PROC ACCELERATOR cannot determine the platform from the LIBNAME assignment. In that case, if you have not specified the PLATFORM= option, an error is written to the SAS log.

Requirement The PLATFORM= option is required for Azure Synapse and is optional for other data sources.

PLATFORMENDPOINT="rest-url"

specifies the URL that is used to submit execution requests over a REST interface to services such as Apache Livy or Databricks execution context. For Databricks, this value is the workspace URL. For Synapse, this value is the Livy REST URL.

See [“Specifying the Workspace URL for Databricks” in SAS In-Database Products: User’s Guide](#)

[“Forming the Livy REST URL for Synapse” in SAS In-Database Products: User’s Guide](#)

PLATFORMPASSWORD=endpoint-password

specifies the Livy endpoint application secret.

Requirement The PLATFORMPASSWORD= option is required for Azure Synapse.

PLATFORMUSER=endpoint-user-identifier

specifies the Livy endpoint user name.

This value is the application ID.

Requirement The PLATFORMUSER= option is required for Azure Synapse.

Optional Arguments

AUTORESTART=

automatically restarts a session if it is terminated on the server side due to a timeout or other events.

Default TRUE

Restriction The AUTORESTART= option is supported for Synapse only.

MAXUNBOUNDEDLENGTH=maximum-unbounded-column-length

specifies the maximum length for unbounded data types (such as BLOB, CLOB, TEXT, or long STRING types) when they are Read into the SAS Embedded Process. The option ensures consistent handling of very large character or binary columns from the database. If data in an input column exceeds the MAXUNBOUNDEDLENGTH= value, truncation could occur during Read. Truncation could occur during Write, but only for an output column that is also

an input column. The option does not apply to any output column that is not an input column.

You can set the MAXUNBOUNDEDLENGTH= option in either the STARTSESSION statement or RUNMODEL statement. When you specify the option in RUNMODEL, you override the value in STARTSESSION. Note that the DBMAX_TEXT= LIBNAME option does not affect processing in PROC ACCELERATOR.

Default 32767

Note The MAXUNBOUNDEDLENGTH= option was added in [2025.09](#).

TIMEOUT=*n*

specifies the number of minutes the Livy session waits for execution requests. If the time-out period is reached, the Livy session terminates. A value of zero disables the timeout. If the value is not zero, it must be at least 15 minutes.

Default 60 minutes

Restriction The TIMEOUT= option is not supported for Databricks.

TRACE

specifies whether the SAS Embedded Process runs with traces.

STOPSESSION Statement

Stops the SAS Embedded Process continuous session.

Notes:

For Databricks, when you are finished running models, you can stop the SAS Embedded Process continuous session by submitting PROC ACCELERATOR with the STOPSESSION statement. If you do not stop the session, it terminates a few minutes after you end SAS.

Do not use the STARTSESSION and STOPSESSION statements with SingleStore or Teradata.

Syntax

STOPSESSION;

PUBLISHMODEL Statement

Publishes a model to a data source.

Syntax

PUBLISHMODEL

MODELTABLENAME=*"fully-qualified-table-name"*

MODELTYPE=DATASTEP | DS2

PLATFORM=DATABRICKS | SINGLESTORE | SYNAPSE | TERADATA

STOREFILES=*file-specification* | **PROGRAMFILE**=*file-specification*

VARFILE=*file-path*

<*publishmodel-optional-argument*>;

Required Arguments

MODELTABLENAME=*"fully-qualified-table-name"*

specifies the name of the table to which the model is published. You can specify a one-part, two-part, or three-part name. For more information, see [“Specifying One-Part, Two-Part, or Three-Part Table Names” on page 92](#).

Requirement Surround a name with backticks (`) if the name contains periods, reserved words, or special characters.

Examples The following example references a catalog, a schema, and a table:
`modeltablename="mycatalog.myschema.mytable"`

The following example references a schema and a table:
`modeltablename="myschema.mytable"`

The following example requires backticks around the table name **my.table** because the name contains a period:
`modeltablename="myschema.`my.table`"`

MODELTYPE=DATASTEP | DS2

specifies the type of the input model program.

DATASTEP

specifies that the input model program is DATA step code.

DS2

specifies that the input model program is DS2 code.

Default DS2

Note The **MODELTYPE**= option is required only for DATA step models. The DATA step code is converted to DS2 code before being bundled into an item store.

PLATFORM=CLOUDERA | DATABRICKS | SINGLESTORE | SYNAPSE | TERADATA

specifies the target platform for model publishing and scoring. This option is required for Azure Synapse, because PROC ACCELERATOR cannot determine the platform from the LIBNAME assignment. In that case, if you have not specified the **PLATFORM**= option, an error is written to the SAS log.

Requirement The **PLATFORM**= option is required for Azure Synapse and is optional for other data sources.

STOREFILES=*("file-path-1" | fileref-1 <,"file-path-2" | fileref-2, ...>)*

specifies a list of analytic store .sasast files, one for each analytic store to be published. The list of files can be a mixture of file path names and file references.

You must specify either STOREFILES= or PROGRAMFILE= or both options. Some analytic store models require only the STOREFILES= option. Some DATA step and DS2 models require only the PROGRAMFILE= option. Some models consist of DATA step code, DS2 code, and one or more ASTORE files. In that case, you specify both STOREFILES= and PROGRAMFILE= options.

If you publish an analytic store without a model program, then a DS2 model program is generated from the analytic store.

To specify an associated model program, use the PROGRAMFILE= option. When an analytic store is published, the associated model program is usually DS2 code.

Interactions You must specify either STOREFILES= or PROGRAMFILE= or both options.

The STOREFILES= option is used only when MODELTYPE=DS2.
The option is not used when MODELTYPE=DATASTEP.

PROGRAMFILE=*"file-path" | fileref*

You must specify either STOREFILES= or PROGRAMFILE= or both options. Some analytic store models require only the STOREFILES= option. Some DATA step and DS2 models require only the PROGRAMFILE= option. Some models consist of DATA step code, DS2 code, and one or more ASTORE files. In that case, you specify both STOREFILES= and PROGRAMFILE= options.

Interaction You must specify either STOREFILES= or PROGRAMFILE= or both options.

VARFILE=*"file-path" | fileref*

specifies the file that contains the variable metadata XML to be used during translation of an input DATA step model program to DS2.

Interaction The VARFILE= option is required by some models when MODELTYPE=DATASTEP. The option is not used when MODELTYPE=DS2.

Optional Argument

FORMATSFILE=*"file-path" | fileref*

specifies the file that contains the user-defined format XML definition to be published.

Note This option is valid for DS2 models or DATA step models.

RUNMODEL Statement

Runs a model in a data source.

Syntax

RUNMODEL

```
INPUTQUERY='sql-query' | INPUTSCALACODE="scala-code" |
INPUTTABLENAME="fully-qualified-table-name"
MODELTABLENAME="fully-qualified-table-name"
OUTKEY="column-specification" | OUTPUTSCALACODE="scala-code" |
OUTPUTTABLENAME="fully-qualified-table-name"
PLATFORM=DATABRICKS | SINGLESTORE | SYNAPSE | TERADATA
<runmodel-optional-arguments>;
```

Required Arguments

INPUTQUERY='sql-query' | INPUTSCALACODE="scala-code" |
INPUTTABLENAME="fully-qualified-table-name"

INPUTQUERY='sql-query'

is a SELECT statement that returns a table results set.

The SQL query must be supported by the data source.

Restriction The INPUTQUERY= option is not supported for Databricks.

Note This option was added in [2025.07](#).

Example Here is an example for SingleStore:

```
inputquery='select a, b, c from mytable where a > 1'
```

Example [“Example 5: Use an SQL Query to Read Data for Scoring” on page 120](#)

INPUTSCALACODE="scala-code"

is a string of Scala code.

Use this option if you want to read your input data by using custom Scala code. Your Scala code must be a single statement that returns a Spark dataset.

In the INPUTSCALACODE= string, do not include the variable declaration.

Restriction The INPUTSCALACODE= option is not supported for SingleStore or Teradata.

Note This option was added in [2025.07](#).

- Tip** If you use double quotation marks (") inside the Scala code string, and the entire string is contained within double quotation marks ("), then you must escape the internal quotation marks. To escape quotation marks, use a pair of quotation marks together (") instead of one (").
- Examples** Here is an example of Scala code that you would submit in a Databricks notebook to read a Parquet file:
- ```
val data = spark.read.parquet("/data/myparquet")
```
- Here is the code to read the same file from Databricks in PROC ACCELERATOR. The double quotation marks (") are not escaped, because the string is enclosed within single quotation marks (').
- ```
inputscalacode='spark.read.parquet("/data/myparquet")'
```
- Here is an example of Scala code used in a Databricks notebook to read a JSON file:
- ```
val df = spark.read.format("json").load("myfile.json")
```
- Here is the code to read the same file from Databricks in PROC ACCELERATOR. Notice that the quotation marks within the string are escaped by using a pair of quotation marks.
- ```
inputscalacode="spark.read.format("""json""").load("""myfile.json""")"
```
- Example** [“Example 6: Use Scala Code to Read and Write Data for Scoring” on page 122](#)

INPUTTABLENAME="fully-qualified-table-name"

specifies the name of the input table.

You can specify a one-part, two-part, or three-part name. For more information, see [“Specifying One-Part, Two-Part, or Three-Part Table Names” on page 92](#).

- Restriction** You must specify either the INPUTQUERY= option or the INPUTSCALACODE= option or the INPUTTABLENAME= option. You cannot specify more than one of the input options.
- Requirement** Surround a name with backticks (`) if the name contains periods, reserved words, or special characters.

MODELTABLENAME="fully-qualified-table-name"

specifies the name of the table to which the model is published. You can specify a one-part, two-part, or three-part name. For more information, see [“Specifying One-Part, Two-Part, or Three-Part Table Names” on page 92](#).

- Requirement** Surround a name with backticks (`) if the name contains periods, reserved words, or special characters.
- Examples** The following example references a catalog, a schema, and a table:
- ```
modeltablename="mycatalog.myschema.mytable"
```
- The following example references a schema and a table:
- ```
modeltablename="myschema.mytable"
```


The following example requires backticks around the table name `my.table` because the name contains a period:
`modeltablename="myschema.`my.table`"`

**OUTKEY="column-specification" | OUTPUTSCALACODE="scala-code" |
 OUTPUTTABLENAME="fully-qualified-table-name"**

OUTKEY="column-1 <column-2 ...>"

OUTKEY=(column-1 <column-2 ...>)

OUTKEY=("column-1" <"column-2" ...>)

specifies the name of one or more columns used for the primary index of the output table that is created by the SAS Embedded Process.

Restriction The OUTKEY= argument is supported for Teradata only.

OUTPUTSCALACODE="scala-code"

is a string of Scala code.

Use this option if you want to write your output data by using custom Scala code.

The string must begin with the function.

Restriction The OUTPUTSCALACODE= option is not supported for SingleStore, or Teradata.

Note This option was added in [2025.07](#).

Tip If you use double quotation marks (") inside the Scala code string, and the entire string is contained within double quotation marks ("), then you must escape the internal quotation marks. To escape quotation marks, use a pair of quotation marks together (") instead of one (").

Example Here is an example of Scala code that uses the WRITE function, which you would submit in a Databricks notebook:
`outputdataset.write.parquet("/data/myparquet_output")`
 Here is code to use the WRITE function in PROC ACCELERATOR for Databricks:
`outputscalacode='write.parquet("/data/myparquet_output")'`

Example ["Example 6: Use Scala Code to Read and Write Data for Scoring" on page 122](#)

OUTPUTTABLENAME="fully-qualified-table-name"

specifies the name of the output table.

You can specify a one-part, two-part, or three-part name. For more information, see ["Specifying One-Part, Two-Part, or Three-Part Table Names" on page 92](#).

Restriction You must specify one of these options: OUTKEY=, OUTPUTSCALACODE=, or OUTPUTTABLENAME=. You cannot specify more than one of these options.

Requirement Surround a name with backticks (` `) if the name contains periods, reserved words, or special characters.

PLATFORM=CLOUDERA | DATABRICKS | SINGLESTORE | SYNAPSE | TERADATA

specifies the target platform for model publishing and scoring. This option is required for Azure Synapse, because PROC ACCELERATOR cannot determine the platform from the LIBNAME assignment. In that case, if you have not specified the PLATFORM= option, an error is written to the SAS log.

Requirement The PLATFORM= option is required for Azure Synapse and is optional for other data sources.

Optional Arguments

KEEPLISTCOLUMNS=("*column-1*"<...<*column-n*">)

specifies the name of one or more columns to keep in the output data. If a column is not specified, it is dropped from the output data. If you want to keep all of the columns in the output data, then do not use the KEEPLISTCOLUMNS= option.

Note This option was added in [2025.06](#).

MAXUNBOUNDEDLENGTH=*maximum-unbounded-column-length*

specifies the maximum length for unbounded data types (such as BLOB, CLOB, TEXT, or long STRING types) when they are Read into the SAS Embedded Process. The option ensures consistent handling of very large character or binary columns from the database. If data in an input column exceeds the MAXUNBOUNDEDLENGTH= value, truncation could occur during Read. Truncation could occur during Write, but only for an output column that is also an input column. The option does not apply to any output column that is not an input column.

You can set the MAXUNBOUNDEDLENGTH= option in either the STARTSESSION statement or RUNMODEL statement. When you specify the option in RUNMODEL, you override the value in STARTSESSION. Note that the DBMAX_TEXT= LIBNAME option does not affect processing in PROC ACCELERATOR.

Default 32767

Note The MAXUNBOUNDEDLENGTH= option was added in [2025.09](#).

OPTIONS="*options-string*"

specifies properties that are passed to the SAS Embedded Process. For full list of supported options, see [options-list](#) in the SAS_SCORE_EP stored procedure.

Restriction The OPTIONS= argument is supported for Teradata only.

Note This option was added in 2026.08.

Example `options = "loglevel=info;encoding=unicode;"`

Details

You can mix input and output objects:

- You must specify one and only one of the input options: INPUTQUERY= or INPUTSCALACODE= or INPUTTABLENAME=.
- You must specify one and only one of the output options: OUTPUTSCALACODE= or OUTPUTTABLENAME=.
- You can mix the input and output options. For example, you can read your input data by using INPUTSCALACODE= and write your output data by using OUTPUTTABLENAME=.

DELETEMODEL Statement

Deletes a model from a data source.

Syntax

DELETEMODEL

MODELTABLENAME=*fully-qualified-table-name*

PLATFORM=DATABRICKS | SINGLESTORE | SYNAPSE | TERADATA;

Required Arguments

MODELTABLENAME=*"fully-qualified-table-name"*

specifies the name of the table to which the model is published. You can specify a one-part, two-part, or three-part name. For more information, see [“Specifying One-Part, Two-Part, or Three-Part Table Names” on page 92](#).

Requirement Surround a name with backticks (`) if the name contains periods, reserved words, or special characters.

Examples The following example references a catalog, a schema, and a table:
`modeltablename="mycatalog.myschema.mytable"`

The following example references a schema and a table:
`modeltablename="myschema.mytable"`

The following example requires backticks around the table name **my.table** because the name contains a period:
`modeltablename="myschema.`my.table`"`

PLATFORM=CLOUDERA | DATABRICKS | SINGLESTORE | SYNAPSE | TERADATA

specifies the target platform for model publishing and scoring. This option is required for Azure Synapse, because PROC ACCELERATOR cannot determine the platform from the LIBNAME assignment. In that case, if you have not specified the PLATFORM= option, an error is written to the SAS log.

Requirement The PLATFORM= option is required for Azure Synapse and is optional for other data sources.

Examples: ACCELERATOR Procedure

Example 1: Publish, Run, and Delete a Model in Azure Databricks

Features:

- LIBNAME statement
- PROC ACCELERATOR statement
- PUBLISHMODEL statement
- STARTSESSION statement
- RUNMODEL statement
- STOPSESSION statement
- DELETEMODEL statement

Details

In this example, PROC ACCELERATOR publishes a model to a Spark table in Databricks on Microsoft Azure. Then PROC ACCELERATOR runs the model in Databricks by using the SAS Embedded Process for Spark. Note that the AZURETENANTID= system option is not necessary for in-database model scoring with PROC ACCELERATOR, because the procedure publishes models to a Spark table, not to the file system.

IMPORTANT SAS Embedded Process for Spark requires a different deployment for PROC ACCELERATOR than for PROC SCOREACCEL. If you need the functionality of both RPMs, they must be installed on separate clusters. For more information, see [“Deployment Details for Databricks and Synapse” on page 98](#).

Note: In 2023.10, SAS changed the redistributed JDBC drivers used by SAS/ACCESS Interface to Hadoop and SAS/ACCESS Interface to Spark. Changes

might be needed in your connection options. For more information, see [“Updates Might Be Needed Due to Driver Change” in SAS In-Database Products: User’s Guide](#).

Program

```
libname spklib spark platform=databricks
    user=token
    password="authentication-token"
    server="myserver.azuredatabricks.net"
    catalog="my-catalog-name"
    schema="my-schema-name"
    port=10016
    httppath="my-http-path";

proc accelerator library=spklib;
    publishModel
        modeltablename="my-model-table"
        programfile="my-program-file"
        storefiles="my-analytic-store-file";
run;
quit;

proc accelerator library=spklib;
    startsession
        platformendpoint="https://my-identifier.databricks.net";
run;
quit;

proc accelerator library=spklib;
    runmodel
        modeltablename="my-model-table"
        inputtablename="catalog.schema.input-table"
        outputtablename="catalog.schema.output-table";
run;
quit;

proc accelerator library=spklib;
    stopsession;
run;
quit;

proc accelerator library=spklib;
    deletemodel
        modeltablename="my-model-table"
    ;
run;
quit;
```

Program Description

Assign a SAS library, using SAS/ACCESS Interface to Spark. Specify PLATFORM=DATABRICKS, connection options, and any other LIBNAME options

that you need. For more information about assigning this SAS library, see [“SAS/ACCESS Interface to Spark” in SAS/ACCESS for Relational Databases: Reference](#).

```
libname spklib spark platform=databricks
      user=token
      password="authentication-token"
      server="myserver.azuredatabricks.net"
      catalog="my-catalog-name"
      schema="my-schema-name"
      port=10016
      httpath="my-http-path";
```

Publish the model. Use the LIBRARY= option in the PROC ACCELERATOR statement to reference the libref.

```
proc accelerator library=spklib;
  publishModel
    modeltablename="my-model-table"
    programfile="my-program-file"
    storefiles="my-analytic-store-file";
run;
quit;
```

Start the SAS Embedded Process continuous session. The procedure starts a continuous session and associates the specified endpoint with the SAS library location. The PLATFORMENDPOINT= value is the Databricks workspace URL.

```
proc accelerator library=spklib;
  startsession
    platformendpoint="https://my-identifier.databricks.net";
run;
quit;
```

Run the model. You must start a SAS Embedded Process continuous session before you run a model.

```
proc accelerator library=spklib;
  runmodel
    modeltablename="my-model-table"
    inputtablename="catalog.schema.input-table"
    outputtablename="catalog.schema.output-table";
run;
quit;
```

Stop the SAS Embedded Process continuous session. When you are finished running models, you can stop the SAS Embedded Process continuous session by submitting PROC ACCELERATOR with the STOPSESSION statement. If you do not stop the session, it terminates a few minutes after you end SAS.

```
proc accelerator library=spklib;
  stopsession;
run;
quit;
```

Delete the model.

```
proc accelerator library=spklib;
  deletemodel
```

```
modeltablename="my-model-table"  
;  
run;  
quit;
```

Example 2: Publish, Run, and Delete a Model in Azure Synapse

Features:

- LIBNAME statement
- PROC ACCELERATOR statement
- PUBLISHMODEL statement
- RUNMODEL statement
- DELETEMODEL statement

Details

In this example, PROC ACCELERATOR publishes a model to an SQL Server table (also known as a dedicated SQL pool table) in Microsoft Azure Synapse Analytics. Then PROC ACCELERATOR runs the model by using the SAS Embedded Process for Spark. The connection uses SAS/ACCESS Interface to Microsoft SQL Server.

IMPORTANT For Synapse, starting with 2026.06, changes are required due to an update in the DataDirect Microsoft SQL Server ODBC driver, which is used for connections to Microsoft SQL Server. If you use the NOPROMPT=LIBNAME option to supply connection values, then you must add three properties in the connection string. If you use a DSN connection, then the properties can be set on the specific DSN in that `odbc.ini`, rather than in the LIBNAME statement. The properties are `KeepConnectionActive=0;ConnectionRetryCount=3;ConnectionRetryDelay=2;`. Without these properties, you receive an SSL error message.

In addition, starting with 2026.04, a new pre-installation task is required in order to download an additional object library and create a wheel file. Then you install that wheel file when you install the `sasindb` wheel. For more information, see [SAS In-Database Products: Deployment and Administration Guide](#).

Program

```
libname mylib sqlsvr  
user=my-user-name
```

```

        password="my-password"
        schema="my-schema"
        noprompt="DRIVER=SAS ACCESS to MS SQL Server;
                KeepConnectionActive=0;
                ConnectionRetryCount=3;
                ConnectionRetryDelay=2;
                Hostname=synapse-workspace-name.sql.azuresynapse.net;
                Port=my-port-number;
                Database=my-database-name
                ...;";

proc accelerator library=mylib;
  publishModel
    platform=synapse
    modeltablename="my-model-table"
    programfile="my-program-file"
    storefiles="my-analytic-store-file";
run;
quit;

proc accelerator library=mylib;
  startsession
    platform=synapse
    platformendpoint=""
    platformuser="endpoint/livyApi/versions/livyApiVersion/
sparkPools/sparkPoolName"
    platformpassword="my-secret";
run;
quit;

proc accelerator library=mylib;
  runmodel
    platform=synapse
    modeltablename="my-model-table"
    inputtablename="my-input-table"
    outputtablename="my-output-table";
run;
quit;

proc accelerator library=mylib;
  stopsession;
run;
quit;

proc accelerator library=mylib;
  deletemodel
    platform=synapse
    modeltablename="my-model-table"
    ;
run;
quit;

```

Program Description

Assign a SAS library, using SAS/ACCESS Interface to Microsoft SQL Server. You must connect to a dedicated SQL pool in the Synapse workspace. The pool must be

unpaused. Specify the schema in the SCHEMA= LIBNAME option. You must specify the Database= connection option within the NOPROMPT= option. The database value is the name of the dedicated SQL pool. Do not use the DATASRC= option. For additional options, see [“SAS/ACCESS Interface to Microsoft SQL Server” in SAS/ACCESS for Relational Databases: Reference](#).

```
libname mylib sqlsvr
    user=my-user-name
    password="my-password"
    schema="my-schema"
    noprompt="DRIVER=SAS ACCESS to MS SQL Server;
        KeepConnectionActive=0;
        ConnectionRetryCount=3;
        ConnectionRetryDelay=2;
        Hostname=synapse-workspace-name.sql.azuresynapse.net;
        Port=my-port-number;
        Database=my-database-name
        ...;";
```

Publish the model. Use the LIBRARY= option in the PROC ACCELERATOR statement to reference the libref. For Synapse, the PLATFORM= option is required in every PROC ACCELERATOR statement except STOPSESSION. You must specify either PROGRAMFILE= or STOREFILES= or both options, depending on your model. See the syntax in [“PUBLISHMODEL Statement” on page 102](#) for more information.

```
proc accelerator library=mylib;
    publishModel
        platform=synapse
        modeltablename="my-model-table"
        programfile="my-program-file"
        storefiles="my-analytic-store-file";
run;
quit;
```

Start the SAS Embedded Process continuous session. The procedure starts a continuous session and associates the specified endpoint with the SAS library location. The PLATFORM=, PLATFORMENDPOINT=, PLATFORMUSER=, and PLATFORMPASSWORD= options are required for Synapse. For more information about the endpoint, see [“Forming the Livy REST URL for Synapse” in SAS In-Database Products: User’s Guide](#).

```
proc accelerator library=mylib;
    startsession
        platform=synapse
        platformendpoint=""
        platformuser="endpoint/livyApi/versions/livyApiVersion/
sparkPools/sparkPoolName"
        platformpassword="my-secret";
run;
quit;
```

Run the model. You must start a SAS Embedded Process continuous session before you run a model.

```
proc accelerator library=mylib;
    runmodel
        platform=synapse
        modeltablename="my-model-table"
```

```

inputtablename="my-input-table"
outputtablename="my-output-table";
run;
quit;

```

Stop the SAS Embedded Process continuous session. When you are finished running models, you can stop the SAS Embedded Process continuous session by submitting PROC ACCELERATOR with the STOPSESSION statement. If you do not stop the session, it terminates a few minutes after you end SAS.

```

proc accelerator library=mylib;
  stopsession;
run;
quit;

```

Delete the model.

```

proc accelerator library=mylib;
  deletemodel
    platform=synapse
    modeltablename="my-model-table"
  ;
run;
quit;

```

Example 3: Publish, Run, and Delete a Model in SingleStore

Features:	LIBNAME statement
	PROC ACCELERATOR statement
	PUBLISHMODEL statement
	RUNMODEL statement
	DELETEMODEL statement

Details

In this example, PROC ACCELERATOR publishes a model to a table in a SingleStore database. Then PROC ACCELERATOR runs the model in SingleStore by using the SAS Embedded Process. The SingleStore instance that is included in the product SAS SpeedyStore is supported. An external SingleStore database is not supported. For more information, see [“Deployment Details for SingleStore” on page 99](#).

Note: For SingleStore, you do not start the SAS Embedded Process continuous session before running a model.

Program

```

libname mylib sstore
    user=my-user-name
    password="my-password"
    database=my-database
    server=my-server
    port=my-port-number;

proc accelerator library=mylib;
    publishModel
        modeltablename="model_table"
        programfile="my_program_file"
        storefiles="my_analytic_store_file";
run;
quit;

proc accelerator library=mylib;
    runmodel
        modeltablename="model_table"
        inputtablename="my_input_table"
        outputtablename="output_table";
run;
quit;

proc accelerator library=mylib;
    deletemodel
        modeltablename="model_table";
run;
quit;

```

Program Description

Assign a SAS library, using SAS/ACCESS Interface to SingleStore. For more information about this LIBNAME statement, see [“SAS/ACCESS Interface to SingleStore” in SAS/ACCESS for Relational Databases: Reference](#).

```

libname mylib sstore
    user=my-user-name
    password="my-password"
    database=my-database
    server=my-server
    port=my-port-number;

```

Publish the model. The LIBRARY= option in the PROC ACCELERATOR statement references the libref.

```

proc accelerator library=mylib;
    publishModel
        modeltablename="model_table"
        programfile="my_program_file"
        storefiles="my_analytic_store_file";
run;
quit;

```

Run the model.

```
proc accelerator library=mylib;
  runmodel
    modeltablename="model_table"
    inputtablename="my_input_table"
    outputtablename="output_table";
run;
quit;
```

Delete the model.

```
proc accelerator library=mylib;
  deletemodel
    modeltablename="model_table";
run;
quit;
```

Example 4: Publish, Run, and Delete a Model in Teradata

Features:

- LIBNAME statement
- PROC ACCELERATOR statement
- PUBLISHMODEL statement
- RUNMODEL statement
- DELETEMODEL statement

Details

In this example, PROC ACCELERATOR publishes a model to a table in a Teradata database. Then PROC ACCELERATOR runs the model in Teradata by using the SAS Embedded Process.

Note: For Teradata, you do not start the SAS Embedded Process continuous session before running a model.

Program

```
libname mylib teradata
  user="my-user-name"
  password="my-password"
  database="my-database"
  server="my-server";

proc accelerator library=mylib;
```

```

publishModel
    modeltablename="model_table"
    programfile="my_program_file"
    storefiles="my_analytic_store_file";
run;
quit;

proc accelerator library=mylib;
    runmodel
        modeltablename="model_table"
        inputtablename="my_input_table"
        outputtablename="output_table"
        options="loglevel=info;encoding=unicode;";
run;
quit;

proc accelerator library=mylib;
    deletemodel
        modeltablename="model_table";
run;
quit;

```

Program Description

Assign a SAS library, using SAS/ACCESS Interface to Teradata. For more information about this LIBNAME statement, see [“SAS/ACCESS Interface to Teradata” in SAS/ACCESS for Relational Databases: Reference](#).

```

libname mylib teradata
    user="my-user-name"
    password="my-password"
    database="my-database"
    server="my-server";

```

Publish the model. The LIBRARY= option in the PROC ACCELERATOR statement references the libref.

```

proc accelerator library=mylib;
    publishModel
        modeltablename="model_table"
        programfile="my_program_file"
        storefiles="my_analytic_store_file";
run;
quit;

```

Run the model.

```

proc accelerator library=mylib;
    runmodel
        modeltablename="model_table"
        inputtablename="my_input_table"
        outputtablename="output_table"
        options="loglevel=info;encoding=unicode;";
run;
quit;

```

Delete the model.

```
proc accelerator library=mylib;
  deletemodel
    modeltablename="model_table";
run;
quit;
```

Example 5: Use an SQL Query to Read Data for Scoring

Features:

- LIBNAME statement
- PROC ACCELERATOR statement
- PUBLISHMODEL statement
- RUNMODEL statement
- DELETEMODEL statement

Details

In this example, PROC ACCELERATOR publishes a model to a table in a SingleStore database. Then PROC ACCELERATOR runs the model in SingleStore by using the SAS Embedded Process. The INPUTQUERY= option specifies a SELECT statement that uses an SQL query to read the data.

The SingleStore instance that is included in the product SAS SpeedyStore is supported. An external SingleStore database is not supported. For more information, see [“Deployment Details for SingleStore” on page 99](#).

.....

Note: For SingleStore, you do not start the SAS Embedded Process continuous session before running a model.

.....

Program

```
libname mylib sstore
  user=my-user-name
  password="my-password"
  database=my-database
  server=my-server
  port=my-port-number;

proc accelerator library=mylib;
  publishModel
    modeltablename="model_table"
    programfile="my_program_file"
    storefiles="my_analytic_store-file";
```

```

run;
quit;

proc accelerator library=mylib;
  runmodel
    modeltablename="model_table" 1
    inputquery="select sales, salary from mydb.mytable" 2
    outputtablename="output_database.output_table"; 3
run;
quit;

proc accelerator library=mylib;
  deletemodel
    modeltablename="model_table";
run;
quit;

```

Program Description

Assign a SAS library, using SAS/ACCESS Interface to SingleStore. For more information about this LIBNAME statement, see [“SAS/ACCESS Interface to SingleStore” in SAS/ACCESS for Relational Databases: Reference](#).

```

libname mylib sstore
  user=my-user-name
  password="my-password"
  database=my-database
  server=my-server
  port=my-port-number;

```

Publish the model. The LIBRARY= option in the PROC ACCELERATOR statement references the libref.

```

proc accelerator library=mylib;
  publishModel
    modeltablename="model_table"
    programfile="my_program_file"
    storefiles="my_analytic_store-file";
run;
quit;

```

Run the model.

```

proc accelerator library=mylib;
  runmodel
    modeltablename="model_table" 1
    inputquery="select sales, salary from mydb.mytable" 2
    outputtablename="output_database.output_table"; 3
run;
quit;

```

- 1 The MODELTABLENAME= option specifies a one-level name, so it references the Mydb database specified in the Mylib LIBNAME statement.
- 2 The INPUTQUERY= option specifies a SELECT statement. The SELECT query must be supported by SingleStore. You must specify the database in the

SELECT statement, even if it is the same database as in the LIBNAME statement.

- 3 The OUTPUTTABLENAME= option specifies a two-level name, so it stores the table in a different database than the Mylib database from the LIBNAME statement.

Delete the model.

```
proc accelerator library=mylib;
    deletemodel
        modeltablename="model_table";
run;
quit;
```

Example 6: Use Scala Code to Read and Write Data for Scoring

Features:

LIBNAME statement
 PROC ACCELERATOR statement
 PUBLISHMODEL statement
 STARTSESSION statement
 RUNMODEL statement
 STOPSESSION statement
 DELETEMODEL statement

Details

In this Databricks example, the input data is read by using custom Scala code, and the output data is written by using custom Scala code.

IMPORTANT SAS Embedded Process for Spark requires a different deployment for PROC ACCELERATOR than for PROC SCOREACCEL. If you need the functionality of both RPMs, they must be installed on separate clusters. For more information, see [“Deployment Details for Databricks and Synapse” on page 98](#).

Assign a SAS library, using SAS/ACCESS Interface to Spark. Specify PLATFORM=DATABRICKS, connection options, and any other LIBNAME options that you need. For more information about assigning this SAS library, see [“SAS/ACCESS Interface to Spark” in SAS/ACCESS for Relational Databases: Reference](#).

```
libname spklib spark platform=databricks
```



```

user=token
password="authentication-token"
server="myserver.azuredatabricks.net"
catalog="my-catalog-name"
schema="my-schema-name"
port=10016
httppath="my-http-path";

```

Publish the model. Use the LIBRARY= option in the PROC ACCELERATOR statement to reference the libref.

```

proc accelerator library=spklib;
  publishModel
    modeltablename="my-model-table"
    programfile="my-program-file"
    storefiles="my-analytic-store-file";
run;
quit;

```

Start the SAS Embedded Process continuous session. The procedure starts a continuous session and associates the specified endpoint with the SAS library location. The PLATFORMENDPOINT= value is the Databricks workspace URL.

```

proc accelerator library=spklib;
  startsession
    platformendpoint="https://my-identifier.databricks.net";
run;
quit;

```

Run the model. The INPUTSCALACODE= option reads a Spark table. The OUTPUTSCALACODE= option writes a Spark table. Your input Scala string must be a single statement that returns a Spark dataset. Your output Scala string must begin with the function. Here, that function is WRITE.

```

%let
mydatabase=september;
1
proc accelerator library=spklib;
  runmodel
    modeltablename="my-model-table"
    inputscalacode="spark.table("&mydatabase..input-
table").select("&col1", "&col2")" 2
    outputscalacode="write.mode("&overwrite").saveAsTable("&mydatabase..o
utput-table")"; 3
run;

```

- 1 The LET statement defines a macro variable mydatabase, because the user wants to run the model against a different database each month.
- 2 Because the INPUTSCALACODE= option references a macro variable, double quotation marks are required around the entire string. Within the string, quotation marks must be escaped around the value that contains the macro variable. To escape a quotation mark in the string, you must use a pair of quotation marks (") instead of one quotation mark ("). Quotation marks are escaped around the column names also.
- 3 Quotation marks in the OUTPUTSCALACODE= string are escaped similarly to the input string.

Stop the SAS Embedded Process continuous session. When you are finished running models, you can stop the SAS Embedded Process continuous session by submitting PROC ACCELERATOR with the STOPSESSION statement. If you do not stop the session, it terminates a few minutes after you end SAS.

```
proc accelerator library=spklib;  
    stopsession;  
run;  
quit;
```

Delete the model.

```
proc accelerator library=spklib;  
    deletemodel  
        modeltablename="my-model-table";  
run;  
quit;
```

APPEND Procedure

Overview: APPEND Procedure	125
What Does the APPEND Procedure Do?	125
Syntax: APPEND Procedure	126
PROC APPEND Statement	127
ATTRIB Statement	129
FORMAT Statement	130
LABEL Statement	131
WHERE Statement	132
Usage: APPEND Procedure	135
Using the APPEND Procedure	135
Examples: APPEND Procedure	136
Example 1: Concatenating Two SAS Data Sets	136
Example 2: Concatenating a CAS Table to a SAS Data Set	139
Example 3: Getting Sort Indicator Information	144

Overview: APPEND Procedure

What Does the APPEND Procedure Do?

The APPEND procedure adds the observations from one SAS data set to the end of another SAS data set.

For more information, see [Chapter 4, “CAS Processing of Base Procedures,”](#) on page 75.

Generally, the APPEND procedure functions the same as the APPEND statement in the DATASETS procedure. The only difference between the APPEND procedure and the APPEND statement in PROC DATASETS is the default for *libref* in the

BASE= and DATA= options. For PROC APPEND, the default is either Work or User. For the APPEND statement, the default is the libref of the procedure input library.

Syntax: APPEND Procedure

- Requirements:** The BASE= data set must be a member of a SAS library that supports update processing.
- Because the CAS libname engine does not support update processing, an existing CAS table cannot be specified with the BASE= option.
- Notes:** NOTE: _LAST_ is the name of the last data set created in your session. When the BASE= data set does not exist and PROC APPEND creates it, PROC APPEND sets _LAST_ to the name of the BASE= data set.
- If you use the DROP=, KEEP=, or RENAME= options on the BASE= data set, the options affect ONLY the APPEND processing and does not change the variables in the appended BASE= data set. Variables that are dropped or not kept using the DROP= and KEEP= options still exist in the appended BASE= data set. Variables that are renamed using the RENAME= option remain with their original name in the appended BASE= data set.
- Tips:** Complete documentation for the APPEND procedure is located within the DATASETS procedure in [APPEND Statement on page 389](#).
- You can use the ATTRIB, FORMAT, LABEL, and WHERE statements. For more information, see “[Statements with the Same Function in Multiple Procedures](#)” on [page 23](#).
- You can use data set options with the BASE= and DATA= options when using the APPEND procedure.

PROC APPEND BASE=<libref.>

<options> SAS-data-set;

ATTRIB variable-list(s) attribute-list(s);

FORMAT variable-1 <...variable-n> <format> <DEFAULT=default-format>;

LABEL variable-1=label-1<...variable-n=label-n>;

WHERE where-expression-1

< logical-operator where-expression-n>;

Statement	Task	Example
APPEND	Add observations from one SAS data set to the end of another SAS data set	Ex. , Ex. 3

PROC APPEND Statement

Add observations from one SAS data set to the end of another SAS data set.

Syntax

```
APPEND BASE=<libref.>
      SAS-data-set <APPENDVER=V6>
      <DATA=<libref.>SAS-data-set>
      <ENCRYPTKEY=key-value>
      <FORCE>
      <GETSORT>
      <NOWARN>;
```

Required Argument

BASE=<libref.>**SAS-data-set**

names the data set to which you want to add observations.

libref

specifies the library that contains the SAS data set. If you omit the libref, the default is the libref for the procedure input library. If you are using PROC APPEND, the default for libref is either Work or User.

SAS-data-set

names a SAS data set. If the APPEND statement cannot find an existing data set with this name, it creates a new data set in the library. That is, you can use the APPEND statement to create a data set by specifying a new data set name in the BASE= argument.

Whether you are creating a new data set or appending to an existing data set, the BASE= data set is the current SAS data set after all Append operations.

Alias OUT=

Example [“Example 6: Concatenating Two SAS Data Sets” on page 511](#)

Optional Arguments

APPENDVER=V6

uses the Version 6 behavior for appending observations to the BASE= data set, which is to append one observation at a time. Beginning in Version 7, to improve performance, the default behavior changed so that all observations are appended after the data set is processed.

See [“Appending to an Indexed Data Set – Fast-Append Method” on page 396](#)

DATA=<libref.> SAS-data-set

names the SAS data set containing observations that you want to append to the end of the SAS data set specified in the BASE= argument.

libref

specifies the library that contains the SAS data set. If you omit libref, the default is the libref for the procedure input library. The DATA= data set can be from any SAS library. You must use the two-level name if the data set resides in a library other than the procedure input library.

SAS-data-set

names a SAS data set. If the APPEND statement cannot find an existing data set with this name, it stops processing.

Alias NEW=

Default the most recently created SAS data set, from any SAS library

See [“Appending with Generation Groups” on page 399](#)

Example [“Example 6: Concatenating Two SAS Data Sets” on page 511](#)

ENCRYPTKEY=key-value

specifies the key value for AES encryption.

See [“Appending AES-Encrypted Data Sets” on page 395](#)

FORCE

forces the APPEND statement to concatenate data sets when the DATA= data set contains variables that meet one of the following criteria:

- are not in the BASE= data set
- do not have the same type as the variables in the BASE= data set
- are longer than the variables in the BASE= data set

Tip You can use the GENNUM= data set option to append to or from a specific version in a generation group. Here are some examples:

```
/* appends historical version to base A */
proc datasets;
  append base=a
    data=a (gennum=2);

/* appends current version of A to historical version */
proc datasets;
  append base=a (gennum=1)
    data=a;
```

See [“Appending to Data Sets with Different Variables” on page 397](#) and [“Appending to Data Sets That Contain Variables with Different Attributes” on page 398](#)

Example [“Example 6: Concatenating Two SAS Data Sets” on page 511](#)

GETSORT

copies the sort indicator from the DATA= data set to the BASE= data set. The sort indicator is established by either a PROC SORT or an ORDERBY clause in PROC SQL if the following criteria are met:

- The BASE= data set must meet the following criteria:
 - contain no observations
 - accept sort indicators

CAUTION

Any pre-existing sort indicator on the BASE= data set is overwritten with no warning, even if the DATA= data set is not sorted at all.

- The DATA= data set must meet the following criteria:
 - contain a sort indicator established by PROC SORT
 - be the same data representation as the BASE= data set

Restrictions The GETSORT option has no effect on the data sets if the BASE= data set has an audit trail associated with it. This restriction causes a WARNING in the output while the APPEND process continues.

The GETSORT option has no effect on the data sets if there are dropped, kept, or renamed variables in the DATA= data file.

NOWARN

suppresses the warning when used with the FORCE option to concatenate two data sets with different variables.

ATTRIB Statement

Associates a format, informat, label, and length with one or more variables.

Syntax

ATTRIB *variable-list(s) attribute-list(s);*

Arguments

variable-list(s)

names the variables that you want to associate with the attributes.

TIP List the variables in any form that SAS allows.

attribute-list(s)

specifies one or more attributes to assign to *variable-list*. Specify one or more of these attributes in the ATTRIB statement:

FORMAT=format

associates a format with variables in *variable-list*.

Tip The format can be either a standard SAS format or a format that is defined with the FORMAT procedure.

INFORMAT=informat

associates an informat with variables in *variable-list*.

Tip The informat can be either a standard SAS informat or an informat that is defined with the FORMAT procedure.

LABEL='label'

associates a label with variables in *variable-list*.

See Also

[“ATTRIB Statement” in SAS DATA Step Statements: Reference.](#)

FORMAT Statement

Associates formats with variables.

Syntax

FORMAT *variable-1* <...*variable-n*> *format variable-1* <...*variable-n*> *format*;

Arguments

variable

names one or more variables for SAS to associate with a format. You must specify at least one *variable*.

Tip To disassociate a format from a variable, use the variable in a FORMAT statement without specifying a format in a DATA step or in PROC DATASETS. In a DATA step, place this FORMAT statement after the SET statement. See [“Removing a Format” in SAS DATA Step Statements: Reference](#). You can also use PROC DATASETS.

format

specifies the format that is listed for writing the values of the variables.

Tip Formats that are associated with variables by using a FORMAT statement behave like formats that are used with a colon modifier in a subsequent PUT statement. For more information about using a colon modifier, see [“PUT Statement: List” in SAS DATA Step Statements: Reference](#).

See [SAS Formats and Informats: Reference](#)

LABEL Statement

Assigns descriptive labels to variables.

Syntax

LABEL *variable-1* = 'text-string' <...*variable-n* = 'text-string'>;

LABEL *variable-1* = '' <...*variable-n*=''>; | *variable-1* = <...*variable-n*= >;

Arguments

variable

specifies the variable that you want to label. You can specify labels for multiple variables in a single LABEL statement:

```
data test;
  L = 5;
  W = 10;
  label L = 'Length' W = 'Width';
run;
```

text-string

specifies a label of up to 256 bytes.

```
data test;
  L = 5;
  label L = 'Length';
run;
```

Restrictions If the label includes a semicolon (;) or an equal sign (=), you must enclose the label in either single or double quotation marks.

```
label N = 'Number = ';
```

If the label includes single quotation marks ('), you must enclose the label in double quotation marks.

```
label Name = "Owner's Last Name";
```

There is partial support for variable name substitution with the JAVAIMG device. When the label text for the variable exceeds the space allowed, the variable name is used instead by

procedures that are labeling an axis or a legend title. With the exception of the SAS/GRAPH GPLOT procedure, the variable name is not used when the JAVAIMG device is specified.

Notes

Quotation marks are not required unless the label includes a semicolon, quotation marks, or an equal sign.

The LABEL statement expects an equal sign (=) after the first variable. If any other character is found, an error occurs. The LABEL statement considers everything after the equal sign, following the first variable, as part of the label until the statement comes to another variable followed by an equal sign, regardless of quotation marks. In this example, the label for x is **this is x y 4 = this is y** because the next variable that is followed by an equal sign is z.

```
data test;
  x = 1;
  y = 1;
  z = 1;
  label x = 'this is x' y 4 = 'this is y' z = 'This is z';
run;
```

In this LABEL statement, the x variable has the same label:

```
label x = this is x y 4 = this is y z = This is z;
```

See

[“Character Constants and Character Variables” in SAS Programmer’s Guide: Essentials.](#)

”

removes a label from a variable. Enclose a single blank space in quotation marks to remove an existing label. You can remove labels from multiple variables in a single LABEL statement:

```
data test;
  L=5; W=10;
  label L = ' ' W = ' ';
run;
```

Alternatively, you can remove a label by specifying nothing after the equal sign:

```
data test;
  L=5; W=10;
  label L = W = ;
run;
```

WHERE Statement

Subsets the input data set by specifying certain conditions that each observation must meet before it is available for processing.

Syntax

WHERE *where-expression-1*
 < *logical-operator where-expression-n* >;

Arguments

where-expression

is an arithmetic or logical expression that generally consists of a sequence of operands and operators.

Tips The operands and operators described in the next several sections are also valid for the WHERE= data set option.

You can specify multiple where-expressions.

logical-operator

can be AND, AND NOT, OR, or OR NOT.

Example

Procedures That Support the WHERE Statement

You can use the WHERE statement with any of the following Base SAS procedures that read a SAS data set:

COMPARE	REPORT
CORR	SORT
DATASETS (APPEND statement)	SQL
FREQ	STANDARD
MEANS or SUMMARY	TABULATE
PRINT	TRANSPOSE
RANK	UNIVARIATE

Details

- The COMPARE procedure and the APPEND statement in PROC DATASETS accept more than one input data set. For more information, see the documentation for the specific procedure.
- To subset the output data set, use the WHERE= data set option:

```
proc report data=debate nowd  
            out=onlyfr(where=(year='1'));  
run;
```

For more information about WHERE=, see [SAS DATA Step Statements: Reference](#).

Temporarily Dissociating a Format from a Variable

In this example, PROC PRINT prints only those observations that meet the condition of the WHERE expression.

```
options nodate pageno=1 linesize=64  
        pagesize=40;  
  
proc print data=debate noobs;  
  where gpa>3.5;  
  title 'Team Members with a GPA';  
  title2 'Greater than 3.5';  
run;
```

Output

**Team Members with a GPA
Greater than 3.5**

Name	Gender	Year	GPA
Capiccio	m	Freshman	3.598
Tucker	m	Freshman	3.901
Bagwell	f	Sophomore	3.722
Gold	f	Junior	3.609
Syme	f	Junior	3.883
Baglione	f	Senior	4.000
Carr	m	Senior	3.750
Hall	m	Senior	3.574

Usage: APPEND Procedure

Using the APPEND Procedure

To copy only the table metadata and structure of a data set but not the data, use the following example where Dataset1 is nonexistent:

```
proc append base=dataset1 data=dataset2(obs=0);  
run;
```

```
proc contents data=dataset1;  
run;
```

Examples: APPEND Procedure

Example 1: Concatenating Two SAS Data Sets

Features:	PROC APPEND statement options BASE= DATA= FORCE OPTIONS statement PRINT procedure
Data set:	EXP Library

Details

This example demonstrates the following tasks:

- suppresses the printing of a library
- appends two data sets
- prints the new data set after appending

To create the Exp.Results and Exp.Sur data sets and print them out before using this example to concatenate them, see [“EXP Library” on page 2513](#).

Program

```
options pagesize=40 linesize=64 nodate pageno=1;
LIBNAME exp 'SAS-library';

proc append base=exp.results data=exp.sur force;
run;

proc print data=exp.results noobs;
  title 'The Concatenated RESULTS Data Set';
run;
```

Program Description

This example appends one data set to the end of another data set.

The data set Exp.Sur contains the variable Wt6Mos, but the Exp.Results data set does not.

Set the system options. The NODATE option suppresses the display of the date and time in the output. The PAGENO= option specifies the starting page number. The LINESIZE= option specifies the output line length, and the PAGESIZE= option specifies the number of lines on an output page.

```
options pagesize=40 linesize=64 nodate pageno=1;
```

The LIBNAME statement assigns the library.

```
LIBNAME exp 'SAS-library';
```

Append the data set Exp.Sur to the Exp.Results data set. PROC APPEND appends the data set Exp.Sur to the data set Exp.Results. FORCE causes PROC APPEND to carry out the Append operation even though Exp.Sur has a variable that Exp.Results does not. PROC APPEND does not add the Wt6Mos variable to Exp.Results.

```
proc append base=exp.results data=exp.sur force;  
run;
```

Print the data set. See [Output 8.20 on page 139](#).

```
proc print data=exp.results noobs;  
  title 'The Concatenated RESULTS Data Set';  
run;
```

Output: Concatenating Two Data Sets

Output 8.1 *The Results Data Set*

The RESULTS Data Set				
ID	TREAT	INITWT	WT3MOS	AGE
1	Other	166.28	146.98	35
2	Other	214.42	210.22	54
3	Other	172.46	159.42	33
5	Other	175.41	160.66	37
6	Other	173.13	169.40	20
7	Other	181.25	170.94	30
10	Other	239.83	214.48	48
11	Other	175.32	162.66	51
12	Other	227.01	211.06	29
13	Other	274.82	251.82	31

Output 8.2 *The Sur Data Set*

The EXP.SUR Data Set					
ID	treat	initwt	wt3mos	wt6mos	age
14	surgery	203.60	169.78	143.88	38
17	surgery	171.52	150.33	123.18	42
18	surgery	207.46	155.22	.	41

Output 8.3 Concatenating the Results and the Sur Data Sets

The RESULTS Data Set				
ID	TREAT	INITWT	WT3MOS	AGE
1	Other	166.28	146.98	35
2	Other	214.42	210.22	54
3	Other	172.46	159.42	33
5	Other	175.41	160.66	37
6	Other	173.13	169.40	20
7	Other	181.25	170.94	30
10	Other	239.83	214.48	48
11	Other	175.32	162.66	51
12	Other	227.01	211.06	29
13	Other	274.82	251.82	31
14	surgery	203.60	169.78	38
17	surgery	171.52	150.33	42
18	surgery	207.46	155.22	41

Example 2: Concatenating a CAS Table to a SAS Data Set

Features:

- PROC APPEND statement options
 - BASE=
 - DATA=
 - FORCE
- OPTIONS statement
- CONTENTS procedure

Details

This example demonstrates the following tasks:

- appending a CAS table to a SAS data set
- contents of the table, the data set, and the new data set after appending

Program

```
options pagesize=40 linesize=64 nodate pageno=1;

libname sasCas1 cas;

libname saleslib 'directory-name';

proc contents data=saleslib.monthly;
run;

proc contents data=sasCas1.lastmonth;
run;

proc append base=saleslib.monthly data=sasCas1.lastmonth force;
run;

proc sql outobs=5;
  select store_id, address, city, state, zipcode, totalsales
  format dollar12.
  from saleslib.monthly(obs=4)
  where totalsales gt 2000000;
quit;
```

Program Description

This example appends a CAS table to the end of a SAS data set.

Set the system options. The NODATE option suppresses the display of the date and time in the output. The PAGENO= option specifies the starting page number. The LINESIZE= option specifies the output line length, and the PAGESIZE= option specifies the number of lines on an output page.

```
options pagesize=40 linesize=64 nodate pageno=1;
```

The LIBNAME statements assign the CAS engine and BASE engine libraries.

```
libname sasCas1 cas;

libname saleslib 'directory-name';
```

Check the contents of the table and data set. Use PROC CONTENTS to view the data set and table.

```
proc contents data=saleslib.monthly;
run;

proc contents data=sasCas1.lastmonth;
run;
```

Append the SasCas1.LastMonth table to the SalesLib.Monthly data set. The data for last month's sales in a CAS table is appended to the accumulated sales data stored in a SAS data set. The CAS table uses VARCHAR to store the city and address values. The SAS data set stores the values in character variables. Since the attribute for the two values differ, the FORCE option is used in PROC APPEND.

```
proc append base=saleslib.monthly data=sasCas1.lastmonth force;
run;
```

Retrieve total sales. Use PROC SQL to retrieve five variables and sales that are greater than \$2,000,000.

```
proc sql outobs=5;
  select store_id, address, city, state, zipcode, totalsales
  format dollar12.
  from saleslib.monthly(obs=4)
  where totalsales gt 2000000;
quit;
```

Warnings in the Log

Note the warnings that were sent to the log.

```
117      proc append base=saleslib.monthly data=sascas1.lastmonth force;
118      run;

NOTE: Appending SASCAS1.LASTMONTH to SALES LIB.MONTHLY.
WARNING: Variable city has different lengths on BASE and DATA files
        (BASE 100 DATA 21).
WARNING: Variable address has different lengths on BASE and DATA files
        (BASE 160 DATA 19).
NOTE: There were 12 observations read from the data set SASCAS1.LASTMONTH.
NOTE: 12 observations added.
NOTE: The data set SALES LIB.MONTHLY has 24 observations and 7 variables.
```

Output: Concatenating a CAS Table to a SAS Data Set

Output 8.4 The CAS Table Contents

LastMonth Table					
The CONTENTS Procedure					
Data Set Name	SASCAS1.LASTMONTH			Observations	12
Member Type	DATA			Variables	7
Engine	CAS			Indexes	0
Created	08/16/2017 18:56:09			Observation Length	80
Last Modified	08/16/2017 18:56:09			Deleted Observations	0
Protection				Compressed	NO
Data Set Type				Sorted	NO
Label					
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64				
Encoding	utf-8 Unicode (UTF-8)				

Engine/Host Dependent Information	
Data Limit	100MB
Caslib	CASUSER
Scope	Session

Alphabetic List of Variables and Attributes					
#	Variable	Type	Len		Max Bytes Used
			Bytes	Chars	
6	address	Varchar	160	40	19
4	city	Varchar	100	25	21
7	month	Char	10		
5	state	Char	2		
1	store_id	Char	3		
2	totalsales	Num	8		
3	zipcode	Char	5		

Output 8.5 The Monthly Data Set Contents

Concatenated Data Set			
The CONTENTS Procedure			
Data Set Name	SALESLIB.MONTHLY	Observations	24
Member Type	DATA	Variables	7
Engine	V9	Indexes	0
Created	08/16/2017 13:56:07	Observation Length	288
Last Modified	08/16/2017 13:56:17	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64		
Encoding	utf-8 Unicode (UTF-8)		

Engine/Host Dependent Information	
Data Set Page Size	65536
Number of Data Set Pages	1
First Data Page	1
Max Obs per Page	227
Obs in First Data Page	24
Number of Data Set Repairs	0
Filename	/r/ge
Release Created	9.0401M5
Host Created	Linux
Inode Number	71586336
Access Permission	nw-r--r--
Owner Name	
File Size	128KB
File Size (bytes)	131072

Alphabetic List of Variables and Attributes			
#	Variable	Type	Len
6	address	Char	160
4	city	Char	100
7	month	Char	10
5	state	Char	2
1	store_id	Char	3
2	totalsales	Num	8
3	zipcode	Char	5

Output 8.6 Concatenated Data Set

Concatenated Data Set			
The CONTENTS Procedure			
Data Set Name	SALES.LIB.MONTHLY	Observations	24
Member Type	DATA	Variables	7
Engine	V9	Indexes	0
Created	08/16/2017 13:56:07	Observation Length	288
Last Modified	08/16/2017 13:56:17	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64		
Encoding	utf-8 Unicode (UTF-8)		

Engine/Host Dependent Information	
Data Set Page Size	65536
Number of Data Set Pages	1
First Data Page	1
Max Obs per Page	227
Obs in First Data Page	24
Number of Data Set Repairs	0
Filename	/r/ge
Release Created	9.0401M5
Host Created	Linux
Inode Number	71586336
Access Permission	rw-r--r--
Owner Name	
File Size	128KB
File Size (bytes)	131072

Output 8.7 Total Sales

store_id	address	city	state	zipcode	totalsales
843	318 S Barnes St	What Cheer	IA	50268	\$2,218,497
486	2345 Lindeman Ln	Nameless	TX	78641	\$2,101,480
814	115 Tuscarora Pike	Shanghai	WV	25427	\$2,195,378
842	318 S Lamar St	What Cheer	IA	50268	\$2,218,497

Example 3: Getting Sort Indicator Information

Features:

- PROC APPEND statement options
- GETSORT
- BY statement
- CONTENTS procedure
- ODS statement
- SORT procedure

Details

This example demonstrates the following tasks:

- creates two data sets: one with no observations and one with observations
- sorts a data set in descending order
- creates a sort indicator using the SORTEDBY data set option
- creates a sort indicator using the SORT procedure

Program

```
data mtea;
    length var1 8.;
    stop;
run;

data phull;
    length var1 8.;
    do var1=1 to 100000;
        output;
    end;
run;

proc sort data=phull;
    by DESCENDING var1;
run;

proc append base=mtea data=phull getsort;
run;

ods select sortedby;

proc contents data=mtea;
run;

data mysort(sortedby=var1);
    length var1 8.;
    do var1=1 to 10;
        output;
    end;
run;

ods select sortedby;

proc contents data=mysort;
run;

data mysort;
    length var1 8.;
    do var1=1 to 10;
```

```

        output;
    end;
run;

proc sort data=mysort;
    by var1;
run;

ods select sortedby;

proc contents data=mysort;
run;

```

Program Description

The following example shows that a sort indicator can be inherited using the GETSORT option with the APPEND procedure.

Create a "shell" data set that contains no observations.

```

data mtea;
    length var1 8.;
    stop;
run;

```

Create another data set with the same structure, but with many observations.

Sort the data set.

```

data phull;
    length var1 8.;
    do var1=1 to 100000;
        output;
    end;
run;

proc sort data=phull;
    by DESCENDING var1;
run;

proc append base=mtea data=phull getsort;
run;

ods select sortedby;

proc contents data=mtea;
run;

```

A sort indicator is being created using the SORTEDBY data set option.

```

data mysort(sortedby=var1);
    length var1 8.;
    do var1=1 to 10;
        output;
    end;
run;

```



```
ods select sortedby;

proc contents data=mysort;
run;
```

A sort indicator is being created by PROC SORT.

```
data mysort;
    length var1 8.;
    do var1=1 to 10;
        output;
    end;
run;

proc sort data=mysort;
    by var1;
run;

ods select sortedby;

proc contents data=mysort;
run;
```

Output Examples

Output 8.8 *Descending Sort Information*

The CONTENTS Procedure	
Sort Information	
Sortedby	DESCENDING var1
Validated	YES
Character Set	ANSI

Output 8.9 *Sort Indicator Information Using the SORTEDBY= Data Set Option*

The CONTENTS Procedure	
Sort Information	
Sortedby	var1
Validated	NO
Character Set	ANSI

Output 8.10 Sort Indicator Information Using the SORT Procedure**The CONTENTS Procedure**

Sort Information	
Sortedby	var1
Validated	YES
Character Set	ANSI

CATALOG Procedure

Overview: CATALOG Procedure	149
What Does the CATALOG Procedure Do?	150
Concepts: CATALOG Procedure	150
Concepts: CATALOG Procedure	150
Syntax: CATALOG Procedure	150
PROC CATALOG Statement	151
CHANGE Statement	154
CONTENTS Statement	154
COPY Statement	156
DELETE Statement	158
EXCHANGE Statement	158
EXCLUDE Statement	159
MODIFY Statement	160
SAVE Statement	161
SELECT Statement	162
QUIT Statement	163
Usage: CATALOG Procedure	163
Interactive Processing with RUN Groups	163
Specifying an Entry Type	164
Catalog Concatenation	167
Results: CATALOG Procedure	168
Results: CATALOG Procedure	168
Examples: CATALOG Procedure	169
Example 1: Copying, Deleting, and Moving Catalog Entries from Multiple Catalogs	169
Example 2: Displaying Contents, Changing Names, and Changing a Description ..	173
Example 3: Using the FORCE Option with the KILL Option	175

Overview: CATALOG Procedure

What Does the CATALOG Procedure Do?

The CATALOG procedure manages entries in SAS catalogs. PROC CATALOG is an interactive, statement-driven procedure that enables you to do the following:

- create a listing of the contents of a catalog
- copy a catalog or selected entries within a catalog
- rename, exchange, or delete entries within a catalog
- change the name of a catalog entry
- modify, by changing or deleting, the description of a catalog entry

Concepts: CATALOG Procedure

Concepts: CATALOG Procedure

To learn how to use the SAS windowing environment to manage entries in a SAS catalog, see the SAS online Help for the Explorer window. You might prefer to use the Explorer window instead of using PROC CATALOG. You can do most of what the procedure does using the Explorer window.

Syntax: CATALOG Procedure

Restriction: Use in the Compute Server, but not in CAS.

Tip: The CATALOG procedure supports RUN-group processing.

```

PROC CATALOG CATALOG=<libref.>catalog <ENTRYTYPE=entry-type>
<FORCE> <KILL>;

CONTENTS <OUT=SAS-data-set> <FILE=fileref>;

COPY OUT=<libref.>catalog <options>;
    SELECT entry-1 <entry-2 ...> </ ENTRYTYPE=entry-type >;
    EXCLUDE entry-1 <entry-2 ...> </ ENTRYTYPE=entry-type>;

CHANGE old-name-1=new-name-1
    <old-name-2=new-name-2 ...>
    </ ENTRYTYPE=entry-type>;

EXCHANGE name-1=other-name-1
    <name-2=other-name-2 ...>
    </ ENTRYTYPE=entry-type>;

DELETE entry-1 <entry-2 ...> </ ENTRYTYPE=entry-type>;

MODIFY entry (DESCRIPTION=<<'>entry-description<'>>)
    </ ENTRYTYPE=entry-type>;

SAVE entry-1 <entry-2 ...> </ ENTRYTYPE=entry-type >;

```

Statement	Task	Example
Chapter 9, "CATALOG Procedure,"	Copy entries from one SAS catalog to another	Ex. 1, Ex. 2, Ex. 3
CHANGE	Change the names of catalog entries	Ex. 2
CONTENTS	Print the contents of a catalog	Ex. 2
COPY	Copy some or all of the entries in one catalog to another catalog	Ex. 1
DELETE	Delete specified entries	Ex. 1
EXCHANGE	Switch the names of two catalog entries	
EXCLUDE	Exclude entries from being copied	Ex. 1
MODIFY	Change the description of a catalog entry	Ex. 2
SAVE	Delete all except the entries specified	
SELECT	Copy only selected entries	Ex. 1

PROC CATALOG Statement

Copies entries from one SAS catalog to another.

Syntax

```
PROC CATALOG CATALOG=<libref.>catalog <ENTRYTYPE=entry-type>
<FORCE> <KILL>;
```

Summary of Optional Arguments

ENTRYTYPE=entry-type

restricts processing of the current PROC CATALOG step to one entry type.

FORCE

forces statements to execute on a catalog that is opened by another resource environment.

KILL

Required Argument

CATALOG=<libref.>catalog

specifies the SAS catalog to process.

Aliases CAT=

C=

Default PROC CATALOG processes all entries in the catalog.

Example [“Example 3: Using the FORCE Option with the KILL Option” on page 175](#)

Optional Arguments

ENTRYTYPE=entry-type

restricts processing of the current PROC CATALOG step to one entry type.

Alias ET=

Default PROC CATALOG processes all entries in a catalog.

Interactions The specified entry type applies to any one-level entry names that are used in a subordinate statement. You cannot override this specification in a subordinate statement.

The ENTRYTYPE= option does not restrict the effects of the KILL option.

Tip In order to process multiple entry types in a single PROC CATALOG step, use the ENTRYTYPE= option in a subordinate statement, not in the PROC CATALOG statement.

See [“Specifying an Entry Type” on page 164](#)

- Examples [“Example 1: Copying, Deleting, and Moving Catalog Entries from Multiple Catalogs” on page 169](#)
- [“Example 2: Displaying Contents, Changing Names, and Changing a Description” on page 173](#)

FORCE

forces statements to execute on a catalog that is opened by another resource environment.

Some CATALOG statements and arguments require exclusive access to the catalog that they operate on if the statement can radically change the contents of a catalog. If exclusive access cannot be obtained, then the action fails. Here are the statements and arguments and the catalogs that are affected by the FORCE option:

- COPY affects the OUT= catalog.
- COPY MOVE affects the IN= catalog and the OUT= catalog.
- SAVE affects the specified catalog.

Tip Use the FORCE option to execute the statement, even if exclusive access cannot be obtained.

Example [“Example 3: Using the FORCE Option with the KILL Option” on page 175](#)

KILL

affects the specified catalog and deletes all entries in a SAS catalog.

deletes all entries in a SAS catalog.

CAUTION

Do not attempt to limit the effects of the KILL option. This option deletes all entries in a SAS catalog before any option or other statement takes effect.

Interactions The KILL option deletes all catalog entries even when the ENTRYTYPE= option is specified.

The SAVE statement has no effect because the KILL option deletes all entries in a SAS catalog before any other statements are processed.

Tip The KILL option deletes all entries but does not remove an empty catalog from the SAS library. You must use another method, such as PROC DATASETS or the DIR window to delete an empty SAS catalog.

Example [“Example 3: Using the FORCE Option with the KILL Option” on page 175](#)

CHANGE Statement

Renames one or more catalog entry names.

Tip: You can change multiple entry names in a single CHANGE statement or use multiple CHANGE statements.

Example: [“Example 2: Displaying Contents, Changing Names, and Changing a Description” on page 173](#)

Syntax

```
CHANGE old-name-1=new-name-1  
      <old-name-2=new-name-2 ...>  
      </ ENTRYTYPE=entry-type>;
```

Required Argument

old-name=new-name

specifies the current name of a catalog entry and the new name that you want to assign to it. Specify any valid SAS name.

Restriction You must designate the type of the entry, either with the name (*entry-name.entry-type*) or with the ENTRYTYPE= option.

Optional Argument

ENTRYTYPE=*entry-type*

restricts processing to one entry type.

Alias ET=

See [“The ENTRYTYPE= Option” on page 166](#)

[“Specifying an Entry Type” on page 164](#)

CONTENTS Statement

Lists the contents of a catalog in the procedure output or writes a list of the contents to a SAS data set, an external file, or both.

Note: The ENTRYTYPE= option is not available for the CONTENTS statement.

Example: [“Example 2: Displaying Contents, Changing Names, and Changing a Description” on page 173](#)

Syntax

CONTENTS <CATALOG=<libref.>catalog > <OUT=SAS-data-set> <FILE=fileref>;

Without Arguments

The output is sent to the procedure output.

Optional Arguments

CATALOG=<libref.>catalog

specifies the SAS catalog to process.

Aliases CAT=

C=

Default None

FILE=fileref

sends the contents to an external file that is identified with a SAS fileref.

Interaction If *fileref* has not been previously assigned to a file, then the file is created and named according to operating environment-dependent rules for external files.

OUT=SAS-data-set

sends the contents to a SAS data set. When the statement executes, a message in the SAS log reports that a data set has been created. The data set contains six variables in the following order:

Table 9.1 OUT= Output

LIBNAME	the libref
MEMNAME	the catalog name
NAME	the names of entries
TYPE	the types of entries
DESC	the descriptions of entries
DATE	the dates entries were last modified

COPY Statement

Copies some or all of the entries in one catalog to another catalog.

- Restriction:** A COPY statement's effect ends at a RUN statement or at the beginning of a statement other than the SELECT or EXCLUDE statement.
- Tips:** Use the SELECT or EXCLUDE statement, but not both, after the COPY statement to limit which entries are copied.
- You can copy entries from multiple catalogs in a single PROC step, not just the one specified in the PROC CATALOG statement.
- The ENTRYTYPE= option does not require a forward slash (/) in the COPY statement.
- Example:** ["Example 1: Copying, Deleting, and Moving Catalog Entries from Multiple Catalogs" on page 169](#)

Syntax

COPY *OUT*=<*libref.*>*catalog* <*options*>;

Required Argument

OUT=<*libref.*>*catalog*
names the catalog to which entries are copied.

Optional Arguments

ENTRYTYPE=*entry-type*
restricts processing to one entry type for the current COPY statement and any subsequent SELECT or EXCLUDE statements.

Alias ET=

See ["The ENTRYTYPE= Option" on page 166](#)

["Specifying an Entry Type" on page 164](#)

IN=<*libref.*>*catalog*
specifies the catalog to copy.

Interaction The IN= option overrides a CATALOG= argument that was specified in the PROC CATALOG statement.

Example ["Example 1: Copying, Deleting, and Moving Catalog Entries from Multiple Catalogs" on page 169](#)

LOCKCAT=EXCLUSIVE | SHARE

specifies whether to enable more than one user to copy to the same catalog at the same time. Using LOCKCAT=SHARE locks individual entries rather than the entire catalog, which enables greater throughput. The default is LOCKCAT=EXCLUSIVE, which locks the entire catalog to one user. Note that using the LOCKCAT=SHARE option can lessen performance if used in a single-user environment because of the overhead associated with locking and unlocking each entry.

MOVE

deletes the original catalog or entries after the new copy is made.

Interaction When the MOVE option removes all entries from a catalog, the procedure deletes the catalog from the library.

NEW

overwrites the catalog (specified by the OUT= option) if it already exists. If you omit the NEW option, then PROC CATALOG updates the catalog.

See For information about using the NEW option with concatenated catalogs, see [“Catalog Concatenation” on page 167](#).

NOEDIT

prevents the copied version of the following SAS/AF entry types from being edited by the BUILD procedure:

CBT
FRAME
HELP
MENU
PROGRAM
SCL
SYSTEM

Restriction If you specify the NOEDIT option for an entry that is not one of the above types, then it is ignored.

Tip When creating SAS/AF applications for other users, use the NOEDIT option to protect the application by preventing certain catalog entries from being altered.

Example [“Example 1: Copying, Deleting, and Moving Catalog Entries from Multiple Catalogs” on page 169](#)

NOSOURCE

omits copying the source lines when you copy a SAS/AF PROGRAM, FRAME, or SCL entry.

Alias NOSRC

Restriction If you specify this option for an entry other than a PROGRAM, FRAME, or SCL entry, then it is ignored.

DELETE Statement

Deletes entries from a SAS catalog.

- Restriction:** This procedure is not supported on the CAS server.
- Tips:** Use the DELETE statement to delete only a few entries; use the SAVE option when it is more convenient to specify which entries not to delete.
You can specify multiple entries. You can also use multiple DELETE statements.
- See:** [SAVE Statement on page 161](#)
- Example:** [“Example 1: Copying, Deleting, and Moving Catalog Entries from Multiple Catalogs” on page 169](#)

Syntax

```
DELETE entry-1 <entry-2 ...> </ ENTRYTYPE=entry-type>;
```

Required Argument

***entry-1* <*entry-2 ...*>**
specifies the name of one or more SAS catalog entries.

Restriction You must designate the type of the entry, either with the name (*entry-name.entry-type*) or with the ENTRYTYPE= option.

Optional Argument

ENTRYTYPE=*entry-type*
restricts processing to one entry type.

Alias ET=

See [“The ENTRYTYPE= Option” on page 166](#)

[“Specifying an Entry Type” on page 164](#)

EXCHANGE Statement

Switches the name of two catalog entries.

- Restriction:** When using the EXCHANGE statement, the catalog entries must be of the same type.

Syntax

```
EXCHANGE name-1=other-name-1
<name-2=other-name-2 ...>
</ ENTRYTYPE=entry-type>;
```

Required Argument

name=other-name

specifies two catalog entry names that the procedure switches.

Interaction You can specify only the entry name without the entry type if you use the ENTRYTYPE= option on either the PROC CATALOG statement or the EXCHANGE statement.

See [“Specifying an Entry Type” on page 164](#)

Optional Argument

ENTRYTYPE=entry-type

restricts processing to one entry type.

Alias ET=

See [“The ENTRYTYPE= Option” on page 166](#)

[“Specifying an Entry Type” on page 164](#)

EXCLUDE Statement

Specifies entries that the COPY statement does not copy.

Restrictions: The EXCLUDE statement requires the COPY statement.
Do not use the EXCLUDE statement with the SELECT statement.

Tips: You can specify multiple entries in a single EXCLUDE statement.
You can use multiple EXCLUDE statements with a single COPY statement within a RUN group.

See: [COPY Statement on page 156](#) and [SELECT Statement on page 162](#)

Example: [“Example 1: Copying, Deleting, and Moving Catalog Entries from Multiple Catalogs” on page 169](#)

Syntax

```
EXCLUDE entry-1 <entry-2 ...> </ ENTRYTYPE=entry-type>;
```

Required Argument

entry-1 <entry-2 ...>

specifies the name of one or more SAS catalog entries.

Restriction You must designate the type of the entry, either when you specify the name (*entry-name.entry-type*) or with the ENTRYTYPE= option.

See [“Specifying an Entry Type” on page 164](#)

Optional Argument

ENTRYTYPE=entry-type

restricts processing to one entry type.

Alias ET=

See [“The ENTRYTYPE= Option” on page 166](#)

[“Specifying an Entry Type” on page 164](#)

MODIFY Statement

Changes the description of a catalog entry.

Example: [“Example 2: Displaying Contents, Changing Names, and Changing a Description” on page 173](#)

Syntax

```
MODIFY entry (DESCRIPTION=<<'>entry-description<'>>
  </ ENTRYTYPE=entry-type>;
```

Required Arguments

entry

specifies the name of one SAS catalog entry. You can specify the entry type with the name (*entry-name.entry-type*).

Restriction You must designate the type of the entry, either when you specify the name (*entry-name.entry-type*) or with the ENTRYTYPE= option.

See [“Specifying an Entry Type” on page 164](#)

DESCRIPTION='entry-description'

changes the description of a catalog entry by replacing it with a new description, up to 256 characters long, or by removing it altogether. You can enclose the description in single or double quotation marks.

Alias DESC

Tip When using the MODIFY statement with the CATALOG procedure, use the DESCRIPTION= option with no text to remove the current description.

Optional Argument

ENTRYTYPE=entry-type

restricts processing to one entry type.

Alias ET=

See [“The ENTRYTYPE= Option” on page 166](#)

[“Specifying an Entry Type” on page 164](#)

SAVE Statement

Specifies entries not to delete from a SAS catalog.

Restriction: The SAVE statement cannot limit the effects of the KILL option.

Tips: Use the SAVE statement to delete all but a few entries in a catalog. Use the DELETE statement when it is more convenient to specify which entries to delete. You can specify multiple entries and use multiple SAVE statements.

See: [DELETE Statement on page 158](#)

Syntax

SAVE *entry-1* *<entry-2 ...>* *</ ENTRYTYPE=entry-type >*;

Required Argument

entry-1 <entry-2...>

specifies the name of one or more SAS catalog entries.

Restriction You must designate the type of the entry, either with the name (*entry-name.entry-type*) or with the ENTRYTYPE= option when using the SAVE statement.

Optional Argument

ENTRYTYPE=entry-type

restricts processing to one entry type.

Alias ET=

See [“The ENTRYTYPE= Option” on page 166](#)

[“Specifying an Entry Type” on page 164](#)

SELECT Statement

Specifies entries that the COPY statement copies.

Restrictions: The SELECT statement requires the COPY statement.

The SELECT statement cannot be used with an EXCLUDE statement.

Tips: You can specify multiple entries in a single SELECT statement.

You can use multiple SELECT statements with a single COPY statement within a RUN group.

See: [COPY Statement on page 156](#) and [EXCLUDE Statement on page 159](#)

Example: [“Example 1: Copying, Deleting, and Moving Catalog Entries from Multiple Catalogs” on page 169](#)

Syntax

```
SELECT entry-1 <entry-2 ...> </ ENTRYTYPE=entry-type >;
```

Required Argument

***entry-1* <*entry-2 ...*>**

specifies the name of one or more SAS catalog entries.

Restriction You must designate the type of the entry, either when you specify the name (*entry-name.entry-type*) or with the ENTRYTYPE= option.

Optional Argument

ENTRYTYPE=*entry-type*

restricts processing to one entry type.

Alias ET=

See [“The ENTRYTYPE= Option” on page 166](#).

[“Specifying an Entry Type” on page 164](#).

QUIT Statement

Executes any statements that have not executed and ends the procedure.

Syntax

QUIT;

Usage: CATALOG Procedure

Interactive Processing with RUN Groups

Definition

The CATALOG procedure is interactive. Once you submit a PROC CATALOG statement, you can continue to submit and execute statements or groups of statements without repeating the PROC CATALOG statement.

A set of procedure statements ending with a RUN statement is called a *RUN group*. The changes specified in a given group of statements take effect when a RUN statement is encountered.

How to End a PROC CATALOG Step

In the DATA step and most SAS procedures, a RUN statement is a step boundary and ends the step. However, a simple RUN statement does not end an interactive procedure. The following list contains ways to terminate a PROC CATALOG step:

- submit a QUIT statement
- submit a RUN statement with the CANCEL option
- submit another DATA or PROC statement
- end your session

Note: When you enter a QUIT, DATA, or PROC statement, any statements following the last RUN group execute before the CATALOG procedure terminates. If you enter a RUN statement with the CANCEL option, then the remaining statements *do not execute* before the procedure ends.

See “[Example 2: Displaying Contents, Changing Names, and Changing a Description](#)” on page 173.

Error Handling and RUN Groups

Error handling is based in part on the division of statements into RUN groups. If a syntax error is encountered, then none of the statements in the current RUN group execute, and execution proceeds to the next RUN group.

For example, the following statements contain a misspelled DELETE statement:

```
proc catalog catalog=misc entrytype=help;
  copy out=drink;
  select coffee tea;
  del juices;          /* INCORRECT!!! */
  exchange glass=plastic;
run;
  change calstats=nutri;
run;
quit;
```

Because the DELETE statement is incorrectly specified as DEL, no statements in that RUN group execute, except the PROC CATALOG statement itself. The CHANGE statement does execute because it is in a different RUN group.

Note: Be careful when setting up batch jobs in which one RUN group’s statements depend on the effects of a previous RUN group, especially when deleting and renaming entries.

Specifying an Entry Type

Four Ways to Supply an Entry Type

There is no default entry type, so if you do not supply one, then PROC CATALOG generates an error. You can supply an entry type in one of four ways, as shown in the following table:

Table 9.2 Supplying an Entry Type

Entry Type	Example
Entry name	<code>delete</code> <code>test1.program</code> <code>test1.log test2.log;</code>
ET= in parentheses	<code>delete</code> <code>test1 (et=program);</code>
ET= after a slash ¹	<code>delete test1 (et=program)</code> <code>test1 test2 / et=log;</code>
ENTRYTYPE= without a slash ²	<code>proc catalog catalog=mycat et=log;</code> <code>delete test1 test2;</code>

¹ in a subordinate statement
² in the PROC CATALOG or the COPY statement

Note: All statements, except the CONTENTS statement, accept the ENTRYTYPE= option.

Advantages of Using the ENTRYTYPE= Option

The ENTRYTYPE= option can save keystrokes when you are processing multiple entries of the same type.

To create a default for entry type for all statements in the current step, use the ENTRYTYPE= option in the PROC CATALOG statement. To set the default for only the current statement, use the ENTRYTYPE= option in a subordinate statement.

You can have many entries of one type and a few of other types. You can use the ENTRYTYPE= option to specify a default and then override that for individual entries with (ENTRYTYPE=) *in parentheses* after those entries.

Avoid a Common Error

You cannot specify the ENTRYTYPE= option in both the PROC CATALOG statement and a subordinate statement. For example, these statements generate an error and do not delete any entries because the ENTRYTYPE= option specifications contradict each other:

```
/* THIS IS INCORRECT CODE. */
proc catalog cat=sample et=help;
  delete a b c / et=program;
run;
quit;
```

The ENTRYTYPE= Option

The ENTRYTYPE= option is available in every statement in the CATALOG procedure except the CONTENTS statement.

ENTRYTYPE=*entry-type*

not in parentheses, sets a default entry type for the entire PROC step when used in the PROC CATALOG statement. In all other statements, this option sets a default entry type for the current statement. If you omit the ENTRYTYPE= option, then PROC CATALOG processes all entries in the catalog.

Alias	ET=
Default	PROC CATALOG processes all entries in the catalog.
Interactions	If you specify the ENTRYTYPE= option in the PROC CATALOG statement, then you do not specify either ENTRYTYPE= or (ENTRYTYPE=) in a subordinate statement. (ENTRYTYPE=) in parentheses immediately following an entry name overrides the ENTRYTYPE= option in that same statement.
Tips	On all statements except the PROC CATALOG and COPY statements, the ENTRYTYPE= option follows a slash. To process multiple entry types in a single PROC CATALOG step, use the ENTRYTYPE= option in a subordinate statement, not in the PROC CATALOG statement.
See	“Specifying an Entry Type” on page 164 and “Example 1: Copying, Deleting, and Moving Catalog Entries from Multiple Catalogs” on page 169

(ENTRYTYPE=*entry-type*)

in parentheses, identifies the type of the entry just preceding it.

Alias	(ET=)
Restriction	An (ENTRYTYPE=) option immediately following an entry name in a subordinate statement cannot override an ENTRYTYPE= option in the PROC CATALOG statement. It generates a syntax error.
Interaction	The (ENTRYTYPE=) option immediately following an entry name overrides the ENTRYTYPE= option in that same statement.
Tips	This form is useful mainly for specifying exceptions to an ENTRYTYPE= option that is used in a subordinate statement. The following statement deletes A.Help, B.Format, and C.Help:

```
delete a b (et=format) c / et=help;
```

For the CHANGE and EXCHANGE statements, specify the (ENTRYTYPE=) option in parentheses only once for each pair of names following the second name in the pair. Here is an example:

```
change old1=new1 (et=log)
      old1=new2 (et=help);
```

See [“Specifying an Entry Type” on page 164](#), [“Example 1: Copying, Deleting, and Moving Catalog Entries from Multiple Catalogs” on page 169](#) and [“Example 2: Displaying Contents, Changing Names, and Changing a Description” on page 173](#)

Catalog Concatenation

About Catalog Concatenation

There are two types of CATALOG concatenation. The first type is specified by the LIBNAME statement and the second type is specified by the global CATNAME statement. All statements and options that can be used on single (unconcatenated) catalogs can be used on catalog concatenations.

Restrictions

When you use the CATALOG procedure to copy concatenated catalogs and you use the NEW option, the following rules apply:

- If the input catalog is a concatenation and if the output catalog exists in any level of the input concatenation, then the copy is not allowed.
- If the output catalog is a concatenation and if the input catalog exists in the first level of the output concatenation, then the copy is not allowed.

For example, the following code demonstrates these two rules, and the copy fails:

```
libname first 'SAS-library-1';
libname second 'SAS-library-2';
/* create concat.x */
libname concat (first second);

/* fails rule #1 */
proc catalog c=concat.x;
  copy out=first.x new;
run;
quit;

/* fails rule #2 */
proc catalog c=first.x;
```

```

copy out=concat.x new;
run;
quit;

```

In summary, the following table shows when copies are allowed. In the table, A and B are libraries, and each contains catalog X. Catalog C is an automatic concatenation of A and B, and catalog D is an automatic concatenation of B and A.

Table 9.3 *Allowing Copies*

Input Catalog	Output Catalog	Copy Allowed?
C.X	B.X	No
C.X	D.X	No
D.X	C.X	No
A.X	A.X	No
A.X	B.X	Yes
B.X	A.X	Yes
C.X	A.X	No
B.X	C.X	Yes
A.X	C.X	No

Results: CATALOG Procedure

Results: CATALOG Procedure

The CATALOG procedure produces output when the CONTENTS statement is executed without options. The procedure output is assigned a name. You can use this name to reference the table when using the Output Delivery System (ODS) to select tables and create output data sets. For more information, see [“ODS Table Names Produced by Base SAS Procedures” in SAS Output Delivery System: Procedures Guide](#).

Table 9.4 ODS Tables Produced by the CATALOG Procedure

Table Name	Type of Library
Catalog_Random	When the catalog is in a random-access library
Catalog_Sequential	When the catalog is in a sequential library

Examples: CATALOG Procedure

Example 1: Copying, Deleting, and Moving Catalog Entries from Multiple Catalogs

Features: PROC CATALOG statement options
 CAT=
 COPY statement
 DELETE statement

Details

This example demonstrates the following tasks:

- copies entries by excluding a few entries
- copies entries by specifying a few entries
- protects entries from being edited
- moves entries
- deletes entries
- processes entries from multiple catalogs
- processes entries in multiple run groups

The SAS catalog Perm.Sample contains the following entries:

DEFAULT	FORM	Default form for printing
FSLETTER	FORM	Standard form for letters (HP Laserjet)
LOAN	FRAME	Loan analysis application
LOAN	HELP	Information about the application

BUILD	KEYS	Function Key Definitions
LOAN	KEYS	Custom key definitions for application
CREDIT	LOG	credit application log
TEST1	LOG	Inventory program
TEST2	LOG	Inventory program
TEST3	LOG	Inventory program
LOAN	PMENU	Custom menu definitions for applicaticm
CREDIT	PROGRAM	credit application pgm
TEST1	PROGRAM	testing budget applic.
TEST2	PROGRAM	testing budget applic.
TEST3	PROGRAM	testing budget applic.
LOAN	SCL	SCL code for loan analysis application
PASSIST	SLIST	User profile

The SAS catalog Perm.Formats contains the following entries:

REVENUE	FORMAT	FORMAT:MAXLEN=16,16,12
DEPT	FORMATC	FORMAT:MAXLEN=1,1,14

Program

```
libname perm 'SAS-library';

proc catalog cat=perm.sample;
    delete credit.program credit.log;
run;

    copy out=tcatal1;
run;

    copy out=testcat;
    exclude test1 test2 test3 passist (et=slist) / et=log;
run;

    copy out=logcat move;
    select test1 test2 test3 / et=log;
run;

    copy out=perm.finance noedit;
    select loan.frame loan.help loan.keys loan.pmenu;
run;

    copy in=perm.formats out=perm.finance;
    select revenue.format dept.formatc;
run;
quit;
```

Program Description

Assign a library reference to a SAS library. The LIBNAME statement assigns the libref Perm to the SAS library that contains a permanent SAS catalog.

```
libname perm 'SAS-library';
```


Delete two entries from the Perm.Sample catalog.

```
proc catalog cat=perm.sample;
  delete credit.program credit.log;
run;
```

Copy all entries in the Perm.Sample catalog to the Work.TCatAll catalog.

```
copy out=tcatal1;
run;
```

Copy everything except three LOG entries and Passist.Slist from Perm.Sample to Work.TestCat. The EXCLUDE statement specifies which entries not to copy. ET= specifies a default type. (ET=) specifies an exception to the default type.

```
copy out=testcat;
  exclude test1 test2 test3 passist (et=slist) / et=log;
run;
```

Move three LOG entries from Perm.Sample to Work.LogCat. The SELECT statement specifies which entries to move. ET= restricts processing to LOG entries.

```
copy out=logcat move;
  select test1 test2 test3 / et=log;
run;
```

Copy four SAS/AF software entries from Perm.Sample to Perm.Finance. The NOEDIT option protects these entries in Perm.Finance from further editing with PROC BUILD.

```
copy out=perm.finance noedit;
  select loan.frame loan.help loan.keys loan.pmenu;
run;
```

Copy two formats from Perm.Formats to Perm.Finance. The IN= option enables you to copy from a different catalog than the one specified in the PROC CATALOG statement. Note the entry types for numeric and character formats: REVENUE.FORMAT is a numeric format and DEPT.FORMATC is a character format. The COPY and SELECT statements execute before the QUIT statement ends the PROC CATALOG step.

```
copy in=perm.formats out=perm.finance;
  select revenue.format dept.formatc;
run;
quit;
```

The SAS Log

Example Code 9.1 Copying, Protecting, Removing, Deleting, and Processing Entries Using PROC CATALOG

```

1  libname perm 'SAS-library';
NOTE: Libref PERM was successfully assigned as follows:
      Engine:          V9
      Physical Name:   SAS-library\perm
2  proc catalog cat=perm.sample;
NOTE: Writing HTML Body file: sashtml.htm
3      delete credit.program credit.log;
4  run;

NOTE: Deleting entry CREDIT.PROGRAM in catalog PERM.SAMPLE.
NOTE: Deleting entry CREDIT.LOG in catalog PERM.SAMPLE.
5      copy out=tcatal;
6  run;

NOTE: Copying entry DEFAULT.FORM from catalog PERM.SAMPLE to catalog WORK.TCATALL.
NOTE: Copying entry FSLETTER.FORM from catalog PERM.SAMPLE to catalog WORK.TCATALL.
NOTE: Copying entry LOAN.FRAME from catalog PERM.SAMPLE to catalog WORK.TCATALL.
NOTE: Copying entry LOAN.HELP from catalog PERM.SAMPLE to catalog WORK.TCATALL.
NOTE: Copying entry BUILD.KEYS from catalog PERM.SAMPLE to catalog WORK.TCATALL.
NOTE: Copying entry LOAN.KEYS from catalog PERM.SAMPLE to catalog WORK.TCATALL.
NOTE: Copying entry TEST1.LOG from catalog PERM.SAMPLE to catalog WORK.TCATALL.
NOTE: Copying entry TEST2.LOG from catalog PERM.SAMPLE to catalog WORK.TCATALL.
NOTE: Copying entry TEST3.LOG from catalog PERM.SAMPLE to catalog WORK.TCATALL.
NOTE: Copying entry LOAN.PMENU from catalog PERM.SAMPLE to catalog WORK.TCATALL.
NOTE: Copying entry TEST1.PROGRAM from catalog PERM.SAMPLE to catalog WORK.TCATALL.
NOTE: Copying entry TEST2.PROGRAM from catalog PERM.SAMPLE to catalog WORK.TCATALL.
NOTE: Copying entry TEST3.PROGRAM from catalog PERM.SAMPLE to catalog WORK.TCATALL.
NOTE: Copying entry LOAN.SCL from catalog PERM.SAMPLE to catalog WORK.TCATALL.
NOTE: Copying entry PASSIST.SLIST from catalog PERM.SAMPLE to catalog WORK.TCATALL.
7      copy out=testcat;
8          exclude test1 test2 test3  passist (et=slist) / et=log;
9  run;

NOTE: Copying entry DEFAULT.FORM from catalog PERM.SAMPLE to catalog WORK.TESTCAT.
NOTE: Copying entry FSLETTER.FORM from catalog PERM.SAMPLE to catalog WORK.TESTCAT.
NOTE: Copying entry LOAN.FRAME from catalog PERM.SAMPLE to catalog WORK.TESTCAT.
NOTE: Copying entry LOAN.HELP from catalog PERM.SAMPLE to catalog WORK.TESTCAT.
NOTE: Copying entry BUILD.KEYS from catalog PERM.SAMPLE to catalog WORK.TESTCAT.
NOTE: Copying entry LOAN.KEYS from catalog PERM.SAMPLE to catalog WORK.TESTCAT.
NOTE: Copying entry LOAN.PMENU from catalog PERM.SAMPLE to catalog WORK.TESTCAT.
NOTE: Copying entry TEST1.PROGRAM from catalog PERM.SAMPLE to catalog WORK.TESTCAT.
NOTE: Copying entry TEST2.PROGRAM from catalog PERM.SAMPLE to catalog WORK.TESTCAT.
NOTE: Copying entry TEST3.PROGRAM from catalog PERM.SAMPLE to catalog WORK.TESTCAT.
NOTE: Copying entry LOAN.SCL from catalog PERM.SAMPLE to catalog WORK.TESTCAT.
10     copy out=logcat move;
11     select test1 test2 test3 / et=log;
12 run;

NOTE: Moving entry TEST1.LOG from catalog PERM.SAMPLE to catalog WORK.LOGCAT.
NOTE: Moving entry TEST2.LOG from catalog PERM.SAMPLE to catalog WORK.LOGCAT.
NOTE: Moving entry TEST3.LOG from catalog PERM.SAMPLE to catalog WORK.LOGCAT.
13     copy out=perm.finance noedit;
14     select loan.frame loan.help loan.keys loan.pmenu;
15 run;

```

```
NOTE: Copying entry LOAN.FRAME from catalog PERM.SAMPLE to catalog PERM.FINANCE.
NOTE: Copying entry LOAN.HELP from catalog PERM.SAMPLE to catalog PERM.FINANCE.
NOTE: Copying entry LOAN.KEYS from catalog PERM.SAMPLE to catalog PERM.FINANCE.
NOTE: Copying entry LOAN.PMENU from catalog PERM.SAMPLE to catalog PERM.FINANCE.
16      copy in=perm.formats out=perm.finance;
17      select revenue.format dept.formatc;
18      quit;

NOTE: Copying entry REVENUE.FORMAT from catalog PERM.FORMATS to catalog PERM.FINANCE.
NOTE: Copying entry DEPT.FORMATC from catalog PERM.FORMATS to catalog PERM.FINANCE.
```

Example 2: Displaying Contents, Changing Names, and Changing a Description

Features:

- PROC CATALOG statement options
- CATALOGS=
- CHANGE statement
- CONTENTS statement
- MODIFY statement
- TITLE statement

Details

This example demonstrates the following tasks:

- lists the entries in a catalog and routes the output to a file
- changes entry names
- changes entry descriptions
- processes entries in multiple run groups

Program

```
libname perm 'SAS-library';

proc catalog catalog=perm.finance;
  contents;
  title1 'Contents of PERM.FINANCE before changes are made';
run;

  change dept=deptcode (et=formatc);
run;

  modify loan.frame (description='Loan analysis app. - ver1');
  contents;
```

```

title1 'Contents of PERM.FINANCE after changes are made';
run;

quit;

```

Program Description

Assign a library reference. The LIBNAME statement assigns a libref to the SAS library that contains a permanent SAS catalog.

```
libname perm 'SAS-library';
```

List the entries in a catalog and route the output to a file. The CONTENTS statement creates a listing of the contents of the SAS catalog Perm.Finance and routes the output to a file.

```

proc catalog catalog=perm.finance;
  contents;
title1 'Contents of PERM.FINANCE before changes are made';
run;

```

Change entry names. The CHANGE statement changes the name of an entry that contains a user-written character format. (ET=) specifies the entry type.

```

change dept=deptcode (et=formatc);
run;

```

Process entries in multiple run groups. The MODIFY statement changes the description of an entry. The CONTENTS statement creates a listing of the contents of Perm.Finance after all the changes have been applied. QUIT ends the procedure.

```

modify loan.frame (description='Loan analysis app. - ver1');
contents;
title1 'Contents of PERM.FINANCE after changes are made';
run;

quit;

```

Output Examples

Output 9.1 Contents of Perm.Finance before and After Changes Are Made

Contents of PERM.FINANCE before changes are made

Contents of Catalog PERM.FINANCE					
#	Name	Type	Create Date	Modified Date	Description
1	REVENUE	FORMAT	10/16/1996 13:48:11	10/16/1996 13:48:11	FORMAT:MAXLEN=16,16,12
2	DEPTCODE	FORMATC	10/30/1996 13:40:42	10/30/1996 13:40:42	FORMAT:MAXLEN=1,1,14
3	LOAN	FRAME	10/30/1996 13:40:43	02/18/2014 13:31:25	Loan analysis app. - ver1
4	LOAN	HELP	10/16/1996 13:48:10	10/16/1996 13:48:10	Information about the application
5	LOAN	KEYS	10/16/1996 13:48:10	10/16/1996 13:48:10	Custom key definitions for application
6	LOAN	PMENU	10/16/1996 13:48:10	10/16/1996 13:48:10	Custom menu definitions for application
7	LOAN	SCL	10/16/1996 13:48:10	10/16/1996 13:48:10	SCL code for loan analysis application

Contents of PERM.FINANCE after changes are made

Contents of Catalog PERM.FINANCE					
#	Name	Type	Create Date	Modified Date	Description
1	REVENUE	FORMAT	10/16/1996 13:48:11	10/16/1996 13:48:11	FORMAT:MAXLEN=16,16,12
2	DEPTCODE	FORMATC	10/30/1996 13:40:42	10/30/1996 13:40:42	FORMAT:MAXLEN=1,1,14
3	LOAN	FRAME	10/30/1996 13:40:43	03/28/2014 10:26:50	Loan analysis app. - ver1
4	LOAN	HELP	10/16/1996 13:48:10	10/16/1996 13:48:10	Information about the application
5	LOAN	KEYS	10/16/1996 13:48:10	10/16/1996 13:48:10	Custom key definitions for application
6	LOAN	PMENU	10/16/1996 13:48:10	10/16/1996 13:48:10	Custom menu definitions for application
7	LOAN	SCL	10/16/1996 13:48:10	10/16/1996 13:48:10	SCL code for loan analysis application

Example 3: Using the FORCE Option with the KILL Option

Features:

- PROC CATALOG statement options
 - CATALOG=
 - FORCE
 - KILL
- %MACRO statement
- %MEND statement
- %PUT statement

Details

This example demonstrates the following tasks:

- creates a resource environment
- tries to delete all catalog entries by using the KILL option but receives an error
- specifies the FORCE option to successfully delete all catalog entries by using the KILL option.

Program

```
%macro matt;  
    %put &syscc;  
%mend matt;  
  
proc catalog c=work.sasmacr kill;  
run;  
quit;  
  
proc catalog c=work.sasmacr kill force;  
run;  
quit;
```

Program Description

Start a process (resource environment). Do this by opening the catalog entry MATT in the Work.SasMacr catalog.

```
%macro matt;  
    %put &syscc;  
%mend matt;
```

Specify the KILL option to delete all catalog entries in Work.SasMacr. Since there is a resource environment (process using the catalog), KILL does not work and an error is sent to the log.

```
proc catalog c=work.sasmacr kill;  
run;  
quit;
```

Specify the FORCE option to the KILL option to delete the catalog entries.

```
proc catalog c=work.sasmacr kill force;  
run;  
quit;
```

Log Examples

Example Code 9.2 KILL Option Causes Error to Be Sent to the SAS Log

```
1      %macro matt;
2          %put &syscc;
3      %mend matt;
4
5      proc catalog c=work.sasmacr kill;
NOTE: Writing HTML Body file: sashtml.htm
6      run;

ERROR: You cannot open WORK.SASMACR.CATALOG for update access because
WORK.SASMACR.CATALOG is in
use by you in resource environment _O_TAGS.
WARNING: Command CATALOG not processed because of errors noted above.
7      quit;

NOTE: The SAS System stopped processing this step because of errors.
NOTE: PROCEDURE CATALOG used (Total process time):
      real time          6.46 seconds
      cpu time           0.62 seconds
```

Example Code 9.3 Adding the FORCE Option to the KILL Option to Delete the Catalog Entry

```
8      proc catalog c=work.sasmacr kill force;
9      run;

NOTE: Deleting entry MATT.MACRO in catalog WORK.SASMACR.
10     quit;

NOTE: PROCEDURE CATALOG used (Total process time):
      real time          0.01 seconds
      cpu time           0.01 seconds
```


CIMPORT Procedure

Overview: CIMPORT Procedure	179
What Does the CIMPORT Procedure Do?	179
Process for Creating and Reading a Transport File	180
Converting Data to UTF-8	180
Syntax: CIMPORT Procedure	181
PROC CIMPORT Statement	182
EXCLUDE Statement	190
SELECT Statement	191
Usage: CIMPORT Procedure	193
CIMPORT Problems: Importing Transport Files	193
Examples: CIMPORT Procedure	201
Example 1: Importing an Entire Library	201
Example 2: Importing Individual Catalog Entries	202
Example 3: Importing a Single Indexed SAS Data Set	204
Example 4: Using PROC CIMPORT to Import a French Data Set into a UTF-8 SAS Session	205

Overview: CIMPORT Procedure

What Does the CIMPORT Procedure Do?

Note: In this section, when we state that PROC CIMPORT and PROC CIMPORT run in the SAS Viya platform, we are stating that these procedures run on the Compute Server.

The CIMPORT procedure *imports* a transport file that was created (*exported*) by the CPORT procedure. PROC CIMPORT restores the transport file to its original form as a SAS catalog, SAS data set, or SAS library. *Transport files* are sequential files that each contain a SAS library, a SAS catalog, or a SAS data set in transport format. The transport format that PROC CPORT writes is the same for all environments and for many releases of SAS.

PROC CIMPORT also *converts* SAS files, which means that it changes the format of a SAS file from the SAS format appropriate for one version of SAS to the SAS format appropriate for another version. For example, you can use PROC CPORT and PROC CIMPORT to move files from earlier releases of SAS to more recent releases (for example, from SAS®9 to SAS Viya) or between the same versions (for example, from one SAS Viya operating environment to another SAS 9 operating environment). PROC CIMPORT automatically converts the transport file as it imports it.

Note: PROC CIMPORT and PROC CPORT can be used to back up graphic catalogs. PROC COPY cannot be used to back up graphic catalogs.

PROC CIMPORT produces no output, but it does write notes to the SAS log.

Process for Creating and Reading a Transport File

Here is the process to create a transport file at the source computer and to read it at a target computer:

- 1 A transport file is created at the source computer using PROC CPORT.
- 2 The transport file is transferred from the source computer to the target computer.
- 3 The transport file is read at the target computer using PROC CIMPORT.

Note: Transport files that are created using PROC CPORT are not interchangeable with transport files that are created using the XPORT engine.

For complete details about the steps to create a transport file (PROC CPORT), to transfer the transport file, and to restore the transport file (PROC CIMPORT), see [Moving and Accessing SAS Files](#).

Converting Data to UTF-8

Note: In this section, when we state that PROC CPORT and PROC CIMPORT run in the SAS Viya platform, we are stating that these procedures run on the Compute Server.

The Compute Server supports UTF-8 and other legacy encodings. The CIMPORT procedure can convert a data set that is not UTF-8 into a UTF-8 session successfully if the transport file has enough encoding information. Because UTF-8 requires more variable length than other encodings, specifying the EXTENDVAR= option can help prevent data from being truncated. Specifying the EXTENDFORMAT= option can ensure that the format width is sufficient for converting the data.

Note: The formats supported are SAS formats only. User-defined formats are not supported.

For more information about migrating data to UTF-8 encoding, see the following resources.

- [“UTF-8 Encoding” in Migrating Data to UTF-8 for the SAS Viya Platform](#)
- [“Read External Files” in Migrating Data to UTF-8 for the SAS Viya Platform](#)

For information about how to set encodings in SAS Data and AI Studio, see [“Opening a Program” in SAS Data and AI Studio: User's Guide](#).

Syntax: CIMPORT Procedure

Tip: Use PROC CIMPORT or PROC CPORT when backing up graphic catalogs. PROC COPY cannot be used to back up graphic catalogs.

See: For complete details about the steps to create a transport file (PROC CPORT), to transfer the transport file, and to restore the transport file (PROC CIMPORT), see [Moving and Accessing SAS Files](#)

PROC CIMPORT *destination=libref* | *<libref.> member-name* *<options>*;

EXCLUDE *SAS file(s) | catalog entry(s)* *< / MEMTYPE=mtype > < / ENTRYTYPE=entry-type >*;

SELECT *SAS file(s) | catalog entry(s)* *< / MEMTYPE=mtype > < / ENTRYTYPE=entry-type >*;

Statement	Task	Example
PROC CIMPORT	Import a transport file	Ex. 1, Ex. 2, Ex. 3, Ex. 4
EXCLUDE	Exclude one or more specified files from the import process	
SELECT	Specify one or more files or entries to import process	Ex. 2

PROC CIMPORT Statement

Imports a transport file.

Examples: [“Example 1: Importing an Entire Library” on page 201](#)
 [“Example 2: Importing Individual Catalog Entries” on page 202](#)
 [“Example 3: Importing a Single Indexed SAS Data Set” on page 204](#)
 [“Example 4: Using PROC CIMPORT to Import a French Data Set into a UTF-8 SAS Session” on page 205](#)

Syntax

PROC CIMPORT *destination=libref* | *<libref.> member-name* *<options>*;

Summary of Optional Arguments

COMPRESS=NO | CHAR | BINARY

Specifies how the resulting CIMPORT data set is to be compressed.

FORCE

enables access to a locked catalog.

NEW

creates a new catalog for the imported transport file, and deletes any existing catalog with the same name.

NOEDIT

imports SAS/AF PROGRAM and SCL entries without Edit capability.

NOSRC

suppresses the importing of source code for SAS/AF entries that contain compiled SCL code.

SORT

causes the output data set to be re-sorted if necessary when PROC CIMPORT is used to import a sorted data set.

Control the contents of the transport file

EXTENDFORMAT=YES | NO

specifies whether to extend the character format width.

EXTENDSN=YES | NO

specifies whether to extend by 1 byte the length of short numerics (less than 8 bytes) when you import them.

EXTENDVAR=*multiplier* | AUTO

specifies a multiplier value that expands character variable lengths.

UPCASE

writes the alphabetic characters in uppercase to the output file.

Identify the input transport file**INFILE=***fileref* | '*filename*'

specifies a previously defined fileref or the filename of the transport file to read.

TAPE

reads the input transport file from a tape.

Look at the encoding of the transport file**ENCODINGINFO=**ALL | *n*

specifies the number of data set headers to read in order to output the encoding value of the data set to the log.

ISFILEUTF8=YES | NO

specifies whether the file is encoded in UTF-8 format.

Select files to import**EET=**(*etype(s)*)

excludes specified entry types from the import process.

ET=(*etype(s)*)

specifies entry types to import.

MEMTYPE=*mtype*

specifies that only data sets, only catalogs, or both, be moved when a library is imported.

Required Argument***destination=libref* | <*libref*.>*member-name***

identifies the type of file to import and specifies the catalog, SAS data set, or SAS library to import.

destination

identifies the file or files in the transport file as a single catalog, as a single SAS data set, or as the members of a SAS library. The *destination* argument can be one of the following:

CATALOG=

specifies that the file type is a SAS catalog.

Aliases C=

CAT=

DATA=

specifies that the file type is a SAS data set.

Aliases D=

DS=

LIBRARY=

specifies that the file type is a SAS library.

Aliases L=

LIB=

libref**<libref.>member-name**

specifies the specific catalog, SAS data set, or SAS library as the destination of the transport file. If the *destination* argument is CATALOG or DATA, you can specify both a *libref* and a member name. If the *libref* is omitted, PROC CIMPORT uses the default library as the *libref*, which is usually the WORK library. If the *destination* argument is LIBRARY, specify only a *libref*.

See Refer to [“SAS Names” in SAS Programmer’s Guide: Essentials](#) for naming conventions that you can use for names and member names.

Optional Arguments

COMPRESS=NO | CHAR | BINARY

specifies whether the resulting CIMPORT data set is compressed. You can specify the compression type that is being used.

NO

specifies that the data set produced by CIMPORT is not compressed.

Aliases N

OFF

CHAR

specifies that the observations in a newly created SAS data set are compressed (producing variable-length records) by using RLE (Run Length Encoding). RLE compresses observations by reducing repeated runs of the same character (including blanks) to two-byte or three-byte representations.

Aliases ON

YES

Y

BINARY

specifies that the observations in a newly created SAS data set are compressed (producing variable-length records) by using RDC (Ross Data Compression). RDC combines run-length encoding and sliding-window compression to compress the file by representing repeated byte patterns more efficiently.

Note: This method is highly effective for compressing medium to large (several hundred bytes or larger) blocks of binary data (character and numeric variables). Because the compression function operates on a single record at a time, the record length needs to be several hundred bytes or larger for effective compression.

See For more information about creating and restoring transport files, see [Moving and Accessing SAS Files](#).

Example

```
proc cimport file=transportFile lib=myLib compress=char;
run;
```

EET=(etype(s))

excludes specified entry types from the import process. If the *etype* is a single entry type, then you can omit the parentheses. Separate multiple values with spaces.

Interaction You cannot specify both the EET= option and the ET= option in the same PROC CIMPORT step.

ENCODINGINFO=ALL | *n*

specifies the number of data set headers to read in order to output the encoding value of the data set to the log.

ALL

specifies that all data set headers are read. The encoding value stored for the data set header is output to the SAS log.

n

specifies an integer greater than zero. This value represents the number of data set headers to read. The encoding value stored for the data set header is output to the SAS log.

Range Specify an integer greater than zero. The upper limit is dependent on system constraints. All data is still processed when *n* is larger than the number of data sets in the transport file.

Note If catalogs are encountered in the library during the process of reading the data set headers, they are skipped. Catalog headers do not contain encoding values.

See For more information about creating and restoring transport files, see [Moving and Accessing SAS Files](#).

Example

```
proc cimport lib=work file='mixed' encodinginfo=3;
run;
```

NOTE: The CPORTed data set ASCII transport file encoding=us-ascii.

NOTE: The CPORTed data set JISDS transport file encoding=wlatin1.

NOTE: The CPORTed data set UTF8 transport file encoding=wlatin1.

NOTE: The CPORTed data set WLATIN1DS transport file encoding=wlatin1.

ET=(etype(s))

specifies the entry types to import. If the *etype* is a single entry type, then you can omit the parentheses. Separate multiple values with spaces.

Interaction You cannot specify both the EET= option and the ET= option in the same PROC CIMPORT step.

EXTENDSN=YES | NO

specifies whether to extend by 1 byte the length of short numerics (fewer than 8 bytes) when you import them. You can avoid a loss of precision when you

transport a short numeric in IBM format to IEEE format if you extend its length. You cannot extend the length of an 8-byte short numeric.

Default YES

Restriction This option applies only to data sets.

Tip Do not store fractions as short numerics.

EXTENDFORMAT=YES | NO

specifies whether to extend the character format width.

Aliases Y

N

Default YES

Restrictions This option is supported in SAS Viya, but is not supported in SAS 9.

The EXTENDFORMAT= option is used in conjunction with the EXTENDVAR= option.

EXTENDVAR=*multiplier* | AUTO

specifies a multiplier value that expands character variable lengths. The expanded length helps to ensure that character data is not truncated when you are processing a SAS data file that requires transcoding.

multiplier

specifies a multiplier value from 1 to 5 or you can specify AUTO. The lengths for character variables are increased by multiplying the current length by the specified value. The default value is 1. When 1 is used, the variable length is not extended. When AUTO is specified, the multiplier is set automatically.

Auto

AUTO is used to set the *multiplier*. This value is dependent on the encoding of the session and the data set in the transport file. When the file that is specified by INFILE= is in ANSI encoding or has no character encoding ID (CEI) information, a multiplier value of one is used. Otherwise, the extended length is calculated for you.

When the string is truncated, a warning message is generated. The message prompts you to use the EXTENDVAR option.

Default 1. When the default is used, the variable length is not extended.

Restrictions This option is supported in SAS Viya but is not supported in SAS 9.

This option applies only to data sets.

This option does not take affect when the NOTRANS CODE attribute is specified for data types for SAS character variables.

PROC CIMPORT does not support VARCHAR.

Note If the ASIS option is specified in PROC CPORT, the EXTENDVAR= option is ignored.

ISFILEUTF8=YES | NO

explicitly designates the encoding of a data set that is contained in a transport file as UTF-8. Although data set encodings are recorded (or stamped) in SAS 9.2 transport files, encodings are not stamped in transport files created using SAS releases before 9.2. Therefore, designating the UTF-8 encoding is useful under these conditions:

- The data set in the transport file was created using a SAS release before 9.2.
- The data set is known to be encoded as UTF-8.

The person who restores the transport file in the target environment should have a description of the transport file in advance of the restore operation.

yes

specifies that the data set in the transport file is encoded as UTF-8.

Aliases YES
Y
TRUE
T

no

specifies that the data set in the transport file is not encoded as UTF-8. NO is the default.

Aliases NO
N
FALSE
F

In order to successfully import a transport file in the target SAS session, you should have this information about the transport file:

- source operating environment. **Linux** for example.
- SAS release. **SAS Viya 4.0** for example.
- name of the transport file. **tport.dat** for example.
- encoding of the character data. **wlatin1** for example.
- national language of the character data. For example, American English (or **en_US**)

Default NO

Restriction PROC CIMPORT uses this option only if the transport file is not stamped with the encoding of the data set. Encodings were not recorded in SAS releases before 9.2. If an encoding is recorded in the

transport file and the ISFILEUTF8= option is specified in PROC CIMPORT, ISFILEUTF8= is ignored.

See [“CIMPORT Problems: Importing Transport Files” on page 193](#)

For more information about creating and restoring transport files, see [Moving and Accessing SAS Files](#).

FORCE

enables access to a locked catalog. By default, PROC CIMPORT locks the catalog that it is updating to prevent other users from accessing the catalog while it is being updated. The FORCE option overrides this lock, which allows other users to access the catalog while it is being imported, or enables you to import a catalog that is currently being accessed by other users.

CAUTION

The FORCE option can lead to unpredictable results. The FORCE option allows multiple users to access the same catalog entry simultaneously.

INFILE=*fileref* | '*filename*'

specifies a previously defined fileref or the filename of the transport file to read. If you omit the INFILE= option, then PROC CIMPORT attempts to read from a transport file with the fileref SASCAT. If a fileref SASCAT does not exist, then PROC CIMPORT attempts to read from a file named SASCAT.DAT.

Alias FILE=

Example [“Example 1: Importing an Entire Library” on page 201](#)

MEMTYPE=*mtype*

specifies that only data sets, only catalogs, or both, be imported from the transport file. Values for *mtype* can be as follows:

Alias MT=

ALL

specifies both catalogs and data sets be imported.

CATALOG

specifies only catalogs be imported.

Alias CAT

DATA

specifies only data sets be imported.

Alias DS

NEW

creates a new catalog to contain the contents of the imported transport file when the destination that you specify has the same name as an existing catalog. NEW deletes any existing catalog with the same name as the one you specify as a destination for the import. If you do not specify NEW, and the destination that

you specify has the same name as an existing catalog, PROC CIMPORT appends the imported transport file to the existing catalog.

NOEDIT

imports SAS/AF PROGRAM and SCL entries without Edit capability.

You obtain the same results if you create a new catalog to contain SCL code by using the MERGE statement with the NOEDIT option in the BUILD procedure of SAS/AF software.

Note: The NOEDIT option affects only SAS/AF PROGRAM and SCL entries. It does not affect FSEDIT SCREEN and FSVIEW FORMULA entries.

Alias NEDIT

NOSRC

suppresses the importing of source code for SAS/AF entries that contain compiled SCL code.

You obtain the same results if you create a new catalog to contain SCL code by using the MERGE statement with the NOSOURCE option in the BUILD procedure of SAS/AF software.

Alias NSRC

Interaction PROC CIMPORT ignores the NOSRC option if you use it with an entry type other than FRAME, PROGRAM, or SCL.

SORT

causes the output data set to be re-sorted if necessary when importing a sorted data set using PROC CIMPORT. A re-sort is necessary if the data set to be imported is sorted by one or more character variables and the ordering of character values in the target SAS session is different from that of the source session.

Note: The CIMPORT SORT option has no effect on data sets that do not contain sort information. This applies only to data sets that were sorted previously.

The ordering of character values, or collating sequence, is dependent upon the session encoding and translation tables used by SAS as well as by the options used when invoking the SORT procedure. If necessary, a re-sort is performed during the import, and the data set's sort indicator is preserved.

For more information, see [“Collating Sequence” in SAS National Language Support \(NLS\): Reference Guide](#) and [Chapter 55, “SORT Procedure,” on page 2039](#).

When PROC CIMPORT re-sorts a data set, the following information is output:

NOTE: PROC CIMPORT re-sorted the data set WORK.TEMP because it contained character variables in the sort key, and the collating sequence of the sort differs from the local host.

Example `proc cimport 'transportfile.tpt' lib=library sort; run;`

TAPE

reads the input transport file from a tape.

Default PROC CIMPORT reads from disk.

UPCASE

reads from the transport file and writes the alphabetic characters in uppercase to the output file.

Restriction The UPCASE option is allowed only if SAS is built with a Double-Byte Character Set (DBCS).

Tip PROC CPORT can be used to create a transport file that includes all uppercase characters. See option [“OUTTYPE=UPCASE” on page 350](#) for details.

EXCLUDE Statement

Excludes specified files or entries from the import process.

Interaction: You can use either EXCLUDE statements or SELECT statements in a PROC CIMPORT step, but not both.

Tip: There is no limit to the number of EXCLUDE statements that you can use in one invocation of PROC CIMPORT.

Syntax

```
EXCLUDE SAS file(s) | catalog entry(s) < / MEMTYPE=mtype >
< / ENTRYTYPE=entry-type >;
```

Required Argument

SAS file(s) | catalog entry(s)

specifies one or more SAS files or one or more catalog entries to be excluded from the import process. Specify SAS filenames if you import a library; specify catalog entry names if you import an individual SAS catalog. Separate multiple filenames or entry names with a space. You can use shortcuts to list many like-named files in the EXCLUDE statement. For more information, see [“SAS Files” in SAS Programmer's Guide: Essentials](#).

Optional Arguments

ENTRYTYPE=entry-type

specifies a single entry type for one or more catalog entries that are listed in the EXCLUDE statement. See [“Catalog Names and Catalog Entries” in SAS V9 LIBNAME Engine: Reference](#).

Aliases ETYPE=

ET=

Restriction ENTRYTYPE= is valid only when you import an individual SAS catalog.

MEMTYPE=*mtype*

specifies a single member type for one or more SAS files listed in the EXCLUDE statement. Values for *mtype* can be

ALL

specifies both catalogs and data sets be imported.

CATALOG

specifies only catalogs be imported.

DATA

specifies only data sets be imported.

You can also specify the MEMTYPE= option, enclosed in parentheses, immediately after the name of a file. In parentheses, MEMTYPE= identifies the type of the filename that just precedes it. When you use this form of the option, it overrides the MEMTYPE= option that follows the slash in the EXCLUDE statement, but it must match the MEMTYPE= option in the PROC CIMPORT statement.

Aliases MTYPE=

MT=

Default ALL

Restriction MEMTYPE= is valid only when you import a SAS library.

SELECT Statement

Specifies individual files or entries to import.

Interaction: You can use either EXCLUDE statements or SELECT statements in a PROC CIMPORT step, but not both.

Tip: There is no limit to the number of SELECT statements that you can use in one invocation of PROC CIMPORT.

Example: [“Example 2: Importing Individual Catalog Entries” on page 202](#)

Syntax

```
SELECT SAS file(s) / catalog entry(s) < / MEMTYPE=mtype >
< / ENTRYTYPE=entry-type >;
```

Required Argument

SAS file(s) | catalog entry(s)

specifies one or more SAS files or one or more catalog entries to import. Specify SAS filenames if you import a library; specify catalog entry names if you import an individual SAS catalog. Separate multiple filenames or entry names with a space. You can use shortcuts to list many like-named files in the SELECT statement. For more information, see [“SAS Files” in SAS Programmer’s Guide: Essentials](#).

Optional Arguments

ENTRYTYPE=entry-type

specifies a single entry type for one or more catalog entries that are listed in the SELECT statement. See [“Catalog Names and Catalog Entries” in SAS V9 LIBNAME Engine: Reference](#).

Aliases ETYPE=

ET=

Restriction ENTRYTYPE= is valid only when you import an individual SAS catalog.

MEMTYPE=mtype

specifies a single member type for one or more SAS files listed in the SELECT statement. Valid values are CATALOG or CAT, DATA, or ALL.

You can also specify the MEMTYPE= option, enclosed in parentheses, immediately after the name of a file. In parentheses, MEMTYPE= identifies the type of the filename that just precedes it. When you use this form of the option, it overrides the MEMTYPE= option that follows the slash in the SELECT statement, but it must match the MEMTYPE= option in the PROC CIMPORT statement.

ALL

specifies both catalogs and data sets be imported.

CATALOG

specifies only catalogs be imported.

DATA

specifies only data sets be imported.

Aliases MTYPE=

MT=

Default ALL

Restriction MEMTYPE= is valid only when you import a SAS library.

Usage: CIMPORT Procedure

CIMPORT Problems: Importing Transport Files

About Transport Files and Encodings

The character data in a transport file is created for the following types of encodings:

- the UTF-8 encoding of the SAS session in which the transport file is created.
- the UTF-8 encoding is maintained when you use PROC CIMPORT to import a data set in a UTF-8 session
- PROC CIMPORT supports the ability to import data sets created in non-UTF-8 SAS sessions into UTF-8 SAS sessions.
- When you use PROC CPORT to create a transport file that is encoded with US-ASCII on an ASCII platform, regardless of the session encoding, the US-ASCII encoding is preserved for that transport file. If you then transport that data set to an ASCII platform using PROC CIMPORT, the US-ASCII encoding for that transport file is preserved. The data set that is created has the US-ASCII encoding, not the session encoding. For example, if your session encoding is WLATIN1, you use PROC CPORT to create a data set that has an encoding of US-ASCII. The US-ASCII encoding is preserved in the transport file, instead of the WLATIN1 encoding. This preservation also occurs when you use PROC CIMPORT on this data set. The US-ASCII encoding is preserved and is not when you use PROC CIMPORT to transport the data set to an ASCII platform.

Note: The preservation of the US-ASCII encoding occurs only on an ASCII platform.

The following example shows how SAS applies an encoding to a transport file.

For information about how to set encodings in SAS Data and AI Studio, see [“Opening a Program” in SAS Data and AI Studio: User’s Guide](#).

Table 10.1 Assignment of Encodings to Transport Files

Encoding Value of the Transport File	SAS Invocation	Explanation
utf-8 or us-ascii	encoding is utf8;	A SAS session is invoked using the UTF-8 encoding. The session encoding is applied to the transport file. If the data set is US-ASCII, the US-ASCII encoding is preserved, not the session encoding.
wlatin2	locale is pl_PL;	A SAS session is invoked using encoding, LATIN2, which is associated with the Polish Poland locale.

For a complete list of encodings that are associated with each locale, see [Locale Tables](#) in *SAS National Language Support (NLS): Reference Guide*.

In order for a transport file to be imported successfully, the encodings of the source and target SAS sessions must be compatible. Here is an example of compatible source and target SAS sessions:

Table 10.2 Compatible Encodings

Source SAS Session			Target SAS Session	
Locale	Linux SAS Session Encoding	Transport File Encoding	Locale	SAS Session Encoding
es_MX (Spanish Mexico)	latin1	wlatin1	it_IT (Italian Italy)	wlatin1

The encodings of the source and target SAS sessions are compatible because the default encoding for the es_MX locale is WLATIN1 and the encoding of the target SAS session is WLATIN1.

However, if the encodings of the source and target SAS sessions are incompatible, a transport file might not be successfully imported. Here is an example of incompatible encodings:

Table 10.3 Incompatible Encodings

Source SAS Session			Target SAS Session	
Locale	LINUX SAS Session Encoding	Transport File Encoding	Locale	Encoding
cs_CZ (Czech Czechoslovakia)	latin2	wlatin2	de_DE (German Germany)	latin1

The encodings of the source and target SAS sessions are incompatible because the default encoding for the cs_CZ locale is WLATIN2 and the encoding of the target SAS session is WLATIN1. A transport file cannot be imported between these locales.

The target session encoding is latin1. The CIMPORT transport file is WLATIN2 and can be imported in this session. However, some characters might be improperly changed. The following warning message is provided:

WARNING: The transport file was created in the encoding wlatin2. This session encoding, wlatin1, can read and create transport files in the encoding wlatin1. Data might not be imported correctly.

Warning Messages Generated When Using PROC CIMPORT in the SAS Viya Platform

The SAS Viya platform supports legacy encoding as well as UTF-8 encoding. When you are using PROC CIMPORT to import a data set in the SAS Viya platform, warnings might occur when the data is imported. Some of these warnings are shown in [Table 10.19 on page 195](#).

WARNING: The Dataset Cannot Be Imported

The following table provides a list of warnings for when a data set cannot be imported using PROC CIMPORT, the reason why there is a problem, and information about how PROC CIMPORT handles the issue.

Table 10.4 Warning Messages Generated When Using PROC CIMPORT in the SAS Viya Platform

PROC CIMPORT Transcoding Issues			
Exception in Transcoding	How the SAS Viya platform	Message Displayed	When the Exception Might be Generated

	handles the Exception		
Unsupported character exists in the name of the data set	Show the message and skip the current data set import	WARNING: The dataset cannot be imported because unsupported characters are found in the dataset <i>"mylib.mydata"</i> .	when transcoding from UTF-8 to non-UTF-8
Unsupported character exists in the name of the data set index	Show the message and skip the current data set import	WARNING: The dataset cannot be imported because unsupported characters are found in the name of index <i>"indexString"</i> in the dataset <i>"mylib.mydata"</i> .	when transcoding from UTF-8 to non-UTF-8
Unsupported character exists in the variable names of the data set	Show the message and skip the current data set import	WARNING: The dataset cannot be imported because unsupported characters are found in one or more variable names.	when transcoding from UTF-8 to non-UTF-8
Data set variable label is too long for the target encoding	Show the message and skip the current data set import	WARNING: The dataset cannot be imported because the variable label <i>"labelString"</i> is truncated.	when transcoding from non-UTF-8 to UTF-8
Data set label is too long for target encoding	Show the message and skip the current data set import	WARNING: The dataset cannot be imported because the dataset label <i>"lableString"</i> is truncated.	when transcoding from non-UTF-8 to UTF-8
Unsupported character exists in the observations of the data set	Show the message and skip the current data set import	WARNING: The dataset cannot be imported because unsupported characters are found in one or more observations.	when transcoding from UTF-8 to non-UTF-8
Unsupported character exists in the variable labels of the data set	Show the message and skip the current data set import	WARNING: The dataset cannot be imported because unsupported characters are found in one or more variable labels.	when transcoding from UTF-8 to non-UTF-8

Unsupported character exists in the labels of the data set	Show the message and skip the current data set import	WARNING: The dataset cannot be imported because unsupported characters are found in the dataset label.	when transcoding from UTF-8 to non-UTF-8
--	---	--	--

Problems with Transport Files Created Using SAS Versions 9.2 through 9.4

Overview

The encoding of the character data is stamped in transport files that are created using SAS versions 9.2 and later. Therefore, the CIMPORT procedure can detect error conditions. For example, UTF-8 data sets in a transport file can be imported to sessions that are not UTF-8 sessions. When all the characters in the UTF-8 transport file data set are supported by the target session, the transport file can be imported. Otherwise, the data set that contains the invalid character fails to import and a warning message is issued:

WARNING: The dataset cannot be imported because unsupported characters are found in one or more observations.

SAS versions 9.2 and later can detect the condition of incompatibility between the encoding of the transport file and the locale of the target SAS session. Because some customers' SAS applications ran successfully using a release prior to SAS 9.2, PROC CIMPORT reports a warning only, but allows the import procedure to continue.

Here are the warning and error messages with recovery actions:

- [“Warning: Target Session Does Not Use UTF-8: Transport File Is Not UTF-8” on page 198](#)

Error: Target Session Uses UTF-8: Transport File Is Not UTF-8

This error message should not appear in [SAS 9.4M3](#) and later. The Error message provides the following information:

- The target SAS session uses the UTF-8 encoding.
- The transport file has an identified encoding that is not UTF-8. The encodings of the transport file and the target SAS session are incompatible.

Note: PROC CIMPORT supports the ability to import data sets that are created in non-UTF-8 SAS sessions into UTF-8 SAS sessions.

If this error is generated, the encoding of the target SAS session cannot be UTF-8. The locales of the source and target SAS sessions must be identical.

Here is an example of a SAS 9.2 WLATIN2 transport file and UTF-8 target SAS session.

- 1 To recover, in the target SAS session, start another SAS session and change the locale to the locale that was used in the source SAS session that created the transport file.

The LOCALE= value is preferred over the ENCODING= value because it sets automatically the default values for the ENCODING=, DFLANG=, DATESTYLE=, and PAPERSIZE= options.

If you do not know the locale of the source session (or the transport file), you can infer it from the language that is used by the character data in the transport file.

For example, if you know that Polish is the language, specify the pl_PL (Polish Poland) locale in a new target SAS session. The encoding value associated with pl_PL (Polish Poland) locale on Linux is latin2.

Here is an example of specifying the pl_PL locale in a new SAS session:

```
set locale=pl_PL;
```

For complete details, see [Locale Table](#).

Note: Verify that you do not have a SAS invocation command that already contains the specification of the UTF-8 encoding. For example: `sas9 -encoding utf8`. If it exists, the UTF-8 encoding would persist regardless of a new locale specification.

- 2 Import the file again. Here is an example:

```
filename importin 'transport-file';
libname target 'SAS-data-library';
proc cimport infile=importin library=target memtype=data;
run;
```

PROC CIMPORT should succeed.

Warning: Target Session Does Not Use UTF-8: Transport File Is Not UTF-8

The warning message provides this information:

- The target SAS session uses an identified encoding.
- The encoding of the transport file is identified. The encodings of the transport file and the target SAS session are incompatible.

This table shows the locale and encoding values of incompatible source and target SAS sessions.

Table 10.5 Encoding Values for the Czech and German Locales

SAS Session	POSIX Locale	Linux Encoding
Source SAS Session	cs_CZ (Czech Czechoslovakia)	latin2
Target SAS Session	de_DE (German Germany)	latin9

The transport file is imported, but the contents of the file are questionable. The message identifies the incompatible encoding formats. To recover, try to read the contents of the imported file. If the file is unreadable, perform these steps:

- 1 In the target SAS session, start a new SAS session and change the locale (rather than the encoding) to the locale that is used in the source SAS session.

The `LOCALE=` value is preferred over the `ENCODING=` value because it automatically sets the default values for the `ENCODING=`, `DFLANG=`, `DATESTYLE=`, and `PAPERSIZE=` options.

If you do not know the locale of the source session (or the transport file), you can infer it from the national language of the transport file.

For example, if you know that Czech is the national language, specify the `cs_CZ` locale in a new target SAS session.

Set the `cs_CZ` locale in a new SAS session:

```
-locale cs_CZ;
```

The target SAS session and the transport file use compatible encodings. They both use `wlatin2`.

For complete details, see [the Locale Table](#) in *SAS National Language Support (NLS): Reference Guide*.

- 2 Import the file again. Here is an example:

```
filename importin 'transport-file';
libname target 'sas-library';
proc cimport infile=importin library=target memtype=data;
run;
```

PROC CIMPORT should succeed.

Problems with Transport Files Created Using a SAS Release Prior to SAS 9.2

Overview: SAS Releases Prior to 9.2

Transport files that were created by SAS releases before 9.2 are not stamped with encoding values. Therefore, the CIMPORT procedure does not know the identity of

the transport file's encoding and cannot report specific warning and error detail. The encoding of the transport file must be inferred when performing recovery actions.

However, using your knowledge about the transport file, you should be able to recover from transport problems. For information that is useful for importing the transport file in the target SAS session, see [Tips: for the ISFILEUTF8 option on page 187](#). For complete details about creating and restoring transport files, see [Moving and Accessing SAS Files](#).

Here are the warning and error messages with recovery actions:

- [“Error: Transport File Encoding Is Unknown: Use the ISFILEUTF8= Option” on page 200](#)
- [“Warning: Transport File Encoding Is Unknown” on page 200](#)

Error: Transport File Encoding Is Unknown: Use the ISFILEUTF8= Option

The error message provides this information:

- The transport file was created using a SAS release before 9.2.
- Because the encoding is not stamped in the transport file, the encoding is unknown.
- The target SAS session uses the UTF-8 encoding.

Note: In order to perform recovery steps, you must know the encoding of the transport file.

If you know that the transport file is encoded as UTF-8, you can import the file again, and use the ISFILEUTF8=YES option in PROC CIMPORT.

Here is an example of the UTF-8 transport file and UTF-8 target SAS session. The UTF-8 transport file was created using a SAS release before 9.2.

```
filename importin 'transport-file';
libname target 'sas-library';
proc cimport isfileutf8=yes infile=importin library=target
  memtype=data;
run;
```

For syntax details, see [the ISFILEUTF8= option on page 187](#).

PROC CIMPORT should succeed.

Warning: Transport File Encoding Is Unknown

The warning message provides this information:

- The transport file was created using a SAS release before 9.2.
- Because the encoding is not stamped in the transport file, the encoding is unknown.

Try to read the character data from the imported data set. If you cannot read the data, you can infer that the locale of the target SAS session is incompatible with the encoding of the transport file.

Note: In order to perform recovery steps, you must know the encoding of the transport file.

For example, the transport file was created in a source SAS session using a SAS release before 9.2. The transport file was also created using a Polish Poland locale. The target SAS session uses a German locale.

- 1 In the target SAS session, start another SAS session and change the locale to the locale of the source SAS session that created the transport file.

In this example, you start a new SAS session in the Polish Poland locale.

```
sas9 -locale pl_PL;
```

- 2 Import the file again. Here is an example:

```
filename importin 'transport-file';
libname target 'sas-library';
proc cimport infile=importin library=target memtype=data;
run;
```

PROC CIMPORT should succeed and the data should be readable in the SAS session that uses a Polish_Poland locale.

Examples: CIMPORT Procedure

Example 1: Importing an Entire Library

Features:	PROC CIMPORT statement option INFILE=
-----------	--

Details

This example shows how to use PROC CIMPORT to read from disk a transport file, named TRANFILE, that PROC CPORT created from a SAS library in another operating environment. The transport file was moved to the new operating environment. PROC CIMPORT imports the transport file to a SAS library, called NEWLIB, in the new operating environment.

Program

```
libname newlib 'sas-library';
filename tranfile 'transport-file';

host-options-for-file-characteristics;

proc cimport library=newlib infile=tranfile;
run;
```

Program Description

Specify the library name and filename. The LIBNAME statement specifies a LIBNAME for the new SAS library. The FILENAME statement specifies the filename of the transport file that PROC CIMPORT created and enables you to specify any operating environment options for file characteristics.

```
libname newlib 'sas-library';
filename tranfile 'transport-file';

host-options-for-file-characteristics;
```

Import the SAS library in the NEWLIB library. PROC CIMPORT imports the SAS library into the library named NEWLIB.

```
proc cimport library=newlib infile=tranfile;
run;
```

Log Examples

Example Code 10.1 Importing an Entire Library

```
NOTE: Proc CIMPORT begins to create/update catalog NEWLIB.FINANCE
NOTE: Entry LOAN.FRAME has been imported.
NOTE: Entry LOAN.HELP has been imported.
NOTE: Entry LOAN.KEYS has been imported.
NOTE: Entry LOAN.PMENU has been imported.
NOTE: Entry LOAN.SCL has been imported.
NOTE: Total number of entries processed in catalog NEWLIB.FINANCE: 5

NOTE: Proc CIMPORT begins to create/update catalog NEWLIB.FORMATS
NOTE: Entry REVENUE.FORMAT has been imported.
NOTE: Entry DEPT.FORMATC has been imported.
NOTE: Total number of entries processed in catalog NEWLIB.FORMATS: 2
```

Example 2: Importing Individual Catalog Entries

Features: PROC CIMPORT statement option

INFILE=
SELECT statement

Details

This example shows how to use PROC CIMPORT to import the individual catalog entries LOAN.PMENU and LOAN.SCL from the transport file TRANS2, which was created from a single SAS catalog.

Program

```
libname newlib 'sas-library';  
filename trans2 'transport-file';  
  
host-options-for-file-characteristics;  
  
proc cimport catalog=newlib.finance infile=trans2;  
    select loan.pmenu loan.scl;  
run;
```

Program Description

Specify the library name, filename, and operating environment options. The LIBNAME statement specifies a LIBNAME for the new SAS library. The FILENAME statement specifies the filename of the transport file that PROC CIMPORT created and enables you to specify any operating environment options for file characteristics.

```
libname newlib 'sas-library';  
filename trans2 'transport-file';  
  
host-options-for-file-characteristics;
```

Import the specified catalog entries to the new SAS catalog. PROC CIMPORT imports the individual catalog entries from the TRANS2 transport file and stores them in a new SAS catalog called NEWLIB.FINANCE. The SELECT statement selects only the two specified entries from the transport file to be imported into the new catalog.

```
proc cimport catalog=newlib.finance infile=trans2;  
    select loan.pmenu loan.scl;  
run;
```

Log Examples

Example Code 10.2 Importing Individual Catalog Entries

```
NOTE: Proc CIMPORT begins to create/update catalog NEWLIB.FINANCE
NOTE: Entry LOAN.PMENU has been imported.
NOTE: Entry LOAN.SCL has been imported.
NOTE: Total number of entries processed in catalog NEWLIB.FINANCE: 2
```

Example 3: Importing a Single Indexed SAS Data Set

Features: PROC CIMPORT statement option
INFILE=

Details

This example shows how to use PROC CIMPORT to import an indexed SAS data set from a transport file that was created by PROC CPORT from a single SAS data set.

Program

```
libname newdata 'sas-library';
filename trans3 'transport-file';

host-options-for-file-characteristics;

proc cimport data=newdata.times infile=trans3;
run;
```

Program Description

Specify the library name, filename, and operating environment options. The LIBNAME statement specifies a LIBNAME for the new SAS library. The FILENAME statement specifies the filename of the transport file that PROC CPORT created and enables you to specify any operating environment options for file characteristics.

```
libname newdata 'sas-library';
filename trans3 'transport-file';
```

```
host-options-for-file-characteristics;
```

Import the SAS data set. PROC CIMPORT imports the single SAS data set that you identify with the DATA= specification in the PROC CIMPORT statement. PROC CIMPORT exported the data set NEWDATA.TIMES in the transport file TRANS3.

```
proc cimport data=newdata.times infile=trans3;
run;
```

Log Examples

Example Code 10.3 Importing a Single Indexed SAS Data Set

```
NOTE: Proc CIMPORT begins to create/update data set NEWDATA.TIMES
NOTE: The data set index x is defined.
NOTE: Data set contains 2 variables and 2 observations.
      Logical record length is 16
```

Example 4: Using PROC CIMPORT to Import a French Data Set into a UTF-8 SAS Session

Features:	PROC CIMPORT statement option DATA= FILE= PROC CIMPORT statement option DATA= INFILE= EXTENDVAR= EXTENDFORMAT=
-----------	---

Details

This example shows you how to import a French data set into a UTF-8 SAS session. The French data set has an 8-byte character variable 'a' with a character format '\$8.'. Using a SAS Viya platform deployment, we start a French SAS session where we use PROC CIMPORT to generate a transport file named 'myfile'.

In another SAS Viya platform window, we start a SAS session that uses UTF-8 encoding. We then import the transport file (non-UTF-8) into a UTF-8 SAS session using PROC CIMPORT. We first show that the imported data set is truncated, and we then show how to extend the variable length using the EXTENDVAR= option to avoid truncation.

Program

Use PROC CPORT to create a transport file using a French data set. From a SAS Viya platform deployment, bring up a default SAS session using the French language. In the following example, we are using a French data set where the data set has an 8-byte character variable 'a' (the value of 'a' is premiere with an accented e) with a character format set to '\$8.'. We use PROC CPORT to create a transport file. In this example, the encoding is wlatin-1, the default.

```
data test;
  a = 'première';
  format a $8.;
run;
proc cport data=test file='myfile';
run;
```

Output 10.1 French Data Set Used in PROC CPORT File

Obs	a
1	premièr

Import the transport file into a UTF-8 SAS session using PROC CIMPORT. From a separate SAS Viya platform session that is using UTF-8 encoding, import the transport file 'myfile' into the UTF-8 SAS session using PROC CIMPORT. Print the SAS data set and the PROC CONTENTS. Because it takes 2 bytes to store the accented 'e' in UTF-8 encoding, the value premiere needs 9 bytes to display the value. The format is \$8. so the value is truncated.

```
proc cimport data=test infile='myfile';
run;
proc print data=test;
run;
proc contents data=test;
run;
```

Output 10.2 CIMPORT a Non-UTF-8 Transport File to an Encoding Session That Is UTF-8 - Truncated

Obs	a
1	premier

The CONTENTS Procedure			
Data Set Name	WORK.TEST	Observations	1
Member Type	DATA	Variables	1
Engine	V9	Indexes	0
Created	04/21/2020 15:01:17	Observation Length	9
Last Modified	04/21/2020 15:01:17	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64, LINUX_POWER_64		
Encoding	utf-8 Unicode (UTF-8)		

Engine/Host Dependent Information	
Data Set Page Size	65536
Number of Data Set Pages	1
First Data Page	1
Max Obs per Page	7177
Obs in First Data Page	1
Number of Data Set Repairs	0
Filename	/viya-directory/test.sas7bdat
Release Created	V.0400M0
Host Created	Linux
Inode Number	105502840
Access Permission	rw-r--r--
Owner Name	UNKNOWN
File Size	128KB
File Size (bytes)	131072

Alphabetic List of Variables and Attributes				
#	Variable	Type	Len	Format
1	a	Char	9	\$8.

Extend the Variable Length to avoid truncation. Because there was truncation, we use the EXTENDVAR= option to extend the variable length and the format width (default for EXTENDFORMAT= is YES).

```
proc cimport data=test infile='myfile' extendvar=1.5;
run;
proc print data=test;
run;
proc contents data=test;
run;
```

Note that the output data set now contains the entire variable. Also, note in the output of PROC CONTENTS that the variable length and the format width have been extended.

Output 10.3 CIMPORT a Non-UTF-8 Transport File to an Encoding Session That Is UTF-8 - EXTENDVAR=

Obs	a
1	première

The CONTENTS Procedure

Data Set Name	WORK.TEST	Observations	1
Member Type	DATA	Variables	1
Engine	V9	Indexes	0
Created	04/21/2020 15:23:51	Observation Length	14
Last Modified	04/21/2020 15:23:51	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64, LINUX_POWER_64		
Encoding	utf-8 Unicode (UTF-8)		

Engine/Host Dependent Information	
Data Set Page Size	65536
Number of Data Set Pages	1
First Data Page	1
Max Obs per Page	4636
Obs in First Data Page	1
Number of Data Set Repairs	0
Filename	/viya-directory/test.sas7bdat
Release Created	V.0400M0
Host Created	Linux
Inode Number	105502841
Access Permission	rw-r--r--
Owner Name	UNKNOWN
File Size	128KB
File Size (bytes)	131072

Alphabetic List of Variables and Attributes				
#	Variable	Type	Len	Format
1	a	Char	14	\$12.

COMPARE Procedure

Overview: COMPARE Procedure	209
What Does the COMPARE Procedure Do?	210
What Information Does PROC COMPARE Provide?	210
Concepts: COMPARE Procedure	212
Comparisons Using PROC COMPARE	212
A Comparison by Position of Observations	212
A Comparison with an ID Variable	213
The Equality Criterion	214
How PROC COMPARE Handles Variable Formats	216
Syntax: COMPARE Procedure	216
PROC COMPARE Statement	217
ATTRIB Statement	228
BY Statement	229
FORMAT Statement	233
ID Statement	233
LABEL Statement	235
VAR Statement	237
WHERE Statement	238
WITH Statement	240
Usage: COMPARE Procedure	241
Customizing PROC COMPARE Output	241
Results: COMPARE Procedure	244
Results Reporting	244
SAS Log	245
Macro Return Codes (SYSINFO)	245
Procedure Output	248
ODS Table Names	258
Output Data Set (OUT=)	259
Output Statistics Data Set (OUTSTATS=)	261
Examples: COMPARE Procedure	262
Example 1: Producing a Complete Report of the Differences	262
Example 2: Comparing Variables in Different Data Sets	269
Example 3: Comparing a Variable Multiple Times	271
Example 4: Comparing Variables That Are in the Same Data Set	273
Example 5: Comparing Observations with an ID Variable	275

Example 6: Comparing Values of Observations Using an Output Data Set (OUT=)	281
Example 7: Creating an Output Data Set of Statistics (OUTSTATS=)	285

Overview: COMPARE Procedure

What Does the COMPARE Procedure Do?

The COMPARE procedure compares the contents of two SAS data sets, selected variables in different data sets, or variables within the same data set.

PROC COMPARE compares two data sets: the base data set and the comparison data set. The procedure determines matching variables and matching observations. Matching variables are variables with the same name or variables that you pair by using the VAR and WITH statements. Matching variables must be of the same type. Matching observations are observations that have the same values for all ID variables that you specify or, if you do not use the ID statement, that occur in the same position in the data sets. If you match observations by ID variables, then both data sets must be sorted by all ID variables.

What Information Does PROC COMPARE Provide?

PROC COMPARE generates the following information about the two data sets that are being compared:

- whether matching variables have different values
- whether one data set has more observations than the other
- what variables the two data sets have in common
- how many variables are in one data set but not in the other
- whether matching variables have different formats, labels, or types
- a comparison of the values of matching observations

Note: You can create a view that has two columns with the same variable name. If duplicate variable names exist in the view, PROC COMPARE cannot determine which column in the base data set should be compared to the compare data set. PROC COMPARE issues an error if it finds duplicate variable names.

Further, PROC COMPARE creates two types of output data sets that give detailed information about the differences between observations of variables that it is comparing.

The following example compares the data sets Proclib.One and Proclib.Two, which contain similar data about students:

```
data proclib.one(label='First Data Set');
  input student year $ state $ gr1 gr2;
  label year='Year of Birth';
  format gr1 4.1;
  datalines;
1000 1990 NC 85 87
1042 1991 MS 90 92
1095 1989 TN 78 92
1187 1990 MA 87 94
;

data proclib.two(label='Second Data Set');
  input student $ year $ state $ gr1
    gr2 major $;
  label state='Home State';
  format gr1 5.2;
  datalines;
1000 1990 NC 85 87 Math
1042 1991 MS 90 92 History
1095 1989 TN 78 92 Physics
1187 1990 MA 87 94 Music
1204 1991 NC 82 96 English
;
```

PROC COMPARE does not produce information about values that are the same in each comparison data set. It produces information about values that are different, not the same.

PROC COMPARE does not produce a data set that contains observations that are in one of the comparison data sets but not in the other, or that are in both comparison data sets. The options for the COMPARE statement can produce much of this information, but they do not produce a data set. If you want to produce a data set that contains this information, use a DATA step that contains a MERGE statement. Here is an example of a DATA step that uses a MERGE statement to create a data set:

```
data inone intwo inboth;
  merge a (in=ina) b(in=inb);
  by byvar;
  if ina and not inb then output inone;
  if inb and not ina then output intwo;
  if ina and inb then output inboth;
run;
```

Concepts: COMPARE Procedure

Comparisons Using PROC COMPARE

PROC COMPARE first compares the following:

- data set attributes (set by the data set options TYPE= and LABEL=).
- variables. PROC COMPARE checks each variable in one data set to determine whether it matches a variable in the other data set.
- attributes (type, length, labels, formats, and informats) of matching variables.
- observations. PROC COMPARE checks each observation in one data set to determine whether it matches an observation in the other data set. PROC COMPARE either matches observations by their position in the data sets or by the values of the ID variable.

After making these comparisons, PROC COMPARE compares the values in the parts of the data sets that match. PROC COMPARE either compares the data by the position of observations or by the values of an ID variable.

A Comparison by Position of Observations

The following figure shows two data sets. The data inside the shaded boxes shows the part of the data sets that the procedure compares. Assume that variables with the same names have the same type.

Figure 11.1 Comparison by the Positions of Observations

Data Set ONE						
Obs	idnum	name	year	state	grade1	grade2
1	1000	Emily	1990	NC	85.0	87
2	1042	Dan	1991	MS	90.0	92
3	1095	Julia	1989	TN	78.0	92
4	1187	Robin	1990	MA	87.0	94

Data Set TWO							
Obs	idnum	name	year	state	grade1	grade2	major
1	1000	Emily	1990	NC	85.00	87	Math
2	1042	Dan	1991	MS	90.00	92	History
3	1095	Julia	1989	TN	78.00	92	Physics
4	1187	Robin	1990	MA	87.00	94	Music
5	1204	Keith	1991	NC	82.00	96	English
6	1198	Ted	1990	PA	84.00	93	Music
7	1054	Lisa	1989	GA	81.00	94	French

When you use PROC COMPARE to compare data set TWO with data set ONE, the procedure compares the first observation in data set ONE with the first observation in data set TWO, and it compares the second observation in the first data set with the second observation in the second data set, and so on. In each observation that it compares, the procedure compares the values of the idnum, name, year, state, grade1, and grade2.

The procedure does not report on the value of the last three observations or the variable major in data set TWO because there is nothing to compare them with in data set ONE.

A Comparison with an ID Variable

In a simple comparison, PROC COMPARE uses the observation number to determine which observations to compare. When you use an ID variable, PROC COMPARE uses the values of the ID variable to determine which observations to compare. ID variables should have unique values and must have the same type.

For the two data sets shown in the following figure, assume that IDNUM is an ID variable and that IDNUM has the same type in both data sets. The procedure compares the observations that have the same value for IDNUM. The data inside the shaded boxes shows the part of the data sets that the procedure compares.

Figure 11.2 Comparison by the Value of the ID Variable

Data Set ONE						
Obs	idnum	name	year	state	grade1	grade2
1	1000	Emily	1990	NC	85.0	87
2	1042	Dan	1991	MS	90.0	92
3	1095	Julia	1989	TN	78.0	92
4	1187	Robin	1990	MA	87.0	94

Data Set TWO							
Obs	idnum	name	year	state	grade1	grade2	major
1	1000	Emily	1990	NC	85.00	87	Math
2	1042	Dan	1991	MS	90.00	92	History
3	1095	Julia	1989	TN	78.00	92	Physics
4	1187	Robin	1990	MA	87.00	94	Music
5	1204	Keith	1991	NC	82.00	96	English
6	1198	Ted	1990	PA	84.00	93	Music
7	1054	Lisa	1989	GA	81.00	94	French

The data sets contain five matching variables: name, year, state, grade1, and grade2. They also contain four matching observations: the observations with values of 1000, 1042, 1095, and 1187 for idnum.

Data set TWO contains three observations (idnum=1204, idnum=1198, and idnum=1054) for which data set ONE contains no matching observations. Similarly, no variable in data set ONE matches the variable “major” in data set TWO.

See [“Example 5: Comparing Observations with an ID Variable” on page 275](#) for an example that uses an ID variable.

The Equality Criterion

Using the CRITERION= Option

The COMPARE procedure judges numeric values unequal if the magnitude of their difference, as measured according to the METHOD= option, is greater than the value of the CRITERION= option. PROC COMPARE provides four methods for applying CRITERION=:

- The EXACT method tests for exact equality.

- The ABSOLUTE method compares the absolute difference to the value specified by CRITERION=.
- The RELATIVE method compares the absolute relative difference to the value specified by CRITERION=.
- The PERCENT method compares the absolute percent difference to the value specified by CRITERION=.

For a numeric variable compared, let x be its value in the base data set and let y be its value in the comparison data set. If both x and y are nonmissing, then the values are judged unequal according to the value of METHOD= and the value of CRITERION= (γ) as follows:

- If METHOD=EXACT, then the values are unequal if y does not equal x .
- If METHOD=ABSOLUTE, then the values are unequal if

$$ABS(y - x) > \gamma$$

- If METHOD=RELATIVE, then the values are unequal if

$$ABS(y - x) / ((ABS(x) + ABS(y)) / 2 + \delta) > \gamma$$

The values are equal if $x=y=0$.

- If METHOD=PERCENT, then the values are unequal if

$$100(ABS(y - x) / ABS(x)) > \gamma \text{ for } x \neq 0$$

or

$$y \neq 0 \text{ for } x = 0$$

If x or y is missing, then the comparison depends on the NOMISSING option. If the NOMISSING option is in effect, then a missing value will always be judged equal to anything. Otherwise, a missing value is judged equal only to a missing value of the same type (that is, .=. , ^=.A, .A=.A, .A^=.B, and so on).

If the value that is specified for CRITERION= is negative, then the actual criterion that is used, γ , is equal to the absolute value of the specified criterion multiplied by a very small number, ϵ (epsilon), that depends on the numerical precision of the computer. This number ϵ is defined as the smallest positive floating-point value such that, using machine arithmetic, $1 - \epsilon < 1 + \epsilon$. Round-off or truncation error in floating-point computations is typically a few orders of magnitude larger than ϵ . CRITERION=-1000 often provides a reasonable test of the equality of computed results at the machine level of precision.

The value δ added to the denominator in the RELATIVE method is specified in parentheses after the method name: METHOD=RELATIVE(δ). If not specified in METHOD=, then δ defaults to 0. The value of δ can be used to control the behavior of the error measure when both x and y are very close to 0. If δ is not given and x and y are very close to 0, then any error produces a large relative error (in the limit, 2).

Specifying a value for δ avoids this extreme sensitivity of the RELATIVE method for small values. If you specify METHOD=RELATIVE(δ) CRITERION= γ when both x and y are much smaller than δ in absolute value, then the comparison is as if you had specified METHOD=ABSOLUTE CRITERION= $\delta\gamma$. However, when either x or y is much larger than δ in absolute value, the comparison is like METHOD=RELATIVE

CRITERION=y. For moderate values of x and y , METHOD=RELATIVE(δ)
 CRITERION=y is, in effect, a compromise between METHOD=ABSOLUTE
 CRITERION= δ y and METHOD=RELATIVE CRITERION=y.

For character variables, if one value is longer than the other, then the shorter value is padded with blanks for the comparison. Nonblank character values are judged equal only if they agree at each character. If the NOMISSING option is in effect, then blank character values are judged equal to anything.

Definition of Difference and Percent Difference

In the reports of value comparisons and in the OUT= data set, PROC COMPARE displays difference and percent difference values for the numbers compared. These quantities are defined using the value from the base data set as the reference value. For a numeric variable compared, let x be its value in the base data set and let y be its value in the comparison data set. If x and y are both nonmissing, then the difference and percent difference are defined as follows:

- Difference = $y - x$
- Percent Difference = $(y - x)/x * 100$ for $x \neq 0$
- Percent Difference = missing for $x = 0$.

How PROC COMPARE Handles Variable Formats

PROC COMPARE compares unformatted values. If you have two matching variables that are formatted differently, then PROC COMPARE lists the formats of the variables. If non-matching values are found in the comparison section of the output, then the formatted values are displayed in the values comparison section.

Syntax: COMPARE Procedure

Restriction: You must use the VAR statement when you use the WITH statement.

Tips: Each password and encryption key option must be coded on a separate line to ensure that they are properly blotted in the log.

You can use the LABEL, ATTRIB, FORMAT, and WHERE statements. For more information, see [“Statements with the Same Function in Multiple Procedures” on page 23](#).

Ensure that the LINESIZE option in the OPTIONS statement specifies an adequate length to display the label information for the data sets.

If problems occur with the length of CHAR variables when you compare two unequal data sets in CAS, you might need to change the value of the

NCHARMULTIPLIER option of the LIBNAME statement. For more information, see “NCHARMULTIPLIER= LIBNAME Statement Option” in *SAS Cloud Analytic Services: User’s Guide* and “LIBNAME Statement: CAS Engine” in *SAS Cloud Analytic Services: User’s Guide*.

```
PROC COMPARE <options>;
  ATTRIB variable-list(s) attribute-list(s);
  BY <DESCENDING> variable-1
    <... <DESCENDING> variable-n>
    <NOTSORTED>;
  FORMAT variable-1 <...variable-n> <format> <DEFAULT=default-format>;
  ID <DESCENDING> variable-1
    <<DESCENDING> variable-2 ...>
    <NOTSORTED>;
  LABEL variable-1=label-1<...variable-n=label-n>;
  VAR variable(s);
  WHERE where-expression-1
    < logical-operator where-expression-n>;
  WITH variable(s);
```

Statement	Task	Example
PROC COMPARE	Compare the contents of SAS data sets, or compare two variables	Ex. 1, Ex. 2, Ex. 4, Ex. 6, Ex. 7
BY	Produce a separate comparison for each BY group	
ID	Identify variables to use to match observations	Ex. 5
VAR	Restrict the comparison to values of specific variables	Ex. 2, Ex. 3, Ex. 4
WITH	Compare variables of different names	Ex. 2, Ex. 3, Ex. 4
WITH	Compare two variables in the same data set	Ex. 4

PROC COMPARE Statement

Compares the contents of two SAS data sets, selected variables in different data sets, or variables within the same data set.

Restriction: If you omit COMPARE=, then you must use the WITH and VAR statements.

Tips: Ensure that the LINESIZE option in the OPTIONS statement specifies an adequate length to display the label information for the data sets.

If problems occur with the length of CHAR variables when you compare two unequal data sets in CAS, you might need to change the value of the NCHARMULTIPLIER option of the LIBNAME statement. For more information, see [“NCHARMULTIPLIER= LIBNAME Statement Option” in SAS Cloud Analytic Services: User’s Guide](#) and [“LIBNAME Statement: CAS Engine” in SAS Cloud Analytic Services: User’s Guide](#).

You can use data set options with the BASE= and COMPARE= options.

Examples:

[“Example 1: Producing a Complete Report of the Differences” on page 262](#)

[“Example 2: Comparing Variables in Different Data Sets” on page 269](#)

[“Example 4: Comparing Variables That Are in the Same Data Set” on page 273](#)

[“Example 6: Comparing Values of Observations Using an Output Data Set \(OUT=\)” on page 281](#)

[“Example 7: Creating an Output Data Set of Statistics \(OUTSTATS=\)” on page 285](#)

Syntax

PROC COMPARE *<options>*;

Summary of Optional Arguments

Control the details in the default report

ALLOBS

includes the values for all matching observations.

ALLSTATS

prints a table of summary statistics for all pairs of matching variables.

ALLVARS

includes in the report the values and differences for all matching variables.

BRIEFSUMMARY

prints only a short comparison summary.

FUZZ=*number*

changes the report for numbers between 0 and 1.

MAXPRINT=*total* | (*per-variable*, *total*)

restricts the number of differences to be printed.

NODATE

suppresses the printing of creation and last-modified dates.

NOPRINT

suppresses all printed output.

NOSUMMARY

suppresses the data set, variable, observation, and values comparison summary reports.

NOVALUES

suppresses the report of the value comparison results.

PRINTALL

produces a complete listing of values and differences.

STATS

prints a table of summary statistics for all pairs of matching numeric variables that are judged unequal.

TRANPOSE

prints the reports of value differences by observation instead of by variable.

Control the listing of variables and observations

LISTALL

lists all variables and observations that are found in only one data set.

LISTBASE

lists all variables and observations found only in the base data set.

LISTBASEOBS

lists all observations found only in the base data set.

LISTBASEVAR

lists all variables found in only the base data set.

LISTCOMP

lists all variables and observations found only in the comparison data set.

LISTCOMPOBS

lists all observations found only in the comparison data set.

LISTCOMPVAR

lists all variables found only in the comparison data set.

LISTEQUALVAR

lists variables whose values are judged equal.

LISTOBS

lists all observations found in only one data set.

LISTVAR

list all variables found in only one data set.

Control the output data set

OUT=SAS-data-set

creates an output data set.

OUTALL

writes an observation for each observation in the BASE= and COMPARE= data sets.

OUTBASE

writes an observation for each observation in the base data set.

OUTCOMP

writes an observation for each observation in the comparison data set

OUTDIF

writes an observation to the output data set for each pair of matching observations.

OUTNOEQUAL

suppresses the writing of observations when all values are equal.

OUTPERCENT

writes an observation to the output data set for each pair of matching observations.

Create an output data set that contains summary statistics**OUTSTATS=SAS-data-set**

writes summary statistics for all pairs of matching variables to the specified SAS-data-set.

Display a warning message in the SAS log**WARNING**

displays a warning message in the SAS log when differences are found.

Display an error message in the SAS log**ERROR**

displays an error message in the SAS log when differences are found.

Specify how the values are compared**CRITERION=y**

specifies the criterion for judging the equality of numeric values.

METHOD=ABSOLUTE | EXACT | PERCENT | RELATIVE<(δ)>

specifies the method for judging the equality of numeric values.

NOMISSBASE

judges a missing value in the base data set equal to any value.

NOMISSCOMP

judges a missing value in the comparison data set equal to any value.

NOMISSING

judges missing values in both the base and comparison data sets equal to any value.

Specify the data sets to compare**BASE=SAS-data-set**

specifies the data set to use as the base data set.

COMPARE=SAS-data-set

specifies the data set to use as the comparison data set.

Write notes to the SAS log**NOTE**

displays notes in the SAS log that describe the results of the comparison.

Optional Arguments**ALLOBS**

includes in the report of value comparison results the values and, for numeric variables, the differences for all matching observations, even if they are judged equal.

Default If you omit ALLOBS, then PROC COMPARE prints values only for observations that are judged unequal.

Interaction When used with the TRANSPOSE option, ALLOBS invokes the ALLVARS option and displays the values for all matching observations and variables.

ALLSTATS

prints a table of summary statistics for all pairs of matching variables.

See [“Table of Summary Statistics” on page 255](#) for information about the statistics produced

ALLVARS

includes in the report of value comparison results the values and, for numeric variables, the differences for all pairs of matching variables, even if they are judged equal.

Default If you omit ALLVARS, then PROC COMPARE prints values only for variables that are judged unequal.

Interaction When used with the TRANSPOSE option, ALLVARS displays unequal values in context with the values for other matching variables. If you omit the TRANSPOSE option, then ALLVARS invokes the ALLOBS option and displays the values for all matching observations and variables.

BASE=SAS-data-set

specifies the data set to use as the base data set.

Aliases DATA=

B=

Default the most recently created SAS data set

Tip You can use the WHERE= data set option with the BASE= option to limit the observations that are available for comparison.

BRIEFSUMMARY

produces a short comparison summary and suppresses the four default summary reports (data set summary report, variables summary report, observation summary report, and values comparison summary report).

Alias BRIEF

Tip By default, a listing of value differences accompanies the summary reports. To suppress this listing, use the NOVALUES option.

Example [“Example 4: Comparing Variables That Are in the Same Data Set” on page 273](#)

COMPARE=SAS-data-set

specifies the data set to use as the comparison data set.

Aliases COMP=

C=

Default If you omit COMPARE=, then the comparison data set is the same as the base data set, and PROC COMPARE compares variables within the data set.

Restriction If you omit COMPARE=, then you must use the WITH and VAR statements.

Tip You can use the WHERE= data set option with COMPARE= to limit the observations that are available for comparison.

CRITERION= γ

specifies the criterion for judging the equality of numeric values. Normally, the value of γ (gamma) is positive. In that case, the number itself becomes the equality criterion. If you use a negative value for γ , then PROC COMPARE uses an equality criterion proportional to the precision of the computer on which SAS is running.

Aliases CRITERIA=
CRIT=

Default 0.00001

See [“The Equality Criterion” on page 214](#)

ERROR

displays an error message in the SAS log when differences are found.

Interaction This option overrides the WARNING option.

FUZZ=*number*

alters the values comparison results for numbers less than *number*. PROC COMPARE prints the following:

- 0 for any variable value that is less than *number*
- a blank for difference or percent difference if it is less than *number*
- 0 for any summary statistic that is less than *number*

Default 0

Range 0 - 1

Tip A report that contains many trivial differences is easier to read in this form.

LISTALL

lists all variables and observations that are found in only one data set.

Alias LIST

Interaction using LISTALL is equivalent to using the following four options: LISTBASEOBS, LISTCOMPOBS, LISTBASEVAR, and LISTCOMPVAR.

LISTBASE

lists all observations and variables that are found in the base data set but not in the comparison data set.

Interaction Using LISTBASE is equivalent to using the LISTBASEOBS and LISTBASEVAR options.

LISTBASEOBS

lists all observations that are found in the base data set but not in the comparison data set.

LISTBASEVAR

lists all variables that are found in the base data set but not in the comparison data set.

LISTCOMP

lists all observations and variables that are found in the comparison data set but not in the base data set.

Alias LISTCOMPARE

Interaction Using LISTCOMP is equivalent to using the LISTCOMPOBS and LISTCOMPVAR options.

LISTCOMPOBS

lists all observations that are found in the comparison data set but not in the base data set.

Alias LISTCOMPAREOBS

LISTCOMPVAR

lists all variables that are found in the comparison data set but not in the base data set.

Aliases LISTCOMPAREVARS

LISTCOMPAREVAR

LISTEQUALVAR

prints a list of variables whose values are judged equal at all observations in addition to the default list of variables whose values are judged unequal.

Aliases LISTEQ

LISTEQUAL

LISTEQVAR

LISTOBS

lists all observations that are found in only one data set.

Interaction Using LISTOBS is equivalent to using the LISTBASEOBS and LISTCOMPOBS options.

LISTVAR

lists all variables that are found in only one data set.

Interaction Using LISTVAR is equivalent to using both the LISTBASEVAR and LISTCOMPVAR options.

MAXPRINT=total | (per-variable, total)

specifies the maximum number of differences to be printed, where

total

is the maximum total number of differences to be printed. The default value is 500 unless you use the ALLOBS option (or both the ALLVAR and TRANSPOSE options). In that case, the default is 32000.

per-variable

is the maximum number of differences to be printed for each variable within a BY group. The default value is 50 unless you use the ALLOBS option (or both the ALLVAR and TRANSPOSE options). In that case, the default is 1000.

The MAXPRINT= option prevents the output from becoming extremely large when data sets differ greatly.

METHOD=ABSOLUTE | EXACT | PERCENT | RELATIVE<(δ)>

specifies the method for judging the equality of numeric values. The constant δ (delta) is a number between 0 and 1 that specifies a value to add to the denominator when calculating the equality measure. By default, δ is 0.

Unless you use the CRITERION= option, the default method is EXACT. If you use the CRITERION= option, then the default method is RELATIVE(ϕ), where ϕ (phi) is a small number that depends on the numerical precision of the computer on which SAS is running and on the value of CRITERION=.

Aliases M=

METH=

See [“The Equality Criterion” on page 214](#)

NODATE

suppresses the display in the data set summary report of the creation dates and the last modified dates of the base and comparison data sets.

NOMISSBASE

(By default, a missing value is equal only to a missing value of the same kind, that is $.=.$, $.^{.}=A$, $.A=.A$, $.A^{.}=B$, and so on.)

You can use this option to determine the changes that would be made to the observations in the comparison data set if it were used as the main data set and the base data set were used as the transaction data set in a DATA step UPDATE statement. For information about the UPDATE statement, see [“UPDATE Statement” in SAS DATA Step Statements: Reference](#).

Alias NOMISS1

NOMISSCOMP

judges a missing value in the comparison data set equal to any value. (By default, a missing value is equal only to a missing value of the same kind, that is $.=.$, $.^{.}=A$, $.A=.A$, $.A^{.}=B$, and so on.)

You can use this option to determine the changes that would be made to the observations in the base data set if it were used as the main data set and the comparison data set were used as the transaction data set in a DATA step

UPDATE statement. For information about the UPDATE statement, see [“UPDATE Statement” in SAS DATA Step Statements: Reference](#).

Aliases NOMISS2

NOMISSCOMPARE

NOMISSING

judges missing values in both the base and comparison data sets equal to any value. By default, a missing value is equal only to a missing value of the same kind, that is `.=.`, `.^=.A`, `.A=.A`, `.A^=.B`, and so on.

Alias NOMISS

Interaction Using NOMISSING is equivalent to using both NOMISSBASE and NOMISSCOMP.

NOPRINT

suppresses all printed output.

Tip You might want to use this option when you are creating one or more output data sets.

Example [“Example 6: Comparing Values of Observations Using an Output Data Set \(OUT=\)” on page 281](#)

NOSUMMARY

suppresses the data set, variable, observation, and values comparison summary reports.

Alias NOSUM

Tip NOSUMMARY produces no output if there are no differences in the matching values.

Example [“Example 2: Comparing Variables in Different Data Sets” on page 269](#)

NOTE

displays notes in the SAS log that describe the results of the comparison, if differences were found.

NOVALUES

suppresses the report of the value comparison results.

Alias NOOBS

Example [Overview: COMPARE Procedure on page 210](#)

OUT=SAS-data-set

names the output data set. If *SAS-data-set* does not exist, then PROC COMPARE creates it. *SAS-data-set* contains the differences between matching variables.

See [“Output Data Set \(OUT=\)” on page 259](#)

Example [“Example 6: Comparing Values of Observations Using an Output Data Set \(OUT=\)” on page 281](#)

OUTALL

writes an observation to the output data set for each observation in the base data set and for each observation in the comparison data set. The option also writes observations to the output data set that contains the differences and percent differences between the values in matching observations.

Tip Using OUTALL is equivalent to using the following four options: OUTBASE, OUTCOMP, OUTDIF, and OUTPERCENT.

See [“Output Data Set \(OUT=\)” on page 259](#)

OUTBASE

writes an observation to the output data set for each observation in the base data set, creating observations in which `_TYPE_=BASE`.

See [“Output Data Set \(OUT=\)” on page 259](#)

Example [“Example 6: Comparing Values of Observations Using an Output Data Set \(OUT=\)” on page 281](#)

OUTCOMP

writes an observation to the output data set for each observation in the comparison data set, creating observations in which `_TYPE_=COMP`.

Alias OUTCOMPARE

See [“Output Data Set \(OUT=\)” on page 259](#)

Example [“Example 6: Comparing Values of Observations Using an Output Data Set \(OUT=\)” on page 281](#)

OUTDIF

writes an observation to the output data set for each pair of matching observations. The values in the observation include values for the differences between the values in the pair of observations. The value of `_TYPE_` in each observation is DIF.

Alias OUTDIFF

Default The OUTDIF option is the default unless you specify the OUTBASE, OUTCOMP, or OUTPERCENT option. If you use any of these options, then you must specify the OUTDIF option to create `_TYPE_=DIF` observations in the output data set.

See [“Output Data Set \(OUT=\)” on page 259](#)

Example [“Example 6: Comparing Values of Observations Using an Output Data Set \(OUT=\)” on page 281](#)

OUTNOEQUAL

suppresses the writing of an observation to the output data set when all values in the observation are judged equal. In addition, in observations containing

values for some variables judged equal and others judged unequal, the OUTNOEQUAL option uses the special missing value ".E" to represent differences and percent differences for variables judged equal.

Alias OUTNOEQ

See [“Output Data Set \(OUT=\)” on page 259](#)

Example [“Example 6: Comparing Values of Observations Using an Output Data Set \(OUT=\)” on page 281](#)

OUTPERCENT

writes an observation to the output data set for each pair of matching observations. The values in the observation include values for the percent differences between the values in the pair of observations. The value of `_TYPE_` in each observation is PERCENT.

See [“Output Data Set \(OUT=\)” on page 259](#)

OUTSTATS=SAS-data-set

writes summary statistics for all pairs of matching variables to the specified *SAS-data-set*.

Tip If you want to print a table of statistics in the procedure output, then use the STATS, ALLSTATS, or PRINTALL option.

See [“Output Statistics Data Set \(OUTSTATS=\)” on page 261](#)

[“Table of Summary Statistics” on page 255](#)

Example [“Example 7: Creating an Output Data Set of Statistics \(OUTSTATS=\)” on page 285](#)

PRINTALL

invokes the following options: ALLVARS, ALLOBS, ALLSTATS, LISTALL, and WARNING.

Alias ALL

Example [“Example 1: Producing a Complete Report of the Differences” on page 262](#)

STATS

prints a table of summary statistics for all pairs of matching numeric variables that are judged unequal.

Alias STATISTICS

See [“Table of Summary Statistics” on page 255](#) for information about the statistics produced.

TRANPOSE

prints the reports of value differences by observation instead of by variable.

Alias TRANS

- Interaction If you also use the NOVALUES option, then the TRANSPOSE option lists only the *names* of the variables whose values are judged unequal for each observation, not the values and differences.
- See [“Comparison Results for Observations \(Using the TRANSPOSE Option\)” on page 257.](#)

WARNING

displays a warning message in the SAS log when differences are found.

Alias WARN

Interaction The ERROR option overrides the WARNING option.

ATTRIB Statement

Associates a format, informat, label, and length with one or more variables.

Syntax

ATTRIB *variable-list(s) attribute-list(s);*

Arguments

variable-list(s)

names the variables that you want to associate with the attributes.

TIP List the variables in any form that SAS allows.

attribute-list(s)

specifies one or more attributes to assign to *variable-list*. Specify one or more of these attributes in the ATTRIB statement:

FORMAT=*format*

associates a format with variables in *variable-list*.

Tip The format can be either a standard SAS format or a format that is defined with the FORMAT procedure.

INFORMAT=*informat*

associates an informat with variables in *variable-list*.

Tip The informat can be either a standard SAS informat or an informat that is defined with the FORMAT procedure.

LABEL='label'

associates a label with variables in *variable-list*.

See Also

[“ATTRIB Statement” in SAS DATA Step Statements: Reference.](#)

BY Statement

Orders the output according to the BY groups.

Restriction: Do not specify a group variable in the BY statement. The BY variable cannot be the same as the group variable. A *group variable* is a variable in the input data set that is used to categorize chart variable values into groups. For more information, see [“Group Variables” on page 1760](#).

Syntax

```
BY <DESCENDING> variable-1  
    <... <DESCENDING> variable-n>  
    <NOTSORTED>;
```

Required Argument

variable

specifies the variable that the procedure uses to form BY groups. You can specify more than one variable. If you do not use the NOTSORTED option in the BY statement, then either the observations in the data set must be sorted by all the variables that you specify, or they must be indexed appropriately. Variables in a BY statement are called *BY variables*.

Optional Arguments

DESCENDING

specifies that the observations are sorted in descending order by the variable that immediately follows the word DESCENDING in the BY statement.

NOTSORTED

specifies that observations are not necessarily sorted in alphabetic or numeric order. The observations are grouped in another way (for example, chronological order).

The requirement for ordering or indexing observations according to the values of BY variables is suspended for BY-group processing when you use the NOTSORTED option. In fact, the procedure does not use an index if you specify

NOTSORTED. The procedure defines a BY group as a set of contiguous observations that have the same values for all BY variables. If observations with the same values for the BY variables are not contiguous, then the procedure treats each contiguous set as a separate BY group.

You cannot use the NOTSORTED option in a PROC SORT step.

Note: You cannot use the GROUPFORMAT option, which is available in the BY statement in a DATA step, in a BY statement in any PROC step.

Details

BY-Group Processing

Procedures create output for each BY group. For example, the elementary statistics procedures and the scoring procedures perform separate analyses for each BY group. The reporting and ODS procedures produce a report for each BY group.

Note: All Base SAS procedures except PROC PRINT process BY groups independently. PROC PRINT can report the number of observations in each BY group as well as the number of observations in all BY groups. Similarly, PROC PRINT can sum numeric variables in each BY group and across all BY groups.

You can use only one BY statement in each PROC step. When you use a BY statement, the procedure expects an input data set that is sorted by the order of the BY variables or one that has an appropriate index. If your input data set does not meet these criteria, then an error occurs. Either sort it with the SORT procedure or create an appropriate index on the BY variables.

Depending on the order of your data, you might need to use the NOTSORTED or DESCENDING option in the BY statement in the PROC step.

- For more information about the BY statement, see [SAS DATA Step Statements: Reference](#).
- For more information about PROC SORT, see [Chapter 55, “SORT Procedure,” on page 2039](#).
- For more information about BY-group processing in CAS, see [“Grouping and Ordering” in SAS Cloud Analytic Services: DATA Step Programming](#).
- For more information about creating indexes, see [“INDEX CREATE Statement” on page 440](#).

Formatting BY-Variable Values

When a procedure is submitted with a BY statement, the following actions are taken with respect to processing of BY groups:

- 1 The procedure determines whether the data is sorted by the internal (unformatted) values of the BY variable(s).

- 2 The procedure determines whether a format has been applied to the BY variable(s). If the BY variable is numeric and has no user-applied format, then the BEST12. format is applied for the purpose of BY-group processing.
- 3 The procedure continues adding observations to the current BY group until both the internal and formatted values of the BY variable or variables change.

This process can have unexpected results if, for example, nonconsecutive internal BY values share the same formatted value. In this case, the formatted value is represented in different BY groups. Alternatively, if different consecutive internal BY values share the same formatted value, then these observations are grouped into the same BY group.

BY Variables That Have Different Lengths in Two Data Sets

When a procedure has a BY statement and two input data sets, one of the input data sets is called the primary data set and the other is called the secondary data set. The primary data set is usually, but not always, the DATA= data set. A BY statement always applies to the primary data set. The variables in the BY statement must appear in the primary data set.

Each procedure determines whether a BY statement applies to a secondary data set, and performs one of the following actions:

- The procedure might always apply the BY statement to the secondary data set. In this case, one or more variables in the BY statement must appear in the secondary data set.
- The procedure might never apply the BY statement to the secondary data set. In this case, the variables in the BY statement are not required in the secondary data set.
- The procedure might check whether the BY variables are in the secondary data set. If none of the BY variables are in the secondary data set, then BY processing does not occur for the secondary data set. If one or more of the BY variables are in the secondary data set, and they match the BY variables in the primary data set, then BY processing is done for the secondary data set. If some but not all BY variables are in the secondary data set, then the procedure might issue an error message and quit. Or, it might take some other action described in the documentation for that particular procedure.

If the BY statement is applied to the secondary data set, then each BY variable that exists on both the data sets must have the same type, character or numeric, in both data sets. The BY variables are required to have either the same formatted value or the same unformatted value. Formatted values match only if both the formatted lengths and the formatted values are the same. Unformatted values are not required to have the same length in order to match. The unformatted character values match if the unformatted values are the same after stripping the trailing blanks. The unformatted doubles match if they have the same value.

A secondary data set does not need to have all of the BY variables that are in the primary data set. A procedure can define a subset of the BY variables for the secondary data set. For example, if the primary data set has the BY variables A,B,C,D, then the procedure can define the following BY variables on the secondary data set:

- A
- A,B
- A,B,C
- A,B,C,D

If both the primary and secondary data sets have the same number of BY variables, and all the BY variables have the same byte lengths and format lengths, then either the unformatted values or the formatted values in the BY buffer (for all of the BY variables) have to match. If they do not match, then each variable is compared. The formatted values of each variable are compared first. The formatted lengths have to match, and the formatted values have to match. If the formatted lengths and values do not match, then the unformatted values are compared even if the byte lengths are different.

If corresponding character variable lengths differ, then the longer character variable can contain only trailing blanks for the extra characters. If the lengths of the character variables are different, then the values match as long as they are the same after stripping the trailing blanks. For example, 'ABCD' in the primary data set matches 'ABCD ' in the secondary data set. If the secondary data set contained 'ABCDEF', then they would not match.

Base SAS Procedures That Support the BY Statement

COMPARE	SORT (required)
CORR	STANDARD
FREQ	SUMMARY
MEANS	TABULATE
PRINT	TRANPOSE
RANK	UNIVARIATE
REPORT (nonwindowing environment only)	ODSTEXT
ODSLIST	ODSTABLE

Note: In the SORT procedure, the BY statement specifies how to sort the data. In the other procedures, the BY statement specifies how the data is currently sorted.

See Also

[“BY Statement” in SAS DATA Step Statements: Reference](#)

FORMAT Statement

Associates formats with variables.

Syntax

FORMAT *variable-1* <...*variable-n*> *format variable-1* <...*variable-n*> *format*;

Arguments

variable

names one or more variables for SAS to associate with a format. You must specify at least one *variable*.

Tip To disassociate a format from a variable, use the variable in a FORMAT statement without specifying a format in a DATA step or in PROC DATASETS. In a DATA step, place this FORMAT statement after the SET statement. See [“Removing a Format” in SAS DATA Step Statements: Reference](#). You can also use PROC DATASETS.

format

specifies the format that is listed for writing the values of the variables.

Tip Formats that are associated with variables by using a FORMAT statement behave like formats that are used with a colon modifier in a subsequent PUT statement. For more information about using a colon modifier, see [“PUT Statement: List” in SAS DATA Step Statements: Reference](#).

See [SAS Formats and Informats: Reference](#)

ID Statement

Lists variables to use to match observations.

See: [“A Comparison with an ID Variable” on page 213](#)

Example: [“Example 5: Comparing Observations with an ID Variable” on page 275](#)

Syntax

ID <DESCENDING> *variable-1*
<<DESCENDING> *variable-2* ...>

<NOTSORTED>;

Required Argument

variable

specifies the variable that the procedure uses to match observations. You can specify more than one variable, but the data set must be sorted by the variable or variables that you specify. These variables are *ID variables*. ID variables also identify observations on the printed reports and in the output data set.

Optional Arguments

DESCENDING

specifies that the data set is sorted in descending order by the variable that immediately follows the word DESCENDING in the ID statement.

If you use the DESCENDING option, then you must sort the data sets. SAS does not use an index to process an ID statement with the DESCENDING option. Further, the use of DESCENDING for ID variables must correspond to the use of the DESCENDING option in the BY statement in the PROC SORT step that was used to sort the data sets.

NOTSORTED

specifies that observations are not necessarily sorted in alphabetic or numeric order. The data are grouped in another way (for example, chronological order).

See [“Comparing Unsorted Data” on page 234](#)

Details

Requirements for ID Variables

- ID variables must be in the BASE= data set or PROC COMPARE stops processing.
- If an ID variable is not in the COMPARE= data set, then PROC COMPARE writes a warning message to the SAS log and does not use that variable to match observations in the comparison data set (but does write it to the OUT= data set).
- ID variables must be of the same type in both data sets.
- You should sort both data sets by the common ID variables (within the BY variables, if any) unless you specify the NOTSORTED option.

Comparing Unsorted Data

If you do not want to sort the data set by the ID variables, then you can use the NOTSORTED option. When you specify the NOTSORTED option, or if the ID statement is omitted, PROC COMPARE matches the observations one-to-one. That is, PROC COMPARE matches the first observation in the base data set with the first observation in the comparison data set, the second with the second, and so on. If

you use NOTSORTED, and the ID values of corresponding observations are not the same, then PROC COMPARE prints an error message and stops processing.

If the data sets are not sorted by the common ID variables and if you do not specify the NOTSORTED option, then PROC COMPARE writes a warning message to the SAS log and continues to process the data sets as if you had specified NOTSORTED.

Avoiding Duplicate ID Values

The observations in each data set should be uniquely labeled by the values of the ID variables. If PROC COMPARE finds two successive observations with the same ID values in a data set, then it does the following:

- prints the warning `Duplicate Observations` for the first occurrence for that data set
- prints the total number of duplicate observations found in the data set in the observation summary report
- uses the duplicate observations in the base data set and the comparison data set to compare the observations on a one-to-one basis

When the data sets are not sorted, PROC COMPARE detects only those duplicate observations that occur in succession.

LABEL Statement

Assigns descriptive labels to variables.

Syntax

LABEL *variable-1* = '*text-string*' <...*variable-n* = '*text-string*'>;

LABEL *variable-1* = '' <...*variable-n*=''>; | *variable-1* = <...*variable-n*=>;

Arguments

variable

specifies the variable that you want to label. You can specify labels for multiple variables in a single LABEL statement:

```
data test;
  L = 5;
  W = 10;
  label L = 'Length' W = 'Width';
run;
```

text-string

specifies a label of up to 256 bytes.

```
data test;
```

```
L = 5;
label L = 'Length';
run;
```

Restrictions If the label includes a semicolon (;) or an equal sign (=), you must enclose the label in either single or double quotation marks.

```
label N = 'Number = ';
```

If the label includes single quotation marks ('), you must enclose the label in double quotation marks.

```
label Name = "Owner's Last Name";
```

There is partial support for variable name substitution with the JAVAIMG device. When the label text for the variable exceeds the space allowed, the variable name is used instead by procedures that are labeling an axis or a legend title. With the exception of the SAS/GRAPH GPLOT procedure, the variable name is not used when the JAVAIMG device is specified.

Notes Quotation marks are not required unless the label includes a semicolon, quotation marks, or an equal sign.

The LABEL statement expects an equal sign (=) after the first variable. If any other character is found, an error occurs. The LABEL statement considers everything after the equal sign, following the first variable, as part of the label until the statement comes to another variable followed by an equal sign, regardless of quotation marks. In this example, the label for x is **this is x y 4 = this is y** because the next variable that is followed by an equal sign is z.

```
data test;
  x = 1;
  y = 1;
  z = 1;
  label x = 'this is x' y 4 = 'this is y' z = 'This is z';
run;
```

In this LABEL statement, the x variable has the same label:

```
label x = this is x y 4 = this is y z = This is z;
```

See [“Character Constants and Character Variables” in SAS Programmer’s Guide: Essentials.](#)

”

removes a label from a variable. Enclose a single blank space in quotation marks to remove an existing label. You can remove labels from multiple variables in a single LABEL statement:

```
data test;
  L=5; W=10;
  label L = ' ' W = ' ';
run;
```

Alternatively, you can remove a label by specifying nothing after the equal sign:

```
data test;
  L=5; W=10;
  label L = W = ;
run;
```

VAR Statement

Restricts the comparison of the values of variables to the ones named in the VAR statement.

Alias: VARIABLE

Examples: [“Example 2: Comparing Variables in Different Data Sets” on page 269](#)
[“Example 3: Comparing a Variable Multiple Times” on page 271](#)
[“Example 4: Comparing Variables That Are in the Same Data Set” on page 273](#)

Syntax

VAR *variable(s)*;

Required Argument

variable(s)

one or more variables that appear in the BASE= and COMPARE= data sets or only in the BASE= data set.

Details

- If you do not use the VAR statement, then PROC COMPARE compares the values of all matching variables except the ones that appear in BY and ID statements.
- If a variable in the VAR statement does not exist in the COMPARE= data set, then PROC COMPARE writes a warning message to the SAS log and ignores the variable.
- If a variable in the VAR statement does not exist in the BASE= data set, then PROC COMPARE stops processing and writes an error message to the SAS log.
- The VAR statement restricts only the comparison of values of matching variables. PROC COMPARE still reports on the total number of matching variables and compares their attributes. However, it produces neither error nor warning messages about these variables.

WHERE Statement

Subsets the input data set by specifying certain conditions that each observation must meet before it is available for processing.

Syntax

```
WHERE where-expression-1  
< logical-operator where-expression-n>;
```

Arguments

where-expression
is an arithmetic or logical expression that generally consists of a sequence of operands and operators.

Tips The operands and operators described in the next several sections are also valid for the WHERE= data set option.

You can specify multiple where-expressions.

logical-operator
can be AND, AND NOT, OR, or OR NOT.

Example

Procedures That Support the WHERE Statement

You can use the WHERE statement with any of the following Base SAS procedures that read a SAS data set:

COMPARE	REPORT
CORR	SORT
DATASETS (APPEND statement)	SQL
FREQ	STANDARD

MEANS or SUMMARY	TABULATE
PRINT	TRANSPOSE
RANK	UNIVARIATE

Details

- The COMPARE procedure and the APPEND statement in PROC DATASETS accept more than one input data set. For more information, see the documentation for the specific procedure.
- To subset the output data set, use the WHERE= data set option:

```
proc report data=debate nowd
              out=onlyfr(where=(year='1')) ;
run;
```

For more information about WHERE=, see [SAS DATA Step Statements: Reference](#).

Temporarily Dissociating a Format from a Variable

In this example, PROC PRINT prints only those observations that meet the condition of the WHERE expression.

```
options nodate pageno=1 linesize=64
              pagesize=40;

proc print data=debate noobs;
  where gpa>3.5;
  title 'Team Members with a GPA';
  title2 'Greater than 3.5';
run;
```

Output

**Team Members with a GPA
Greater than 3.5**

Name	Gender	Year	GPA
Capiccio	m	Freshman	3.598
Tucker	m	Freshman	3.901
Bagwell	f	Sophomore	3.722
Gold	f	Junior	3.609
Syme	f	Junior	3.883
Baglione	f	Senior	4.000
Carr	m	Senior	3.750
Hall	m	Senior	3.574

WITH Statement

Compares variables in the base data set with variables that have different names in the comparison data set, and compares different variables that are in the same data set.

Restriction: You must use the VAR statement when you use the WITH statement.

Examples: [“Example 2: Comparing Variables in Different Data Sets” on page 269](#)
[“Example 3: Comparing a Variable Multiple Times” on page 271](#)
[“Example 4: Comparing Variables That Are in the Same Data Set” on page 273](#)

Syntax

WITH *variable(s)*;

Required Argument

variable(s)

one or more variables to compare with variables in the VAR statement.

Details

Comparing Selected Variables

If you want to compare variables in the base data set with variables that have different names in the comparison data set, then specify the names of the variables in the base data set in the VAR statement and specify the names of the matching variables in the WITH statement. The first variable that you list in the WITH statement corresponds to the first variable that you list in the VAR statement, the second with the second, and so on. If the WITH statement list is shorter than the VAR statement list, then PROC COMPARE assumes that the extra variables in the VAR statement have the same names in the comparison data set as they do in the base data set. If the WITH statement list is longer than the VAR statement list, then PROC COMPARE ignores the extra variables.

A variable name can appear any number of times in the VAR statement or the WITH statement. By selecting VAR and WITH statement lists, you can compare the variables in any permutation.

If you omit the COMPARE= option in the PROC COMPARE statement, then you must use the WITH statement. In this case, PROC COMPARE compares the values of variables with different names in the BASE= data set.

Usage: COMPARE Procedure

Customizing PROC COMPARE Output

PROC COMPARE produces lengthy output. You can use one or more options to determine the types of comparisons to make and the degree of detail in the report. For example, in the following PROC COMPARE step, the NOVALUES option suppresses the part of the output that shows the differences in the values of matching variables:

```
options nodate pageno=1 linesize=80 pagesize=40;

title 'The SAS System';

proc compare base=proclib.one
             compare=proclib.two novalues;
run;
```

Output 11.1 Part One of Comparison of WORK.ONE and WORK.TWO**The SAS System**

The COMPARE Procedure
 Comparison of WORK.ONE with WORK.TWO
 (Method=EXACT)

Data Set Summary

Dataset	Created	Modified	NVar	NObs	Label
WORK.ONE	16MAR11:16:43:16	16MAR11:16:43:16	5	4	First Data Set
WORK.TWO	16MAR11:16:43:16	16MAR11:16:43:16	6	5	Second Data Set

Variables Summary

Number of Variables in Common: 5.
 Number of Variables in WORK.TWO but not in WORK.ONE: 1.
 Number of Variables with Conflicting Types: 1.
 Number of Variables with Differing Attributes: 3.

Listing of Common Variables with Conflicting Types

Variable	Dataset	Type	Length
student	WORK.ONE	Num	8
	WORK.TWO	Char	8

Listing of Common Variables with Differing Attributes

Variable	Dataset	Type	Length	Format	Label
year	WORK.ONE	Char	8		Year of Birth
	WORK.TWO	Char	8		
state	WORK.ONE	Char	8		
	WORK.TWO	Char	8		Home State

Output 11.2 Part Two of Comparison of WORK.ONE and WORK.TWO**The SAS System**

The COMPARE Procedure
 Comparison of WORK.ONE with WORK.TWO
 (Method=EXACT)

Listing of Common Variables with Differing Attributes

Variable	Dataset	Type	Length	Format	Label
gr1	WORK.ONE	Num	8	4.1	
	WORK.TWO	Num	8	5.2	

Observation Summary

Observation	Base	Compare
First Obs	1	1
First Unequal	1	1
Last Unequal	4	4
Last Match	4	4
Last Obs	.	5

Number of Observations in Common: 4.
 Number of Observations in WORK.TWO but not in WORK.ONE: 1.
 Total Number of Observations Read from WORK.ONE: 4.
 Total Number of Observations Read from WORK.TWO: 5.

Number of Observations with Some Compared Variables Unequal: 4.
 Number of Observations with All Compared Variables Equal: 0.

Output 11.3 Part Three of Comparison of WORK.ONE and WORK.TWO

The SAS System						
The COMPARE Procedure Comparison of WORK.ONE with WORK.TWO (Method=EXACT)						
Values Comparison Summary						
Number of Variables Compared with All Observations Equal: 1. Number of Variables Compared with Some Observations Unequal: 3. Total Number of Values which Compare Unequal: 6. Maximum Difference: 20.						
Variables with Unequal Values						
Variable	Type	Len	Compare Label	Ndif	MaxDif	
state	CHAR	8	Home State	2		
gr1	NUM	8		2	1.000	
gr2	NUM	8		2	20.000	

“Procedure Output” on page 248 shows the default output for these two data sets.
 “Example 1: Producing a Complete Report of the Differences” on page 262 shows the complete output for these two data sets.

Results: COMPARE Procedure

Results Reporting

PROC COMPARE reports the results of its comparisons in the following ways:

- the SAS log
- return codes stored in the automatic macro SYSINFO
- procedure output
- output data sets

SAS Log

When you use the WARNING, PRINTALL, or ERROR option, PROC COMPARE writes a description of the differences to the SAS log.

Macro Return Codes (SYSINFO)

PROC COMPARE stores a return code in the automatic macro variable SYSINFO. The value of the return code provides information about the result of the comparison. By checking the value of SYSINFO after PROC COMPARE has run and before any other step begins, SAS macros can use the results of a PROC COMPARE step to determine what action to take or what parts of a SAS program to execute.

The following table is a key for interpreting the SYSINFO return code from PROC COMPARE. For each of the conditions listed, the associated value is added to the return code if the condition is true. Thus, the SYSINFO return code is the sum of the codes listed in the following table for the applicable conditions:

Table 11.1 Macro Return Codes

Bit	Condition	Code	Hexadecimal	Description
1	DSLABEL	1	0001X	Data set labels differ
2	DSTYPE	2	0002X	Data set types differ
3	INFORMAT	4	0004X	Variable has different informat
4	FORMAT	8	0008X	Variable has different format
5	LENGTH	16	0010X	Variable has different length
6	LABEL	32	0020X	Variable has different label
7	BASEOBS	64	0040X	Base data set has observation not in comparison
8	COMPOBS	128	0080X	Comparison data set has observation not in base
9	BASEBY	256	0100X	Base data set has BY group not in comparison
10	COMPBY	512	0200X	Comparison data set has BY group not in base

Bit	Condition	Code	Hexadecimal	Description
11	BASEVAR	1024	0400X	Base data set has variable not in comparison
12	COMPVAR	2048	0800X	Comparison data set has variable not in base
13	VALUE	4096	1000X	A value comparison was unequal
14	TYPE	8192	2000X	Conflicting variable types
15	BYVAR	16384	4000X	BY variables do not match
16	ERROR	32768	8000X	Fatal error: comparison not done

These codes are ordered and scaled to enable a simple check of the degree to which the data sets differ. For example, if you want to check that two data sets contain the same variables, observations, and values, but you do not care about differences in labels, formats, and so on, then use the following statements:

```
proc compare base=SAS-data-set
             compare=SAS-data-set;
run;

%if &sysinfo >= 64 %then
  %do;
    handle error;
  %end;
```

Each time you include SYSINFO in your code, the value for each subsequent instance of SYSINFO is added to the previous value. The final value for SYSINFO is the cumulative value for all of the instances of SYSINFO in the code. For example, if you run the following example code, the first instance of SYSINFO produces a return code of 32, and the second instance produces a return code of 4096. The two values are added together to produce the final return code of 4128.

```
/*diff label -- RC is 32*/
data class1;
  set sashelp.class;
  label sex='Gender';
run;

data class2;
  set sashelp.class;
run;

proc compare base=class1 comp=class2;
run;

%let rc=&sysinfo
%put 'RC' &rc

/*diff label and value -- RC is 4128*/
data class1;
```

```

        set sashelp.class;
        label sex='Gender';
run;
data class2;
    set sashelp.class;
    if name="Jeffrey" then name="Jeff";
run;

proc compare base=class1 comp=class2;
run;

%let rc=&sysinfo;
%put 'RC' &rc

```

You can examine individual bits in the SYSINFO value by using DATA step bit-testing features to check for specific conditions. For example, to check for the presence of observations in the base data set that are not in the comparison data set, use the following statements:

```

proc compare base=SAS-data-set
             compare=SAS-data-set;
run;

%let rc=&sysinfo;
data _null_;
    /* Test for data set label */
    if &rc = '1'b then
        put '<<<< Data sets have different labels';
    /* Test for data set types */
    if &rc = '1.'b then
        put '<<<< Data set types differ';
    /* Test for label */
    if &rc = '1.....'b then
        put '<<<< Variable has different label';
    /* Test for base observation */
    if &rc = '1.....'b then
        put '<<<< Base data set has observation not in comparison data set';
    /* Test for length */
    if &rc = '1....'b then
        put '<<<< Variable has different lengths between the base data set
            and the comparison data set';
    /* Variable in base data set not in compare data set */
    if &rc = '1.....'b then
        put '<<<< Variable in base data set not found in comparison data set';
    /* Comparison data set has variable not in base data set */
    if &rc = '1.....'b then
        put '<<<< Comparison data set has variable not contained in the
            base data set';
    /* Test for values */
    if &rc = '1.....'b then
        put '<<<< A value comparison was unequal';
    /* Conflicting variable types */
    if &rc = '1.....'b then
        put '<<<< Conflicting variable types between the two data sets
            being compared';
run;

```

PROC COMPARE must run before you check SYSINFO and you must obtain the SYSINFO value before another SAS step starts because every SAS step resets SYSINFO.

Procedure Output

Procedure Output Overview

The following sections show and describe the default output of the two data sets shown in [Overview: COMPARE Procedure on page 210](#). Because PROC COMPARE produces lengthy output, the output is presented in seven pieces.

```
options nodate pageno=1 linesize=80 pagesize=60;
proc compare base=proclib.one compare=proclib.two;
run;
```

Data Set Summary

This report lists the attributes of the data sets that are being compared. These attributes include the following:

- the data set names
- the data set types, if any
- the data set labels, if any
- the dates created and last modified
- the number of variables in each data set
- the number of observations in each data set

To view the Data Set Summary, see [Output 11.35 on page 250](#).

Note: The COMPARE procedure omits data set labels if the line size is too small for them.

Variables Summary

This report compares the variables in the two data sets. The first part of the report lists the following:

- the number of variables the data sets have in common

- the number of variables in the base data set that are not in the comparison data set and vice versa
- the number of variables in both data sets that have different types
- the number of variables that differ on other attributes (length, label, format, or informat)
- the number of BY, ID, VAR, and WITH variables specified for the comparison

The second part of the report lists matching variables with different attributes and shows how the attributes differ. (The COMPARE procedure omits variable labels if the line size is too small for them.)

The following output shows the Data Set Summary and the Variables Summary.

Output 11.4 Partial Output Showing the Data Set Summary and Variables Summary

The SAS System

The COMPARE Procedure
Comparison of WORK.ONE with WORK.TWO
(Method=EXACT)

Data Set Summary

Dataset	Created	Modified	NVar	NObs	Label
WORK.ONE	18MAR11:11:22:21	18MAR11:11:22:21	5	4	First Data Set
WORK.TWO	18MAR11:11:22:22	18MAR11:11:22:22	6	5	Second Data Set

Variables Summary

Number of Variables in Common: 5.
Number of Variables in WORK.TWO but not in WORK.ONE: 1.
Number of Variables with Conflicting Types: 1.
Number of Variables with Differing Attributes: 3.

Listing of Common Variables with Conflicting Types

Variable	Dataset	Type	Length
student	WORK.ONE	Num	8
	WORK.TWO	Char	8

Listing of Common Variables with Differing Attributes

Variable	Dataset	Type	Length	Format	Label
year	WORK.ONE	Char	8		Year of Birth
	WORK.TWO	Char	8		
state	WORK.ONE	Char	8		Home State
	WORK.TWO	Char	8		
gr1	WORK.ONE	Num	8	4.1	
	WORK.TWO	Num	8	5.2	

Observation Summary

This report provides information about observations in the base and comparison data sets. First of all, the report identifies the first and last observation in each data set, the first and last matching observations, and the first and last different observations. Then, the report lists the following:

- the number of observations that the data sets have in common

- the number of observations in the base data set that are not in the comparison data set and vice versa
- the total number of observations in each data set
- the number of matching observations for which PROC COMPARE judged some variables unequal
- the number of matching observations for which PROC COMPARE judged all variables equal

The following output shows the Observation Summary.

Output 11.5 Partial Output Showing the Observation Summary

Observation Summary		
Observation	Base	Compare
First Obs	1	1
First Unequal	1	1
Last Unequal	4	4
Last Match	4	4
Last Obs	.	5

Number of Observations in Common: 4.
 Number of Observations in WORK.TWO but not in WORK.ONE: 1.
 Total Number of Observations Read from WORK.ONE: 4.
 Total Number of Observations Read from WORK.TWO: 5.

 Number of Observations with Some Compared Variables Unequal: 4.
 Number of Observations with All Compared Variables Equal: 0.

Values Comparison Summary

This report first lists the following:

- the number of variables compared with all observations equal
- the number of variables compared with some observations unequal
- the number of variables with differences involving missing values, if any
- the total number of values judged unequal
- the maximum difference measure between unequal values for all pairs of matching variables (for differences not involving missing values)

In addition, for the variables for which some matching observations have unequal values, the report lists the following:

- the name of the variable
- other variable attributes
- the number of times PROC COMPARE judged the variable unequal

- the maximum difference measure found between values (for differences not involving missing values)
- the number of differences caused by comparison with missing values, if any

The following output shows the Values Comparison Summary.

Output 11.6 Partial Output Showing the Values Comparison Summary

The SAS System					
The COMPARE Procedure Comparison of WORK.ONE with WORK.TWO (Method=EXACT)					
Values Comparison Summary					
Number of Variables Compared with All Observations Equal: 1. Number of Variables Compared with Some Observations Unequal: 3. Total Number of Values which Compare Unequal: 6. Maximum Difference: 20.					
Variables with Unequal Values					
Variable	Type	Len	Compare Label	Ndif	MaxDif
state	CHAR	8	Home State	2	
gr1	NUM	8		2	1.000
gr2	NUM	8		2	20.000

Value Comparison Results

This report consists of a table for each pair of matching variables judged unequal at one or more observations.

Note:

- When it is comparing character values, PROC COMPARE displays only the first 20 characters. When you use the TRANSPOSE option, PROC COMPARE displays only the first 12 characters.

If you are comparing character values, in the output PROC COMPARE displays a plus sign at the end of a character string that is longer than 20 characters. If you specify the TRANSPOSE option PROC COMPARE displays a plus sign in the output at the end of a character string that is longer than 12 characters. In both instances, the plus sign appears in the table-cell border above the character string. Here is an example.

The COMPARE Procedure			
Comparison of WORK.X with WORK.Y			
(Method=EXACT)			
Value Comparison Results for Variables			
Obs		Base Value	Compare Value
		x	x
1		12345678901234567890	12345678901234567890

Note: The plus sign is displayed at the end of the border above the character string.

- If you do not specify an adequate value for the LINESIZE option, SAS might write a warning that indicates the LINESIZE value is too small to print all of the ID variables in the value comparison reports. PROC COMPARE reserves space for the value section based on the formatted width of the data values. The rest of the space is available for ID values. The character ID values use the width that is specified by their format, and numeric variables use approximately 10 spaces.

Each table shows the following:

- the number of the observation or, if you use the ID statement, the values of the ID variables
- the value of the variable in the base data set
- the value of the variable in the comparison data set
- the difference between these two values (numeric variables only)
- the percent difference between these two values (numeric variables only)

The following output shows the Value Comparison Results for Variables.

Output 11.7 Partial Output Showing the Value Comparison Results for Variables

Value Comparison Results for Variables					
Obs		Home State			
		Base Value state	Compare Value state		
2		MD	MA		
4		MA	MD		

Obs		Base gr1	Compare gr1	Diff.	% Diff
1		85.0	84.00	-1.0000	-1.1765
3		78.0	79.00	1.0000	1.2821

Obs		Base gr2	Compare gr2	Diff.	% Diff
3		72.0000	73.0000	1.0000	1.3889
4		94.0000	74.0000	-20.0000	-21.2766

You can suppress the value comparison results with the NOVALUES option. If you use both the NOVALUES and TRANSPOSE options, then PROC COMPARE lists for each observation the names of the variables with values judged unequal but does not display the values and differences.

The display limits of PROC COMPARE and the TRANSPOSE option apply only to the printed output generated by PROC COMPARE. To see the entire value, you have to use the following options to create an output data set with PROC COMPARE:

- OUT= specifies the name of the output data set.
- OUTNOEQUAL suppresses writing observations where all variables match.
- OUTBASE and OUTCOMP include the observations from the BASE= and COMPARE= data sets.
- NOPRINT suppresses the default printed reports.

The following example produces a Differences data set and then runs PROC PRINT.

```
data x;
a='aaaaaaaaaaaaaaaaaaaaa';
```

```
data y;  
a='aaaaaaaaaaaaaaaaaaaaab';  
run;  
proc compare base=x comp=y out=dif outbase outcomp outdif outnoequal;  
run;  
proc print data=dif;  
run;
```

Figure 11.3 Differences Data Set

The SAS System			
Obs	_TYPE_	_OBS_	a
1	BASE	1	aaaaaaaaaaaaaaaaaaaaaa
2	COMPARE	1	aaaaaaaaaaaaaaaaaaaaaab
3	DIF	1	X

Table of Summary Statistics

If you use the STATS, ALLSTATS, or PRINTALL option, then the Value Comparison Results for Variables section contains summary statistics for the numeric variables that are being compared. The STATS option generates these statistics for only the numeric variables whose values are judged unequal. The ALLSTATS and PRINTALL options generate these statistics for all numeric variables, even if all values are judged equal. Here is the formula that is used to generate the statistics for the ALLSTATS section.

$$(ystat-xstat)/x*100$$

Note: In all cases PROC COMPARE calculates the summary statistics based on all matching observations that do not contain missing values, not just on those containing unequal values.

The following output shows the following summary statistics for base data set values, comparison data set values, differences, and percent differences:

N
the number of nonmissing values.

Note: The value of the N statistic is always the count of nonmissing observations for all four of the columns in the output. The Diff and %Diff columns do not perform any calculations on the N statistic.

MEAN
the mean, or average, of the values.

STD
the standard deviation.

MAX	the maximum value.
MIN	the minimum value.
MISSDIFF	the number of missing values in either a base or compare data set.
STDERR	the standard error of the mean.
T	the T ratio (MEAN/STDERR).
PROB> T	the probability of a greater absolute T value if the true population mean is 0.
NDIF	the number of matching observations judged unequal, and the percent of the matching observations that were judged unequal.
DIFMEANS	the difference between the mean of the base values and the mean of the comparison values. This line contains three numbers. The first is the mean expressed as a percentage of the base values mean. The second is the mean expressed as a percentage of the comparison values mean. The third is the difference in the two means (the comparison mean minus the base mean).
R	the correlation of the base and comparison values for matching observations that are nonmissing in both data sets.
RSQ	the square of the correlation of the base and comparison values for matching observations that are nonmissing in both data sets.

The following output is from the ALLSTATS option using the two data sets shown in [Overview: COMPARE Procedure on page 210](#):

```
options nodate pageno=1
      linesize=80 pagesize=60;
proc compare base=proclib.one
      compare=proclib.two allstats;
      title 'Comparing Two Data Sets: Default Report';
run;
```

Output 11.8 Partial Output Showing Value Comparison Results for Variables

Value Comparison Results for Variables					
		Home State			
		Base Value		Compare Value	
Obs		state		state	
2		MD		MA	
4		MA		MD	
					</

Note: If you use a wide line size with PRINTALL, then PROC COMPARE prints the value comparison result for character variables next to the result for numeric variables. In that case, PROC COMPARE calculates only NDIF for the character variables.

Comparison Results for Observations (Using the TRANSPOSE Option)

The TRANSPOSE option prints the comparison results by observation instead of by variable. The comparison results precede the observation summary report. By

default, the source of the values for each row of the table is indicated by the following label:

```
_OBS_1=number-1 _OBS_2=number-2
```

where *number-1* is the number of the observation in the base data set for which the value of the variable is shown, and *number-2* is the number of the observation in the comparison data set.

The following output shows the differences in PROCLIB.ONE and PROCLIB.TWO by observation instead of by variable.

```
options nodate pageno=1
      linesize=80 pagesize=60;
proc compare base=proclib.one
      compare=proclib.two transpose;
      title 'Comparing Two Data Sets: Default Report';
run;
```

Output 11.9 Partial Output Showing Comparison Results for Observations

Comparing Two Data Sets: Default Report				
The COMPARE Procedure				
Comparison of WORK.ONE with WORK.TWO				
(Method=EXACT)				
Comparison Results for Observations				
_OBS_1=3 _OBS_2=3:				
Variable	Base Value	Compare	Diff.	% Diff
gr1	78.0	79.00	1.000000	1.282051
gr2	72.000000	73.000000	1.000000	1.388889
_OBS_1=4 _OBS_2=4:				
Variable	Base Value	Compare	Diff.	% Diff
gr2	94.000000	74.000000	-20.000000	-21.276596
state	MA	MD		

If you use an ID statement, then the identifying label has the following form:

```
ID-1=ID-value-1 ... ID-n=ID-value-n
```

where *ID* is the name of an ID variable and *ID-value* is the value of the ID variable.

Note: When you use the TRANSPOSE option, PROC COMPARE prints only the first 12 characters of the value.

ODS Table Names

The COMPARE procedure assigns a name to each table that it creates. You can use these names to reference the table when using the Output Delivery System (ODS)

to select tables and create output data sets. For more information, see *SAS Output Delivery System: User's Guide*.

Table 11.2 ODS Tables Produced by the COMPARE Procedure

Table Name	Description	Conditions When Table Is Generated
CompareData sets	Information about the data set or data sets	By default, unless NOSUMMARY or NOVALUES option is specified
CompareDetails (Comparison Results for Observations)	A listing of observations that the base data set and the compare data set do not have in common	If PRINTALL option is specified
CompareDetails (ID variable notes and warnings)	A listing of notes and warnings concerning duplicate ID variable values	If ID statement is specified and duplicate ID variable values exist in either data set
CompareDifferences	A report of variable value differences	By default unless NOVALUES option is specified
CompareSummary	Summary report of observations, values, and variables with unequal values	By default
CompareVariables	A listing of differences in variable types or attributes between the base data set and the compare data set	By default, unless the variables are identical or the NOSUMMARY option is specified

Note: The ODS output tables contain the same information as that written to the Output window. These tables do not contain any additional information, and they do not provide a different format for the information. The ODS output tables might be of some use to you, but it depends on the task that you are performing. For an example of the format produced in the Output window, see the outputs in [“Customizing PROC COMPARE Output” on page 241](#).

Output Data Set (OUT=)

By default, the OUT= data set contains an observation for each pair of matching observations. The OUT= data set contains the following variables from the data sets you are comparing:

- all variables named in the BY statement

- all variables named in the ID statement
- all matching variables or, if you use the VAR statement, all variables listed in the VAR statement

In addition, the data set contains two variables created by PROC COMPARE to identify the source of the values for the matching variables: `_TYPE_` and `_OBS_`.

`_TYPE_`

is a character variable of length 8. Its value indicates the source of the values for the matching (or VAR) variables in that observation. (For ID and BY variables, which are not compared, the values are the values from the original data sets.) `_TYPE_` has the label *Type of Observation*. The four possible values of this variable are as follows:

BASE

the values in this observation are from an observation in the base data set. PROC COMPARE writes this type of observation to the OUT= data set when you specify the OUTBASE option.

COMPARE

the values in this observation are from an observation in the comparison data set. PROC COMPARE writes this type of observation to the OUT= data set when you specify the OUTCOMP option.

DIF

the values in this observation are the differences between the values in the base and comparison data sets.

For character variables, PROC COMPARE uses a period (.) to represent equal characters and an X to represent unequal characters.

For numeric variables, an E means that there is no difference. Otherwise, the numeric difference is shown.

PROC COMPARE writes this type of observation to the OUT= data set by default. However, if you request any other type of observation with the OUTBASE, OUTCOMP, or OUTPERCENT option, then you must specify the OUTDIF option to generate observations of this type in the OUT= data set.

For an example output that shows the use of the period, X, and E to represent equal and different values, see [“Example 6: Comparing Values of Observations Using an Output Data Set \(OUT=\)”](#) on page 281.

PERCENT

the values in this observation are the percent differences between the values in the base and comparison data sets. For character variables the values in observations of type PERCENT are the same as the values in observations of type DIF.

`_OBS_`

is a numeric variable that contains a number further identifying the source of the OUT= observations.

For observations with `_TYPE_` equal to BASE, `_OBS_` is the number of the observation in the base data set from which the values of the VAR variables were copied. Similarly, for observations with `_TYPE_` equal to COMPARE, `_OBS_` is the number of the observation in the comparison data set from which the values of the VAR variables were copied.

For observations with `_TYPE_` equal to DIF or PERCENT, `_OBS_` is a sequence number that counts the matching observations in the BY group.

`_OBS_` has the label `Observation Number`.

The COMPARE procedure takes variable names and attributes for the `OUT=` data set from the base data set except for the length of the VAR variable. The COMPARE procedure uses the longer length for the VAR variable regardless of which data set contains that length is from. This behavior has two important repercussions:

- If you use the VAR and WITH statements, then the names of the variables in the `OUT=` data set come from the VAR statement. Thus, observations with `_TYPE_` equal to `BASE` contain the values of the VAR variables, whereas observations with `_TYPE_` equal to `COMPARE` contain the values of the WITH variables.
- If you include a variable more than once in the VAR statement in order to compare it with more than one variable, then PROC COMPARE can include only the first comparison in the `OUT=` data set because each variable must have a unique name. Other comparisons produce warning messages.

For an example of the `OUT=` option, see [“Example 6: Comparing Values of Observations Using an Output Data Set \(OUT=\)”](#) on page 281.

Output Statistics Data Set (OUTSTATS=)

When you use the `OUTSTATS=` option, PROC COMPARE calculates the same summary statistics as the `ALLSTATS` option for each pair of numeric variables that are compared. For more information, see [“Table of Summary Statistics”](#) on page 255). The `OUTSTATS=` data set contains an observation for each summary statistic for each pair of variables. The data set also contains the BY variables used in the comparison and several variables created by PROC COMPARE:

`_VAR_`

is a character variable that contains the name of the variable from the base data set for which the statistic in the observation was calculated.

`_WITH_`

is a character variable that contains the name of the variable from the comparison data set for which the statistic in the observation was calculated. The `_WITH_` variable is not included in the `OUTSTATS=` data set unless you use the WITH statement.

`_TYPE_`

is a character variable that contains the name of the statistic contained in the observation. Values of the `_TYPE_` variable are N, MEAN, STD, MIN, MAX, STDERR, T, PROBT, NDIF, DIFMEANS, and R, RSQ.

`_BASE_`

is a numeric variable that contains the value of the statistic calculated from the values of the variable named by `_VAR_` in the observations in the base data set with matching observations in the comparison data set.

COMP

is a numeric variable that contains the value of the statistic calculated from the values of the variable named by the `_VAR_` variable (or by the `_WITH_` variable if you use the `WITH` statement) in the observations in the comparison data set with matching observations in the base data set.

DIF

is a numeric variable that contains the value of the statistic calculated from the differences of the values of the variable named by the `_VAR_` variable in the base data set and the matching variable (named by the `_VAR_` or `_WITH_` variable) in the comparison data set.

PCTDIF

is a numeric variable that contains the value of the statistic calculated from the percent differences of the values of the variable named by the `_VAR_` variable in the base data set and the matching variable (named by the `_VAR_` or `_WITH_` variable) in the comparison data set.

Note: For both types of output data sets, PROC COMPARE assigns one of the following data set labels:

*Comparison of base-SAS-data-set
with comparison-SAS-data-set*

Comparison of variables in base-SAS-data-set

Labels are limited to 40 characters.

See “[Example 7: Creating an Output Data Set of Statistics \(OUTSTATS=\)](#)” on page 285 for an example of an `OUTSTATS=` data set.

Examples: COMPARE Procedure

Example 1: Producing a Complete Report of the Differences

Features: PROC COMPARE statement options
 BASE=
 PRINTALL
 COMPARE=

Data set: [Proclib.One](#), [Proclib.Two](#)

Details

This example shows the most complete report that PROC COMPARE produces as procedure output.

Program

```
libname proclib 'SAS-library';
options nodate pageno=1 linesize=80 pagesize=40;
proc compare base=proclib.one compare=proclib.two printall;
    title 'Comparing Two Data Sets: Full Report';
run;
```

Program Description

Declare the PROCLIB SAS library.

```
libname proclib 'SAS-library';
```

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=40;
```

Create a complete report of the differences between two data sets. BASE= and COMPARE= specify the data sets to compare. PRINTALL prints a full report of the differences.

```
proc compare base=proclib.one compare=proclib.two printall;
    title 'Comparing Two Data Sets: Full Report';
run;
```

Output: HTML

A > in the output marks information that is in the full report but not in the default report. The additional information includes a listing of variables found in one data set but not the other, a listing of observations found in one data set but not the other, a listing of variables with all equal values, and summary statistics. For an explanation of the statistics, see [“Table of Summary Statistics” on page 255](#).

Output 11.10 Part One of Comparing Two Data Sets: Full Report**Comparing Two Data Sets: Full Report**

The COMPARE Procedure
 Comparison of WORK.ONE with WORK.TWO
 (Method=EXACT)

Data Set Summary

Dataset	Created	Modified	NVar	NObs	Label
WORK.ONE	16MAR11:17:52:15	16MAR11:17:52:15	5	4	First Data Set
WORK.TWO	16MAR11:17:52:15	16MAR11:17:52:15	6	5	Second Data Set

Variables Summary

Number of Variables in Common: 5.
 Number of Variables in WORK.TWO but not in WORK.ONE: 1.
 Number of Variables with Conflicting Types: 1.
 Number of Variables with Differing Attributes: 3.

Listing of Variables in WORK.TWO but not in WORK.ONE

Variable	Type	Length
major	Char	8

Listing of Common Variables with Conflicting Types

Variable	Dataset	Type	Length
student	WORK.ONE	Num	8
	WORK.TWO	Char	8

Output 11.11 Part Two of Comparing Two Data Sets: Full Report**Comparing Two Data Sets: Full Report**

The COMPARE Procedure
 Comparison of WORK.ONE with WORK.TWO
 (Method=EXACT)

Listing of Common Variables with Differing Attributes

Variable	Dataset	Type	Length	Format	Label
year	WORK.ONE	Char	8		Year of Birth
	WORK.TWO	Char	8		
state	WORK.ONE	Char	8		Home State
	WORK.TWO	Char	8		
gr1	WORK.ONE	Num	8	4.1	
	WORK.TWO	Num	8	5.2	

Comparison Results for Observations

Observation 5 in WORK.TWO not found in WORK.ONE.

Observation Summary

Observation	Base	Compare
First Obs	1	1
First Unequal	1	1
Last Unequal	4	4
Last Match	4	4
Last Obs	.	5

Number of Observations in Common: 4.

Number of Observations in WORK.TWO but not in WORK.ONE: 1.

Total Number of Observations Read from WORK.ONE: 4.

Total Number of Observations Read from WORK.TWO: 5.

Number of Observations with Some Compared Variables Unequal: 4.

Number of Observations with All Compared Variables Equal: 0.

Output 11.12 Part Three of Comparing Two Data Sets: Full Report**Comparing Two Data Sets: Full Report**

The COMPARE Procedure
 Comparison of WORK.ONE with WORK.TWO
 (Method=EXACT)

Values Comparison Summary

Number of Variables Compared with All Observations Equal: 1.
 Number of Variables Compared with Some Observations Unequal: 3.
 Total Number of Values which Compare Unequal: 6.
 Maximum Difference: 20.

Variables with All Equal Values

Variable	Type	Len	Label
year	CHAR	8	Year of Birth

Variables with Unequal Values

Variable	Type	Len	Compare Label	Ndif	MaxDif
state	CHAR	8	Home State	2	
gr1	NUM	8		2	1.000
gr2	NUM	8		2	20.000

Output 11.13 Part Four of Comparing Two Data Sets: Full Report**Comparing Two Data Sets: Full Report**

The COMPARE Procedure
 Comparison of WORK.ONE with WORK.TWO
 (Method=EXACT)

Value Comparison Results for Variables

		Year of Birth	
		Base Value	Compare Value
Obs		year	year
1		1970	1970
2		1971	1971
3		1969	1969
4		1970	1970

		Home State	
		Base Value	Compare Value
Obs		state	state
1		NC	NC
2		MD	MA
3		PA	PA
4		MA	MD

Output 11.14 Part Five of Comparing Two Data Sets: Full Report**Comparing Two Data Sets: Full Report**

The COMPARE Procedure
 Comparison of WORK.ONE with WORK.TWO
 (Method=EXACT)

Value Comparison Results for Variables

Obs	Base gr1	Compare gr1	Diff.	% Diff
1	85.0	84.00	-1.0000	-1.1765
2	92.0	92.00	0	0
3	78.0	79.00	1.0000	1.2821
4	87.0	87.00	0	0
N	4	4	4	4
Mean	85.5000	85.5000	0	0.0264
Std	5.8023	5.4467	0.8165	1.0042
Max	92.0000	92.0000	1.0000	1.2821
Min	78.0000	79.0000	-1.0000	-1.1765
StdErr	2.9011	2.7234	0.4082	0.5021
t	29.4711	31.3951	0.0000	0.0526
Prob> t	<.0001	<.0001	1.0000	0.9614
Ndif	2	50.000%		
DifMeans	0.000%	0.000%	0	
r, rsq	0.991	0.983		

Output 11.15 Part Six of Comparing Two Data Sets: Full Report

Comparing Two Data Sets: Full Report					
The COMPARE Procedure					
Comparison of WORK.ONE with WORK.TWO					
(Method=EXACT)					
Value Comparison Results for Variables					
Obs		Base gr2	Compare gr2	Diff.	% Diff
1		87.0000	87.0000	0	0
2		92.0000	92.0000	0	0
3		72.0000	73.0000	1.0000	1.3889
4		94.0000	74.0000	-20.0000	-21.2766
N		4	4	4	4
Mean		86.2500	81.5000	-4.7500	-4.9719
Std		9.9457	9.4692	10.1776	10.8895
Max		94.0000	92.0000	1.0000	1.3889
Min		72.0000	73.0000	-20.0000	-21.2766
StdErr		4.9728	4.7346	5.0888	5.4447
t		17.3442	17.2136	-0.9334	-0.9132
Prob> t		0.0004	0.0004	0.4195	0.4285
Ndif		2	50.000%		
DifMeans		-5.507%	-5.828%	-4.7500	
r, rsq		0.451	0.204		

Example 2: Comparing Variables in Different Data Sets

Features: PROC COMPARE statement option
NOSUMMARY

VAR statement

WITH statement

Data set: [Proclib.One](#), [Proclib.Two](#)

Details

This example compares a variable from the base data set with a variable in the comparison data set. All summary reports are suppressed.

Program

```
libname proclib 'SAS-library';
options nodate pageno=1 linesize=80 pagesize=40;
proc compare base=proclib.one compare=proclib.two nosummary;
    var gr1;
    with gr2;
    title 'Comparison of Variables in Different Data Sets';
run;
```

Program Description

Declare the PROCLIB SAS library.

```
libname proclib 'SAS-library';
```

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=40;
```

Suppress all summary reports of the differences between two data sets. BASE= specifies the base data set and COMPARE= specifies the comparison data set. NOSUMMARY suppresses all summary reports.

```
proc compare base=proclib.one compare=proclib.two nosummary;
```

Specify one variable from the base data set to compare with one variable from the comparison data set. The VAR and WITH statements specify the variables to compare. This example compares GR1 from the base data set with GR2 from the comparison data set.

```
    var gr1;
    with gr2;
    title 'Comparison of Variables in Different Data Sets';
run;
```

Output: HTML

Output 11.16 Comparison of Variables in Different Data Sets

Comparison of Variables in Different Data Sets					
The COMPARE Procedure Comparison of WORK.ONE with WORK.TWO (Method=EXACT) NOTE: Data set WORK.TWO contains 1 observations not in WORK.ONE. NOTE: Values of the following 1 variables compare unequal: gr1^=gr2					
Value Comparison Results for Variables					
Obs		Base gr1	Compare gr2	Diff.	% Diff
1		85.0	87.0000	2.0000	2.3529
3		78.0	73.0000	-5.0000	-6.4103
4		87.0	74.0000	-13.0000	-14.9425

Example 3: Comparing a Variable Multiple Times

Features: VAR statement
WITH statement

Data set: [Proclib.One](#), [Proclib.Two](#)

Details

This example compares one variable from the base data set with two variables in the comparison data set.

Program

```
libname proclib 'SAS-library';
options nodate pageno=1 linesize=80 pagesize=40;
```

```
proc compare base=proclib.one compare=proclib.two nosummary;
    var gr1 gr1;
    with gr1 gr2;
    title 'Comparison of One Variable with Two Variables';
run;
```

Program Description

Declare the PROCLIB SAS library.

```
libname proclib 'SAS-library';
```

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=40;
```

Suppress all summary reports of the differences between two data sets. BASE= specifies the base data set and COMPARE= specifies the comparison data set. NOSUMMARY suppresses all summary reports.

```
proc compare base=proclib.one compare=proclib.two nosummary;
```

Specify one variable from the base data set to compare with two variables from the comparison data set. The VAR and WITH statements specify the variables to compare. This example compares GR1 from the base data set with GR1 and GR2 from the comparison data set.

```
    var gr1 gr1;
    with gr1 gr2;
    title 'Comparison of One Variable with Two Variables';
run;
```

Output: HTML

The Value Comparison Results section shows the result of the comparison.

Output 11.17 Comparison of One Variable with Two Variables**Comparison of One Variable with Two Variables**

The COMPARE Procedure
 Comparison of WORK.ONE with WORK.TWO
 (Method=EXACT)

NOTE: Data set WORK.TWO contains 1 observations not in WORK.ONE.

NOTE: Values of the following 2 variables compare unequal: gr1^=gr1 gr1^=gr2

Value Comparison Results for Variables

Obs		Base gr1	Compare gr1	Diff.	% Diff
1		85.0	84.00	-1.0000	-1.1765
3		78.0	79.00	1.0000	1.2821

Obs		Base gr1	Compare gr2	Diff.	% Diff
1		85.0	87.0000	2.0000	2.3529
3		78.0	73.0000	-5.0000	-6.4103
4		87.0	74.0000	-13.0000	-14.9425

Example 4: Comparing Variables That Are in the Same Data Set

Features: PROC COMPARE statement options
 ALLSTATS
 BRIEFSUMMARY
 VAR statement
 WITH statement

Data set: [Proclib.One](#)

Details

This example shows that PROC COMPARE can compare two variables that are in the same data set.

Program

```
libname proclib 'SAS-library';
options nodate pageno=1 linesize=80 pagesize=40;
proc compare base=proclib.one allstats briefsummary;
    var gr1;
    with gr2;
    title 'Comparison of Variables in the Same Data Set';
run;
```

Program Description

Declare the Proclib SAS library.

```
libname proclib 'SAS-library';
```

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=40;
```

Create a short summary report of the differences within one data set. ALLSTATS prints summary statistics. BRIEFSUMMARY prints only a short comparison summary.

```
proc compare base=proclib.one allstats briefsummary;
```

Specify two variables from the base data set to compare. The VAR and WITH statements specify the variables in the base data set to compare. This example compares GR1 with GR2. Because there is no comparison data set, the variables GR1 and GR2 must be in the base data set.

```
    var gr1;
    with gr2;
    title 'Comparison of Variables in the Same Data Set';
run;
```


Output: HTML

Output 11.18 Comparison of One Variable with Two Variables

Comparison of One Variable with Two Variables					
The COMPARE Procedure Comparisons of variables in WORK.ONE (Method=EXACT)					
NOTE: Values of the following 1 variables compare unequal: gr1^=gr2					
Value Comparison Results for Variables					
Obs		Base gr1	Compare gr2	Diff.	% Diff
1		85.0	87.0000	2.0000	2.3529
3		78.0	72.0000	-6.0000	-7.6923
4		87.0	94.0000	7.0000	8.0460
N		4	4	4	4
Mean		85.5000	86.2500	0.7500	0.6767
Std		5.8023	9.9457	5.3774	6.5221
Max		92.0000	94.0000	7.0000	8.0460
Min		78.0000	72.0000	-6.0000	-7.6923
StdErr		2.9011	4.9728	2.6887	3.2611
t		29.4711	17.3442	0.2789	0.2075
Prob> t		<.0001	0.0004	0.7984	0.8489
Ndif		3	75.000%		
DifMeans		0.877%	0.870%	0.7500	
r, rsq		0.898	0.807		

Example 5: Comparing Observations with an ID Variable

Features: ID statement

Data sets: [Proclib.Emp95](#)
[Proclib.Emp96](#)

Details

In this example, PROC COMPARE compares only the observations that have matching values for the ID variable.

Program

```
libname proclib 'SAS-library';

options nodate pageno=1 linesize=80 pagesize=40;

data proclib.emp95;
    input #1 idnum $4. @6 name $15.
           #2 address $42.
           #3 salary 6.;
    datalines;
2388 James Schmidt
100 Apt. C Blount St. SW Raleigh NC 27693
92100
2457 Fred Williams
99 West Lane Garner NC 27509
33190
... more data lines...
3888 Kim Siu
5662 Magnolia Blvd Southeast Cary NC 27513
77558
;

data proclib.emp96;
    input #1 idnum $4. @6 name $15.
           #2 address $42.
           #3 salary 6.;
    datalines;
2388 James Schmidt
100 Apt. C Blount St. SW Raleigh NC 27693
92100
2457 Fred Williams
99 West Lane Garner NC 27509
33190
...more data lines...
6544 Roger Monday
3004 Crepe Myrtle Court Raleigh NC 27604
47007
;

proc sort data=proclib.emp95 out=emp95_byidnum;

    by idnum;
run;

proc sort data=proclib.emp96 out=emp96_byidnum;
```

```

        by idnum;
run;

proc compare base=emp95_byidnum compare=emp96_byidnum;
    id idnum;
    title 'Comparing Observations that Have Matching IDNUMs';
run;

```

Program Description

Declare the PROCLIB SAS library.

```
libname proclib 'SAS-library';
```

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=40;
```

Create the Proclib.Emp95 and Proclib.Emp96 data sets. Proclib.Emp95 and Proclib.Emp96 contain employee data. IDNUM works well as an ID variable because it has unique values. The first DATA step creates Proclib.Emp95. The second DATA step creates Proclib.Emp96.

```

data proclib.emp95;
    input #1 idnum $4. @6 name $15.
           #2 address $42.
           #3 salary 6.;
    datalines;
2388 James Schmidt
100 Apt. C Blount St. SW Raleigh NC 27693
92100
2457 Fred Williams
99 West Lane Garner NC 27509
33190
... more data lines...
3888 Kim Siu
5662 Magnolia Blvd Southeast Cary NC 27513
77558
;

data proclib.emp96;
    input #1 idnum $4. @6 name $15.
           #2 address $42.
           #3 salary 6.;
    datalines;
2388 James Schmidt
100 Apt. C Blount St. SW Raleigh NC 27693
92100
2457 Fred Williams
99 West Lane Garner NC 27509
33190
...more data lines...
6544 Roger Monday

```

```

3004 Crepe Myrtle Court Raleigh NC 27604
47007
;

```

Sort the data sets by the ID variable. Both data sets must be sorted by the variable that will be used as the ID variable in the PROC COMPARE step. OUT= specifies the location of the sorted data.

```

proc sort data=proclib.emp95 out=emp95_byidnum;

    by idnum;
run;

proc sort data=proclib.emp96 out=emp96_byidnum;
    by idnum;
run;

```

Create a summary report that compares observations with matching values for the ID variable. The ID statement specifies IDNUM as the ID variable.

```

proc compare base=emp95_byidnum compare=emp96_byidnum;
    id idnum;
    title 'Comparing Observations that Have Matching IDNUMs';
run;

```

Output: HTML

PROC COMPARE identifies specific observations by the value of IDNUM. In the Value Comparison Results for Variables section, PROC COMPARE prints the nonmatching addresses and nonmatching salaries. For salaries, PROC COMPARE computes the numerical difference and the percent difference. Because ADDRESS is a character variable, PROC COMPARE displays only the first 20 characters. For addresses where the observation has an IDNUM of 0987, 2776, or 3888, the differences occur after the 20th character and the differences do not appear in the output. The plus sign in the output indicates that the full value is not shown. To see the entire value, create an output data set. See [“Example 6: Comparing Values of Observations Using an Output Data Set \(OUT=\)”](#) on page 281.

Output 11.19 Part One of Comparing Observations That Have Matching IDNUMs**Comparing Observations that Have Matching IDNUMs**

The COMPARE Procedure
 Comparison of WORK.EMP95_BYIDNUM with WORK.EMP96_BYIDNUM
 (Method=EXACT)

Data Set Summary

Dataset	Created	Modified	NVar	NObs
WORK.EMP95_BYIDNUM	17MAR11:08:58:10	17MAR11:08:58:10	4	10
WORK.EMP96_BYIDNUM	17MAR11:08:58:10	17MAR11:08:58:10	4	12

Variables Summary

Number of Variables in Common: 4.
 Number of ID Variables: 1.

Observation Summary

Observation	Base	Compare	ID
First Obs	1	1	idnum=0987
First Unequal	1	1	idnum=0987
Last Unequal	10	12	idnum=9857
Last Obs	10	12	idnum=9857

Number of Observations in Common: 10.
 Number of Observations in WORK.EMP96_BYIDNUM but not in WORK.EMP95_BYIDNUM: 2.
 Total Number of Observations Read from WORK.EMP95_BYIDNUM: 10.
 Total Number of Observations Read from WORK.EMP96_BYIDNUM: 12.

Number of Observations with Some Compared Variables Unequal: 5.
 Number of Observations with All Compared Variables Equal: 5.

Output 11.20 Part Two of Comparing Observations That Have Matching IDNUMs**Comparing Observations that Have Matching IDNUMs**

The COMPARE Procedure
 Comparison of WORK.EMP95_BYIDNUM with WORK.EMP96_BYIDNUM
 (Method=EXACT)

Values Comparison Summary

Number of Variables Compared with All Observations Equal: 1.
 Number of Variables Compared with Some Observations Unequal: 2.
 Total Number of Values which Compare Unequal: 8.
 Maximum Difference: 2400.

Variables with Unequal Values

Variable	Type	Len	Ndif	MaxDif
address	CHAR	42	4	
salary	NUM	8	4	2400

Value Comparison Results for Variables

	Base Value	Compare Value
idnum	address	address
	_____+	_____+
0987	2344 Persimmons Bran	2344 Persimmons Bran
2776	12988 Wellington Far	12988 Wellington Far
3888	5662 Magnolia Blvd S	5662 Magnolia Blvd S
9857	1000 Taft Ave. Morri	100 Taft Ave. Morris

Output 11.21 Part Three of Comparing Observations That Have Matching IDNUMs

Comparing Observations that Have Matching IDNUMs					
The COMPARE Procedure					
Comparison of WORK.EMP95_BYIDNUM with WORK.EMP96_BYIDNUM					
(Method=EXACT)					
Value Comparison Results for Variables					
idnum		Base salary	Compare salary	Diff.	% Diff
0987		44010	45110	1100	2.4994
3286		87734	89834	2100	2.3936
3888		77558	79958	2400	3.0945
9857		38756	40456	1700	4.3864

Example 6: Comparing Values of Observations Using an Output Data Set (OUT=)

Features: PROC COMPARE statement options

NOPRINT
OUT=
OUTBASE
OUTCOMP
OUTDIF
OUTNOEQUAL

PRINT procedure

Data sets: [Proclib.Emp95](#)
[Proclib.Emp96](#)

Details

This example creates and prints an output data set that shows the differences between matching observations.

In “[Example 5: Comparing Observations with an ID Variable](#)” on page 275, the output does not show the differences past the 20th character. The output data set in this example shows the full values. Further, it shows the observations that occur in only one of the data sets.

Program

```

libname proclib 'SAS-library';
options nodate pageno=1 linesize=120 pagesize=40;
proc sort data=proclib.emp95 out=emp95_byidnum;

    by idnum;
run;

proc sort data=proclib.emp96 out=emp96_byidnum;
    by idnum;
run;

proc compare base=emp95_byidnum compare=emp96_byidnum
             out=result outnoequal outbase outcomp outdif
             noprint;
    id idnum;
run;

proc print data=result noobs;
    by idnum;
    id idnum;
    title 'The Output Data Set RESULT';
run;

```

Program Description

Declare the PROCLIB SAS library.

```
libname proclib 'SAS-library';
```

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=120 pagesize=40;
```

Sort the data sets by the ID variable. Both data sets must be sorted by the variable that will be used as the ID variable in the PROC COMPARE step. OUT= specifies the location of the sorted data.

```

proc sort data=proclib.emp95 out=emp95_byidnum;

    by idnum;
run;

proc sort data=proclib.emp96 out=emp96_byidnum;
    by idnum;
run;

```

Specify the data sets to compare. BASE= and COMPARE= specify the data sets to compare.


```
proc compare base=emp95_byidnum compare=emp96_byidnum
```

Create the Result output data set and include all unequal observations and their differences. OUT= names and creates the output data set. NOPRINT suppresses the printing of the procedure output. OUTNOEQUAL includes only observations that are judged unequal. OUTBASE writes an observation to the output data set for each observation in the base data set. OUTCOMP writes an observation to the output data set for each observation in the comparison data set. OUTDIF writes an observation to the output data set that contains the differences between the two observations.

```
    out=result outnoequal outbase outcomp outdif
  noprint;
```

Specify the ID variable. The ID statement specifies IDNUM as the ID variable.

```
    id idnum;
run;
```

Print the Result output data set and use the BY and ID statements with the ID variable. PROC PRINT prints the output data set. Using the BY and ID statements with the same variable makes the output easy to read. See the PRINT procedure for more information about this technique.

```
proc print data=result noobs;
  by idnum;
  id idnum;
  title 'The Output Data Set RESULT';
run;
```

Output: HTML

The differences for character variables are noted with an X or a period (.). An X shows that the characters do not match. A period shows that the characters do match. For numeric variables, an E means that there is no difference. Otherwise, the numeric difference is shown. By default, the output data set shows that two observations in the comparison data set have no matching observation in the base data set. You do not have to use an option to make those observations appear in the output data set.

Output 11.22 Part One of the Output Data Set RESULT**The Output Data Set RESULT**

idnum	_TYPE_	_OBS_	name	address	salary
0987	BASE	1	Dolly Lunford	2344 Persimmons Branch Apex NC 27505	44010
	COMPARE	1	Dolly Lunford	2344 Persimmons Branch Trail Apex NC 27505	45110
	DIF	1		XXXXX.XXXXXXXXXXXXXX	1100

idnum	_TYPE_	_OBS_	name	address	salary
2776	BASE	5	Robert Jones	12988 Wellington Farms Ave. Cary NC 27512	29025
	COMPARE	5	Robert Jones	12988 Wellington Farms Ave. Cary NC 27511	29025
	DIF	5		X.	E

idnum	_TYPE_	_OBS_	name	address	salary
3278	COMPARE	6	Mary Cravens	211 N. Cypress St. Cary NC 27512	35362

idnum	_TYPE_	_OBS_	name	address	salary
3286	BASE	6	Hoa Nguyen	2818 Long St. Cary NC 27513	87734
	COMPARE	7	Hoa Nguyen	2818 Long St. Cary NC 27513	89834
	DIF	6			2100

Output 11.23 Part Two of the Output Data Set RESULT

idnum	_TYPE_	_OBS_	name	address	salary
3888	BASE	7	Kim Siu	5662 Magnolia Blvd Southeast Cary NC 27513	77558
	COMPARE	8	Kim Siu	5662 Magnolia Blvd Southwest Cary NC 27513	79958
	DIF	7		XX.....	2400

idnum	_TYPE_	_OBS_	name	address	salary
6544	COMPARE	9	Roger Monday	3004 Crepe Myrtle Court Raleigh NC 27604	47007

idnum	_TYPE_	_OBS_	name	address	salary
9857	BASE	10	Kathy Krupski	1000 Taft Ave. Morrisville NC 27508	38756
	COMPARE	12	Kathy Krupski	100 Taft Ave. Morrisville NC 27508	40456
	DIF	10		...XXXXXXXXXXXXX.XXXXX.XXXXXXXXXXXXXX.....	1700

Example 7: Creating an Output Data Set of Statistics (OUTSTATS=)

Features: PROC COMPARE statement options
NOPRINT
OUTSTATS=

Data sets: [Proclib.Emp95](#)
[Proclib.Emp96](#)

Details

This example creates an output data set that contains summary statistics for the numeric variables that are compared.

Program

```
libname proclib 'SAS-library';  
options nodate pageno=1 linesize=80 pagesize=40;  
proc sort data=proclib.emp95 out=emp95_byidnum;  
    by idnum;  
run;  
  
proc sort data=proclib.emp96 out=emp96_byidnum;  
    by idnum;  
run;  
  
proc compare base=emp95_byidnum compare=emp96_byidnum  
    outstats=diffstat noprint;  
    id idnum;  
run;  
  
proc print data=diffstat noobs;  
    title 'The DIFFSTAT Data Set';  
run;
```

Program Description

Declare the Proclib SAS library.

```
libname proclib 'SAS-library';
```

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=40;
```

Sort the data sets by the ID variable. Both data sets must be sorted by the variable that will be used as the ID variable in the PROC COMPARE step. OUT= specifies the location of the sorted data.

```
proc sort data=proclib.emp95 out=emp95_byidnum;
    by idnum;
run;
```

```
proc sort data=proclib.emp96 out=emp96_byidnum;
    by idnum;
run;
```

Create the output data set of statistics and compare observations that have matching values for the ID variable. BASE= and COMPARE= specify the data sets to compare. OUTSTATS= creates the output data set Diffstat. Noprint and suppresses the procedure output. The ID statement specifies IDNUM as the ID variable. PROC COMPARE uses the values of IDNUM to match observations.

```
proc compare base=emp95_byidnum compare=emp96_byidnum
    outstats=diffstat noprint;
    id idnum;
run;
```

Print the output data set Diffstat. PROC PRINT prints the output data set Diffstat.

```
proc print data=diffstat noobs;
    title 'The DIFFSTAT Data Set';
run;
```

Output: HTML

The variables are described in [“Output Statistics Data Set \(OUTSTATS=\)”](#) on page 261.

Output 11.24 The Diffstat Data Set

The DIFFSTAT Data Set					
VAR	_TYPE_	_BASE_	_COMP_	_DIF_	_PCTDIF_
salary	N	10.00	10.00	10.00	10.0000
salary	MEAN	52359.00	53089.00	730.00	1.2374
salary	STD	24143.84	24631.01	996.72	1.6826
salary	MAX	92100.00	92100.00	2400.00	4.3864
salary	MIN	29025.00	29025.00	0.00	0.0000
salary	STDERR	7634.95	7789.01	315.19	0.5321
salary	T	6.86	6.82	2.32	2.3255
salary	PROBT	0.00	0.00	0.05	0.0451
salary	NDIF	4.00	40.00	.	.
salary	DIFMEANS	1.39	1.38	730.00	.
salary	R,RSQ	1.00	1.00	.	.

CONTENTS Procedure

Overview: CONTENTS Procedure	289
What Does the CONTENTS Procedure Do?	289
Concepts: CONTENTS Procedure	290
Concepts: CONTENTS Procedure	290
Syntax: CONTENTS Procedure	292
PROC CONTENTS Statement	293
Usage: CONTENTS Procedure	297
Using PROC CONTENTS	297
Examples: CONTENTS Procedure	298
Example 1: Describing a SAS Data Set	298
Example 2: Using the DIRECTORY Option	300
Example 3: Using the ORDER= Option	302

Overview: CONTENTS Procedure

What Does the CONTENTS Procedure Do?

The CONTENTS procedure shows the contents of a SAS data set and prints the directory of the SAS library.

Generally, the CONTENTS procedure functions the same as the CONTENTS statement in the DATASETS procedure. The differences between the CONTENTS procedure and the CONTENTS statement in PROC DATASETS are as follows:

- The default for *libref* in the DATA= option in PROC CONTENTS is Work. For the CONTENTS statement, the default is the libref of the procedure input library.
- PROC CONTENTS can read sequential files. The CONTENTS statement cannot.

Concepts: CONTENTS Procedure

Concepts: CONTENTS Procedure

See [Concepts for the CONTENTS Statement on page 405](#).

PROC CONTENTS reports metadata about the table and the metadata about the variables.

The CAS engine is the only SAS engine that supports VARCHAR. If there is a VARCHAR data type in the table, PROC CONTENTS shows the length in bytes and characters, as well as the maximum bytes used.

Just like with SAS engine data sets, the top portion of PROC CONTENTS reports the information about the table. The bottom portion of PROC CONTENTS (related to the variable metadata reports) is the metadata that is represented in the Compute Server session.

The Encoding shows the encoding of the CAS table. Based on what type of transcoding that might be needed to go from the CAS UTF-8 encoding to the session encoding, The variable byte length used in the session may differ from the byte length in the CAS table depending on the encoding of the session.

Output 12.1 Output of Mycas.French2 with VARCHAR

Data Set Name	MYCAS.FRENCH2	Observations	11
Member Type	DATA	Variables	2
Engine	CAS	Indexes	0
Created	08/16/2017 18:05:10	Observation Length	40
Last Modified	08/16/2017 18:05:10	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64		
Encoding	utf-8 Unicode (UTF-8)		

Engine/Host Dependent Information	
Data Limit	100MB
Caslib	CASUSER.
Scope	Session

Alphabetic List of Variables and Attributes					
#	Variable	Type	Len		Max Bytes Used
			Bytes	Chars	
1	nterm	Varchar	80	20	19
2	term	Char	19		

The following PROC CONTENTS shows the output from the Hadoop engine.

Output 12.2 PROC CONTENTS for the Hadoop Engine

The CONTENTS Procedure			
Data Set Name	X.CLASS	Observations	.
Member Type	DATA	Variables	5
Engine	HADOOP	Indexes	0
Created	07/13/2022 23:28:04	Observation Length	0
Last Modified	-	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	Default		
Encoding	Default		

Engine/Host Dependent Information	
Table Type	MANAGED_TABLE
Location	hdfs://fedvm171.unix.sas.com:8020/warehouse/tablespace/managed/hive/accesstesting.db/class
Partition Columns	.
Bucket Columns	.
Num Buckets	.
SerDe Library	org.apache.hadoop.hive.q1.io.orc.OrcSerde
Input Format	org.apache.hadoop.hive.q1.io.orc.OrcInputFormat
Output Format	org.apache.hadoop.hive.q1.io.orc.OrcOutputFormat

Alphabetic List of Variables and Attributes						
#	Variable	Type	Len	Format	Informat	Label
3	age	Num	8			age
4	height	Num	8			height
1	name	Char	24	\$24.	\$24.	name
2	sex	Char	3	\$3.	\$3.	sex
5	weight	Num	8			weight

CAUTION

Do not confuse the GENNUM variable value in CONTENTS' OUT= data set with the GEN variable value from DICTIONARY tables. GENNUM from a CONTENTS procedure or statement refers to a specific generation of a data set. GEN from DICTIONARY tables refers to the total number of generations for a data set. Each SAS procedure is designed and architected to deliver specific information. The output from DICTIONARY.TABLES and PROC CONTENTS are not interchangeable.

Syntax: CONTENTS Procedure

Restrictions:

You cannot use the WHERE option to affect the output because PROC CONTENTS does not process any observations.

When a SAS data file reaches the maximum observation count, SAS procedures that return an observation count (such as the PRINT procedure or the CONTENTS procedure) return a missing value, which is represented by a period (.), for the number of observations.

When using a SAS/ACCESS LIBNAME engine to access a database, some of the information that is available in the header of a SAS data set is not available.

Procedures like CONTENTS and DATASETS don't query the system tables so indexes, integrity constraints, number of observations, and so on will not be displayed.

Notes: When using the Parquet engine, the Compressed field contains COLUMNAR in the output. See [“COLUMN_COMPRESS= Data Set Option” in SAS Viya LIBNAME Engines for ORC and Parquet: Reference](#) for compression types.

The ATTRIB statement does not affect the PROC CONTENTS output. PROC CONTENTS reports the labels, informats, and formats on the actual member.

Tips: Complete documentation for the CONTENTS procedure is in [CONTENTS Statement on page 405](#). The links in the table below are to the DATASETS procedure documentation, which explains these options.

When using PROC CONTENTS, you can use data set options with the DATA=, OUT=, and OUT2= options.

The ORDER= option does not affect the order of the OUT= and OUT2= data sets.

PROC CONTENTS <options>;

Statement	Task	Example
CONTENTS	List the contents of one or more SAS data sets and print the directory of the SAS library	Ex. 1 , Ex. 2 , Ex. 3

PROC CONTENTS Statement

Describes the contents of one or more SAS data sets and prints the directory of the SAS library.

Restriction: You cannot use the WHERE option to affect the output because PROC CONTENTS does not process any observations.

Notes: The ATTRIB statement does not affect the CONTENTS statement output. CONTENTS reports the labels, informats, and formats on the actual member. When using the CONTENTS statement with SAS/ACCESS LIBNAME engines, see [“Differences in the DATASETS Procedure Output When Using SAS/ACCESS LIBNAME Engines” on page 375](#).

Tip: You can use data set options with the DATA=, OUT=, and OUT2= options. You can use any global statements as well.

Example: [“Example 5: Describing a SAS Data Set” on page 508](#)

Syntax

PROC CONTENTS <options>;

Optional Arguments

CENTILES

prints centiles information for indexed variables.

The following additional fields are printed in the default report of PROC CONTENTS when the CENTILES option is selected and an index exists on the data set. Note that the additional fields depend on whether the index is simple or complex.

#	number of the index on the data set.
Index	name of the index.
Update Centiles	percentage of the data values that must be changed before the CENTILES for the indexed variables are automatically updated.
Current Update Percentage	percentage of index updated since CENTILES were refreshed.
# of Unique Values	number of unique indexed values.
Variables	names of the variables used to make up the index. Centile information is listed below the variables.

DATA=SAS-file-specification

specifies an entire library or a specific SAS data set within a library. *SAS-file-specification* can take one of the following forms:

<libref.>SAS-data-set

names one SAS data set to process. The default for libref is the libref of the procedure input library. For example, to obtain the contents of the SAS data set HtWt from the procedure input library, use the following CONTENTS statement:

```
contents data=HtWt;
```

To obtain the contents of a specific version from a generation group, use the GENNUM= data set option as shown in the following CONTENTS statement:

```
contents data=HtWt (gennum=3) ;
```

<libref.>_ALL_

gives you information about all SAS data sets that have the type or types specified by the MEMTYPE= option. *libref* refers to the SAS library. The default for libref is the libref of the procedure input library.

- If you are using the _ALL_ keyword, you need Read access to all read-protected SAS data sets in the SAS library.
- DATA=_ALL_ automatically prints a listing of the SAS files that are contained in the SAS library. Note that for SAS views, all librefs that are associated with the views must be assigned in the current session in order for them to be processed for the listing.

Default most recently created data set in your job or session, from any SAS library.

Tip If you specify a read-protected data set in the DATA= option but do not give the Read password, by default the procedure looks in the PROC DATASETS statement for the Read password. However, if you do not specify the DATA= option and the default data set (last one created in the session) is Read protected, the procedure does not look in the PROC DATASETS statement for the Read password.

Example [“Example 5: Describing a SAS Data Set” on page 508](#)

DETAILS | NODETAILS

includes information in the output about the number of observations, number of variables, number of indexes, and data set labels. DETAILS includes these additional columns of information in the output, but only if DIRECTORY is also specified.

Default If neither DETAILS nor NODETAILS is specified, the defaults are as follows: for the CONTENTS procedure, the default is the system option setting, which is NODETAILS; for the CONTENTS statement, the default is whatever is specified in the PROC DATASETS statement, which also defaults to the system option setting.

See a description of the additional columns in the Optional Argument section of [PROC DATASETS Statement on page 382](#)

DIRECTORY

prints a list of all SAS files in the specified SAS library. If DETAILS is also specified, using DIRECTORY causes the additional columns described in [DETAILS | NODETAILS on page 295](#) to be printed.

ENCRYPTKEY=key-value

specifies the key value for AES encryption. For more information, see [“Library Contents and AES Encryption” on page 410](#).

FMTLEN

prints the length of the informat or format. If you do not specify a length for the informat or format when you associate it with a variable, the length does not appear in the output of the CONTENTS statement unless you use the FMTLEN option. The length also appears in the FORMATL or INFORML variable in the output data set.

MEMTYPE=(member-type(s))

restricts processing to one or more member types. The CONTENTS statement produces output only for member types DATA, VIEW, and ALL, which includes DATA and VIEW.

MEMTYPE= in the CONTENTS statement differs from MEMTYPE= in most of the other statements in the DATASETS procedure in the following ways:

- A slash does not precede the option.
- You cannot enclose the MEMTYPE= option in parentheses to limit its effect to only the SAS file immediately preceding it.

MEMTYPE= results in a directory of the library in which the DATA= member is located. However, MEMTYPE= does not limit the types of members whose contents are displayed unless the _ALL_ keyword is used in the DATA= option.

For example, the following statements produce the contents of only the SAS data sets with the member type DATA:

```
proc datasets memtype=data;
  contents data=_all_;
run;
```

Aliases MTYPE=

MT=

Default DATA

NODS

suppresses printing the contents of individual files when you specify `_ALL_` in the `DATA=` option. The `CONTENTS` statement prints only the SAS library directory. You cannot use the `NODS` option when you specify only one SAS data set in the `DATA=` option.

NOPRINT

suppresses printing the output of the `CONTENTS` statement.

ORDER=COLLATE | CASECOLLATE | IGNORECASE | VARNUM

COLLATE prints a list of variables in alphabetical order beginning with uppercase and then lowercase names.

CASECOLLATE prints a list of variables in alphabetical order even if they include mixed-case names and numerics.

IGNORECASE prints a list of variables in alphabetical order ignoring the case of the letters.

VARNUM is the same as the `VARNUM` option.

Note The `ORDER=` option does not affect the order of the `OUT=` and `OUT2=` data sets.

See [“VARNUM” on page 297](#)

Example See [“Example 3: Using the ORDER= Option” on page 302](#) to compare the default and the four options for `ORDER=`.

OUT=SAS-data-set

names an output SAS data set.

Tip `OUT=` does not suppress the printed output from the statement. If you want to suppress the printed output, you must use the `NOPRINT` option.

See [“The OUT= Data Set” on page 485](#) for a description of the variables in the `OUT=` data set.

OUT2=SAS-data-set

names the output data set to contain information about indexes and integrity constraints.

Note When you use the `OUT2=PermanentLibrary_ALL_` option within `PROC CONTENTS` or `PROC DATASETS` with the `CONTENTS` statement, you

must also set the REPLACE=YES data set option or the REPLACE system option.

Tips If UPDATECENTILES was not specified in the index definition, then the default value of 5 is used in the re-create variable of the OUT2 data set.

OUT2= does not suppress the printed output from the statement. To suppress the printed output, use the NOPRINT option.

See [“The OUT2= Data Set” on page 491](#) for a description of the variables in the OUT2= data set.

SHORT

prints only the list of variable names, the index information, and the sort information for the SAS data set.

Restriction If the list of variables is more than 32,767 characters, the list is truncated and a WARNING is written to the SAS log. To get a complete list of the variables, request an alphabetical listing of the variables.

VARNUM

prints a list of the variable names in the order of their logical position in the data set. By default, the CONTENTS statement lists the variables alphabetically. The physical position of the variable in the data set is engine-dependent.

Usage: CONTENTS Procedure

Using PROC CONTENTS

For information on using PROC CONTENTS with SAS data sets, see [“CONTENTS Statement” on page 405](#).

For more information, see [Chapter 4, “CAS Processing of Base Procedures,” on page 75](#).

Examples: CONTENTS Procedure

Example 1: Describing a SAS Data Set

Features: PROC CONTENTS statement options
DATA=
OUT=
OPTIONS statement
TITLE statement

Details

This example shows the output from the CONTENTS procedure for the Class data set. The output shows the modifications made to the Class data set in [“Example 4: Modifying SAS Data Sets”](#) on page 505 and the contents of the Class2 data set.

Program

```
options pagesize=40 linesize=80 nodate pageno=1;
LIBNAME v9 'SAS-library';
proc contents data=sashelp (read=green) out=Class3;
    title 'The Contents of the Class Data Set';
run;
proc contents data=shelp.class;
    title 'The Contents of the Class Data Set';
run;
```

Program Description

Set the system options. The PAGESIZE= option specifies the number of lines that compose a page of the SAS log and SAS output. The LINESIZE= option specifies the line size for the SAS log and for the SAS procedure output. The NODATE option

specifies that the date and the time are not printed. The PAGENO= option specifies a beginning page number for the next page of output.

```
options pagesize=40 linesize=80 nodate pageno=1;
```

Set your libref.

```
LIBNAME v9 'SAS-library';
```

Create the output data set Class3 from the data set Class. Give Read access to the Class data set.

```
proc contents data=sashelp (read=green) out=Class3;
  title 'The Contents of the Class Data Set';
run;
```

Display the contents of the Class data set.

```
proc contents data=shelp.class;
  title 'The Contents of the Class Data Set';
run;
```

Output Examples

Output 12.3 Contents of the Class Data Set

The Contents Using Directory	
The CONTENTS Procedure	
Directory	
Libref	WORK
Engine	V9
Physical Name	/opt/sas/viya/config/var/tmp/compsrv/default/c4f057e0-c534-4802-967d-5bf07f9a0901/SAS_work923C000002A6_launcher-b966616f-b389-4b4e-8f22-3c578bc9789c-bxwxt
Filename	/opt/sas/viya/config/var/tmp/compsrv/default/c4f057e0-c534-4802-967d-5bf07f9a0901/SAS_work923C000002A6_launcher-b966616f-b389-4b4e-8f22-3c578bc9789c-bxwxt
Inode Number	222690576
Access Permission	rwX-----
Owner Name	UNKNOWN
File Size	4KB
File Size (bytes)	4096

Engine/Host Dependent Information	
Data Set Page Size	73728
Number of Data Set Pages	1
First Data Page	1
Max Obs per Page	82
Obs in First Data Page	41
Number of Data Set Repairs	0
Filename	/opt/sas/viya/config/var/tmp/compsrv/default/c4f057e0-c534-4802-967d-5bf07f9a0901/SAS_work923C000002A6_launcher-b966616f-b389-4b4e-8f22-3c578bc9789c-bxwxt/class.sas7bdat
Release Created	V.0400M0
Host Created	Linux
Inode Number	222692708
Access Permission	rw-r--r--
Owner Name	UNKNOWN
File Size	144KB
File Size (bytes)	147456

Alphabetic List of Variables and Attributes					
#	Variable	Type	Len	Format	Label
32	CHARSET	Char	8		Host Character Set
33	COLLATE	Char	8		Collating Sequence
28	COMPRESS	Char	8		Compression Routine
20	CRDATE	Num	8	DATETIME16.	Create Date
22	DELOBS	Num	8		Deleted Observations in Data Set
36	ENCRYPT	Char	8		Encryption Routine
19	ENGINE	Char	8		Engine Name
27	FLAGS	Char	3		Update Flags (Protect Contribute Add)
10	FORMAT	Char	32		Variable Format
12	FORMATD	Num	8		Number of Format Decimals
11	FORMATL	Num	8		Format Length
38	GENMAX	Num	8		Maximum Number of Generations
40	GF				Number

Example 2: Using the DIRECTORY Option

Features:

- PROC CONTENTS statement options
 - DATA=
 - DIRECTORY
 - OUT=
- OPTIONS statement
- TITLE statement

Details

This example shows the output from the CONTENTS procedure for the Class3 data set.

Program

```
options pagesize=40 linesize=80 nodate pageno=1;

LIBNAME health 'SAS-library';

proc contents data=health.group (read=green) directory;
title 'Contents Using the DIRECTORY Option';
run;
```

Program Description

Set the system options. The PAGESIZE= option specifies the number of lines that compose a page of the SAS log and the SAS output. The LINESIZE= option specifies the line size for the SAS log and for the SAS procedure output. The NODATE option specifies that the date and the time are not printed. The PAGENO= option specifies a beginning page number for the next page of output.

```
options pagesize=40 linesize=80 nodate pageno=1;
```

Set your libref.

```
LIBNAME health 'SAS-library';
```

Specify Class3 as the data set to describe. Use the DIRECTORY option to print a listing of all the data sets that are in the HEALTH library.

```
proc contents data=health.group (read=green) directory;
title 'Contents Using the DIRECTORY Option';
run;
```

Output Examples

Output 12.4 Using the DIRECTORY Option - Section 1

The Contents Using Directory	
The CONTENTS Procedure	
Directory	
Libref	WORK
Engine	V9
Physical Name	/opt/sas/viya/config/var/tmp/compsrv/default/c4f057e0-c534-4802-967d-5bf07f9a0901/SAS_work923C000002A6_launcher-b966616f-b389-4b4e-8f22-3c578bc9789c-bxwxt
Filename	/opt/sas/viya/config/var/tmp/compsrv/default/c4f057e0-c534-4802-967d-5bf07f9a0901/SAS_work923C000002A6_launcher-b966616f-b389-4b4e-8f22-3c578bc9789c-bxwxt
Inode Number	222690576
Access Permission	rwX-----
Owner Name	UNKNOWN
File Size	4KB
File Size (bytes)	4096

Engine/Host Dependent Information	
Data Set Page Size	73728
Number of Data Set Pages	1
First Data Page	1
Max Obs per Page	82
Obs in First Data Page	41
Number of Data Set Repairs	0
Filename	/opt/sas/viya/config/var/tmp/compsrv/default/c4f057e0-c534-4802-967d-5bf07f9a0901/SAS_work923C000002A6_launcher-b966616f-b389-4b4e-8f22-3c578bc9789c-bxwxt/class3.sas7bdat
Release Created	V.0400M0
Host Created	Linux
Inode Number	222692710
Access Permission	rw-r--r--
Owner Name	UNKNOWN
File Size	144KB
File Size (bytes)	147456

Alphabetic List of Variables and Attributes					
#	Variable	Type	Len	Format	Label
32	CHARSET	Char	8		Host Character Set
33	COLLATE	Char	8		Collating Sequence
28	COMPRESS	Char	8		Compression Routine
20	CRDATE	Num	8	DATETIME16.	Create Date
22	DELOBS	Num	8		Deleted Observations in Data Set
36	ENCRYPT	Char	8		Encryption Routine
19	ENGINE	Char	8		Engine Name
27	FLAGS	Char	3		Update Flags (Protect Contribute Add)
10	FORMAT	Char	32		Variable Format
12	FORMATD	Num	8		Number of Format Decimals
11	FORMATL	Num	8		Format Length
38	GENMAX	Num	8		Maximum Number of Generations
40	GENNEXT	Num	8		Next Generation Number
39	GENNUM	Num	8		Generation Number
25	IDXCOUNT	Num	8		Number of Indexes for Data Set
23	IDXUSAGE	Char	9		Use of Variable in Indexes
13	INFORMAT	Char	32		Variable Informat
15	INFORMD	Num	8		Number of Informat Decimals

Example 3: Using the ORDER= Option

Features: PROC CONTENTS statement options

DATA=
ORDER=
OUT=

OPTIONS statement

TITLE statement

Details

This example shows the output from the CONTENTS procedure for the Grpout data set using the ORDER= option, which prints a list of variables in different orders.

Program

```
options pagesize=40 linesize=80 nodate pageno=1;

LIBNAME health 'SAS-library';

proc contents data=health.grpout order=collate;
title 'Contents Using the ORDER= Option';
run;

proc contents data=health.grpout order=varnum;
title 'Contents Using the ORDER= Option';
run;
```

Program Description

Set the system options. The PAGESIZE= option specifies the number of lines that compose a page of the SAS log and the SAS output. The LINESIZE= option specifies the line size for the SAS log and for the SAS procedure output. The NODATE option specifies that the date and the time are not printed. The PAGENO= option specifies a beginning page number for the next page of output.

```
options pagesize=40 linesize=80 nodate pageno=1;
```

Set your libref.

```
LIBNAME health 'SAS-library';
```

Specify the Grpout data set. Use the ORDER=COLLATE option to print a listing of all variables in alphabetical order.

```
proc contents data=health.grpout order=collate;
title 'Contents Using the ORDER= Option';
run;
```

Specify the Grpout data set. Use the ORDER=VARNUM option to print a listing of all variables in number order.

```
proc contents data=health.grpout order=varnum;
title 'Contents Using the ORDER= Option';
run;
```

Output Examples

Output 12.5 Using the ORDER=COLLATE Option

Alphabetic List of Variables and Attributes					
#	Variable	Type	Len	Format	Label
32	CHARSET	Char	8		Host Character Set
33	COLLATE	Char	8		Collating Sequence
28	COMPRESS	Char	8		Compression Routine
20	CRDATE	Num	8	DATETIME16.	Create Date
22	DELOBS	Num	8		Deleted Observations in Data Set
36	ENCRYPT	Char	8		Encryption Routine
19	ENGINE	Char	8		Engine Name
27	FLAGS	Char	3		Update Flags (Protect Contribute Add)
10	FORMAT	Char	32		Variable Format
12	FORMATD	Num	8		Number of Format Decimals
11	FORMATL	Num	8		Format Length
38	GENMAX	Num	8		Maximum Number of Generations
40	GENNEXT	Num	8		Next Generation Number
39	GENNUM	Num	8		Generation Number
25	IDXCOUNT	Num	8		Number of Indexes for Data Set
23	IDXUSAGE	Char	9		Use of Variable in Indexes
13	INFORMAT	Char	32		Variable Informat
15	INFORMD	Num	8		Number of Informat Decimals
14	INFORML	Num	8		Informat Length
16	JUST	Num	8		Justification

9	LABEL	Char	256		Variable Label
7	LENGTH	Num	8		Variable Length
1	LIBNAME	Char	8		Library Name
3	MEMLABEL	Char	256		Data Set Label
2	MEMNAME	Char	32		Library Member Name
24	MEMTYPE	Char	8		Library Member Type
21	MODATE	Num	8	DATETIME16.	Last Modified Date
5	NAME	Char	32		Variable Name
18	NOBS	Num	8		Observations in Data Set
34	NODUPKEY	Char	3		Sort Option: No Duplicate Keys
35	NODUPREC	Char	3		Sort Option: No Duplicate Records
17	NPOS	Num	8		Position in Buffer
37	POINTOBS	Char	3		Point to Observations
26	PROTECT	Char	3		Password Protection (Read Write Alter)
29	REUSE	Char	3		Reuse Space
30	SORTED	Num	8		Sorted and/or Validated
31	SORTEDBY	Num	8		Position of Variable in Sortedby Clause
41	TRANSCOD	Char	3		Character Variables Transcoded
6	TYPE	Num	8		Variable Type
4	TYPEMEM	Char	8		Special Data Set Type (From TYPE=)
8	VARNUM	Num	8		Variable Number

Output 12.6 Using the ORDER=VARNUM Option

Variables in Creation Order					
#	Variable	Type	Len	Format	Label
1	LIBNAME	Char	8		Library Name
2	MEMNAME	Char	32		Library Member Name
3	MEMLABEL	Char	256		Data Set Label
4	TYPEMEM	Char	8		Special Data Set Type (From TYPE=)
5	NAME	Char	32		Variable Name
6	TYPE	Num	8		Variable Type
7	LENGTH	Num	8		Variable Length
8	VARNUM	Num	8		Variable Number
9	LABEL	Char	256		Variable Label
10	FORMAT	Char	32		Variable Format
11	FORMATL	Num	8		Format Length
12	FORMATD	Num	8		Number of Format Decimals
13	INFORMAT	Char	32		Variable Informat
14	INFORML	Num	8		Informat Length
15	INFORMD	Num	8		Number of Informat Decimals
16	JUST	Num	8		Justification
17	NPOS	Num	8		Position in Buffer
18	NOBS	Num	8		Observations in Data Set
19	ENGINE	Char	8		Engine Name
20	CRDATE	Num	8	DATETIME16.	Create Date

21	MODATE	Num	8	DATETIME16.	Last Modified Date
22	DELOBS	Num	8		Deleted Observations in Data Set
23	IDXUSAGE	Char	9		Use of Variable in Indexes
24	MEMTYPE	Char	8		Library Member Type
25	IDXCOUNT	Num	8		Number of Indexes for Data Set
26	PROTECT	Char	3		Password Protection (Read Write Alter)
27	FLAGS	Char	3		Update Flags (Protect Contribute Add)
28	COMPRESS	Char	8		Compression Routine
29	REUSE	Char	3		Reuse Space
30	SORTED	Num	8		Sorted and/or Validated
31	SORTEDBY	Num	8		Position of Variable in Sortedby Clause
32	CHARSET	Char	8		Host Character Set
33	COLLATE	Char	8		Collating Sequence
34	NODUPKEY	Char	3		Sort Option: No Duplicate Keys
35	NODUPREC	Char	3		Sort Option: No Duplicate Records
36	ENCRYPT	Char	8		Encryption Routine
37	POINTOBS	Char	3		Point to Observations
38	GENMAX	Num	8		Maximum Number of Generations
39	GENNUM	Num	8		Generation Number
40	GENNEXT	Num	8		Next Generation Number
41	TRANSCOD	Char	3		Character Variables Transcoded

COPY Procedure

Overview: COPY Procedure	309
What Does the COPY Procedure Do?	309
In-Database Processing for PROC COPY	310
Syntax: COPY Procedure	311
PROC COPY Statement	313
ATTRIB Statement	320
EXCLUDE Statement	321
FORMAT Statement	322
LABEL Statement	323
SELECT Statement	324
WHERE Statement	326
Usage: COPY Procedure	328
CAS Processing for PROC COPY	328
Using CLONE NOCLONE on a CAS Table	329
Compressing Output CAS Tables	332
Using the COPY Procedure	333
Examples: COPY Procedure	335
Example 1: Copying SAS Data Sets between Hosts	335
Example 2: Converting SAS Data Sets Encodings	337
Example 3: Copy a SAS Data Set to a CAS Table	339

Overview: COPY Procedure

What Does the COPY Procedure Do?

The COPY procedure copies one or more tables from one SAS library to another.

Generally, the COPY procedure functions the same as the COPY statement in the DATASETS procedure. The differences are as follows:

- The IN= argument is required with PROC COPY. In the COPY statement, IN= is optional. If IN= is omitted, the default value is the libref of the procedure input library.
- PROC DATASETS cannot work with libraries that allow only sequential data access.
- The COPY statement honors the NOWARN option but PROC COPY does not.

Note: The MIGRATE procedure is available specifically for migrating a SAS library from a previous release to the most recent release. For migration, PROC MIGRATE offers benefits that PROC COPY does not. For more information, see [MIGRATE Procedure on page 1343](#).

With the ACCEL option, PROC COPY executes a CAS action to copy a CAS table from one caslib to another caslib in the same CAS session.

In-Database Processing for PROC COPY

In-database processing has several advantages over processing within SAS. These advantages include increased security, reduced network traffic, and the potential for faster processing. Increased security is possible because sensitive data does not have to be extracted from the data source. Faster processing is possible because data is manipulated locally, on the data source, using high-speed secondary storage devices instead of being transported across a relatively slow network connection. The data source is used because it might have more processing resources at its disposal, and it might be capable of optimizing a query for execution in a highly parallel and scalable fashion.

When the DATA= input table is stored as a table or view in a data source, PROC COPY can use in-database processing to perform most of its work within the data source. In-database processing can provide the advantages of faster processing and reduced data transfer between the data source and SAS.

In-database processing for PROC COPY supports the following data sources:

- Greenplum
- PostgreSQL
- Vertica

The COPY procedure performs a dynamic transformation of the data in which the characteristics of the output table, specifically the number and names of the variables, as well as their types, are determined from the variable values, as well as the characteristics of the input table. This dynamic behavior is achieved by a two-pass process in which rows of the input table are examined to determine the characteristics of the output table, during the first pass, and in which the work of transposing the data is performed during the second pass. Parallel operation within the massively parallel processing (MPP) database speeds up both the first and

second pass. The MPP performs a set of coordinated computations in parallel with the use of a large number of processors or separate computers. In the first pass, rows are examined in parallel and in place, according to the manner in which they are already partitioned across the nodes of a cluster. In the second pass, rows are repartitioned to form BY groups that are then processed independently and in parallel.

Both the first and second passes of the in-database processing are performed by executing a DS2 program within the SAS Embedded Process that resides within the nodes of the cluster. The DBMS provides the SAS Embedded Process with the ability to read data from and write data to tables. The SAS Embedded Process provides an execution context for the DS2 program. Because the two passes of work are expressed in the DS2 language, columns of the tables are cast to variables that have DS2 data types. Data type support within DS2 is more extensive than that provided by the traditional SAS system so, when executing inside the database, the ability of the COPY procedure to preserve data types and value of input data within the transposed output data is enhanced.

The SQLGENERATION system option or LIBNAME statement option controls whether and how in-database procedures are run inside the database. There are many programming considerations that can prevent in-database processing. For a complete listing, see [“Procedure Considerations and Limitations” in SAS In-Database Products: User’s Guide](#).

For examples of using in-database processing with PROC COPY, see the resources:

- [“Running PROC COPY In-Database for Greenplum” in SAS/ACCESS for Relational Databases: Reference](#)
- [“Running PROC COPY In-Database for PostgreSQL” in SAS/ACCESS for Relational Databases: Reference](#)
- [“Running PROC COPY In-Database for Vertica” in SAS/ACCESS for Relational Databases: Reference](#)

Syntax: COPY Procedure

Valid in:	Compute Server, CAS
Restrictions:	PROC COPY ignores concatenations with catalogs. Use PROC CATALOG COPY to copy concatenated catalogs. PROC COPY does not support data set options. PROC COPY does not back up graphic catalogs. Use PROC CPORT or PROC CIMPORT when doing back ups with graphic catalogs.
Interaction:	The International Components for Unicode (ICU) version is used to sort data sets with a linguistic collating sequence. If a linguistically sorted data set has a different ICU version number than that of the current session, the following occurs: PROC COPY retains the data set’s sort order in the OUT= destination library. However, the data set is no longer marked as sorted, and a message is written to the SAS log. For

more information about linguistic sorting, see [Chapter 55, “SORT Procedure,”](#) on page 2039.

Notes:

Data set names and variable names that end with large numeric values that are larger than a long integer cannot be used in numbered-range lists. For more information, see [“Definitions for SAS Variables”](#) in *SAS Programmer’s Guide: Essentials*..

PROC COPY uses a CAS action to provide the copy operation in the CAS server when both the IN= and OUT= values use the CAS libname engine and both libnames use the same CAS session. See [“Copying a CAS Table to Another CAS Table”](#) on page 422 and [“CAS Processing for PROC COPY”](#) on page 328.

Tips:

Complete documentation for the COPY procedure is in [COPY Statement](#) on page 413 in PROC DATASETS.

You can use the ATTRIB, FORMAT, LABEL, and WHERE statements with the PROC COPY procedure. For more information, see [“Statements with the Same Function in Multiple Procedures”](#) on page 23.

PROC COPY

```
<ACCEL | NOACCEL >
  OUT=libref-1
<CLONE | NOCLONE>
< CONSTRAINT=YES | NO >
< DATECOPY >
< ENCRYPTKEY=key-value >
< FORCE >
  IN=libref-2
< INDEX=YES | NO >
< MEMTYPE=(member-type(s)) >
< MOVE < ALTER=alter-password >>
< OVERRIDE=(ds-option-1=value-1 <ds-option-2=value-2 ...> ) >;
```

ATTRIB *variable-list(s) attribute-list(s);*

EXCLUDE *SAS-file(s) < / MEMTYPE=member-type >;*

FORMAT *variable-1 <...variable-n> <format> <DEFAULT=default-format>;*

LABEL *variable-1=label-1<...variable-n=label-n>;*

```
SELECT SAS-file(s)
  < / < ENCRYPTKEY=key-value > < ALTER=alter-password > <
  MEMTYPE=member-type > >;
```

```
WHERE where-expression-1
  < logical-operator where-expression-n>;
```

Statement	Task	Example
COPY	Copy one or more files	Ex. 1
EXCLUDE	Exclude files or memtypes	Ex.
SELECT	Select files or memtypes	Ex. 1, Ex. 2

PROC COPY Statement

Copies all or some of the SAS files in a SAS library.

- Restriction:** The COPY statement does not support data set options.
- Notes:** When using the COPY statement with SAS/ACCESS LIBNAME engines, see [“Differences in the DATASETS Procedure Output When Using SAS/ACCESS LIBNAME Engines” on page 375](#).
For CAS engine specifics, see [“CAS Processing for PROC COPY” on page 328](#).
- Tip:** The COPY statement defaults to the encoding and data representation of the output library when you use Remote Library Services (RLS) such as SAS/SHARE or SAS/CONNECT. If you are not using RLS, you must use the PROC COPY option NOCLONE for the output files to take on the encoding and data representation of the output library. Using the NOCLONE option results in a copy with the data representation of the data library (if specified in the OUTREP= LIBNAME option) or the native data representation of the operating environment.
- Example:** [“Example 2: Manipulating SAS Files” on page 497](#)

Syntax

```
PROC COPY <ACCEL | NOACCEL >OUT=libref-1
<CLONE | NOCLONE>
< CONSTRAINT=YES | NO >
< DATECOPY >
< ENCRYPTKEY=key-value >
< FORCE >
IN=libref-2
< INDEX=YES | NO >
< MEMTYPE=(member-type(s)) >
< MOVE < ALTER=alter-password >>
< OVERRIDE=(ds-option-1=value-1 <ds-option-2=value-2 ...> ) >;
```

Required Argument

OUT=libref-1

names the SAS library to copy SAS files to.

Aliases OUTLIB=

OUTDD=

Example [“Example 2: Manipulating SAS Files” on page 497](#)

Optional Arguments

ACCEL | NOACCEL

specifies whether to perform the copy operation in CAS. Both the IN= and OUT= libraries must be CAS engine libraries and they must use the same CAS session.

Default ACCEL

See [“Copying a CAS Table to Another CAS Table” on page 422](#)

CLONE | NOCLONE

specifies whether to copy the following data set attributes:

- size of input/output buffers
- whether the data set is compressed
- whether free space is reused
- data representation of input data set, library, or operating environment
- encoding value
- whether a compressed data set can be randomly accessed by an observation number

These attributes are specified with data set options, SAS system options, and LIBNAME statement options:

- BUFSIZE= value for the size of the input/output buffers
- COMPRESS= value for whether the data set is compressed
- REUSE= value for whether free space is reused
- OUTREP= value for data representation
- ENCODING= or INENCODING= for encoding value
- POINTOBS= value for whether a compressed data set can be randomly accessed by an observation number

For the BUFSIZE= attribute, the following table summarizes how the COPY statement works:

Table 13.1 CLONE and the Buffer Page Size Attribute

Option	COPY Statement
CLONE	Uses the BUFSIZE= value from the input data set for the output data set. However, specifying BUFSIZE= value in the OVERRIDE= option list results in a copy that uses the specified value.
NOCLONE	Uses the current setting of the SAS system option BUFSIZE= for the output data set.
Neither	Determines the type of access method, sequential or random, used by the engine for the input data set and the engine for the output data set. If both engines use the

Option	COPY Statement
	same type of access, the COPY statement uses the BUFSIZE= value from the input data set for the output data set. If the engines do not use the same type of access, the COPY statement uses the setting of SAS system option BUFSIZE= for the output data set.

For the COMPRESS= attribute, the following table summarizes how the COPY statement works:

Table 13.2 CLONE and the Compression Attribute

Option	COPY Statement
CLONE	Uses the values from the input data set for the output data set. However, specifying COMPRESS= value in the OVERRIDE= option list results in a copy that uses the specified encoding.
NOCLONE	Results in a copy with the compression of the operating environment or, if specified, the value of the COMPRESS= option in the LIBNAME statement for the library.
Neither	Defaults to CLONE.

For the REUSE= attribute, the following table summarizes how the COPY statement works:

Table 13.3 CLONE and the Reuse Space Attribute

Option	COPY Statement
CLONE	Uses the values from the input data set for the output data set. If the engine for the input data set does not support the reuse space attribute, then the COPY statement uses the current setting of the corresponding SAS system option. However, specifying REUSE= value in the OVERRIDE= option list results in a copy that uses the specified value.
NOCLONE	Uses the current setting of the SAS system options COMPRESS= and REUSE= for the output data set.
Neither	Defaults to CLONE.

For the OUTREP= attribute, the following table summarizes how the COPY statement works:

Table 13.4 *CLONE and the Data Representation Attribute*

Option	COPY Statement
CLONE	Results in a copy with the same data representation of the input data set. However, if you specify an OUTREP= value in the OVERRIDE= option list, the data representation is changed, and its encoding will change to an encoding that is compatible with the data representation and the locale of the current session. The encoding will also change if it is specified in the OVERRIDE= option list.
NOCLONE	Results in a copy with the data representation of the operating environment or, if specified, the value of the OUTREP= option in the LIBNAME statement for the OUT= library.
Neither	Defaults to CLONE.

Data representation is the form in which data is stored in a particular operating environment. Different operating environments use the following different standards or conventions:

- for storing floating-point numbers (for example, IEEE or IBM 390)
- for character encoding (ASCII or EBCDIC)
- for the ordering of bytes in memory (big Endian or little Endian)
- for word alignment (4-byte boundaries or 8-byte boundaries)
- for data-type length (16-bit, 32-bit, or 64-bit)

Native data representation is when the data representation of a file is the same as the CPU operating environment.

For the ENCODING= attribute, the following table summarizes how the COPY statement works.

Table 13.5 *CLONE and the Encoding Attribute*

Option	COPY Statement
CLONE	Results in a copy that uses the encoding of the input data set or, if specified, the value of the INENCODING= option in the LIBNAME statement for the input library. However, specifying ENCODING= value in the OVERRIDE= option list results in a copy that uses the specified encoding.
NOCLONE	Results in a copy that uses the encoding of the current session encoding or, if specified, the value of the OUTENCODING= option in the LIBNAME statement for the output library.

Option	COPY Statement
Neither	Defaults to CLONE.

All text (character) data that is stored, transmitted, or processed by a computer is in an encoding. An encoding maps each character to a unique numeric representation. An encoding is a combination of a character set with an encoding method. A character set is the repertoire of characters and symbols that are used by a language or group of languages. An encoding method is the set of rules that are used to assign the numbers to the set of characters that are used in an encoding.

For the POINTOBS= attribute, the following table summarizes how the COPY statement works. To use POINTOBS=, the output data set must be compressed.

Table 13.6 CLONE and the POINTOBS= Attribute

Option	COPY Statement
CLONE	Uses the POINTOBS= value from the input data set for the output data set. However, specifying POINTOBS= value in the OVERRIDE= option list results in a copy that uses the specified value.
NOCLOSE	Uses the LIBNAME statement if the output data set is compressed and the POINTOBS= option is specified and supported by the output engine. If the LIBNAME statement is not specified and the data set is compressed, the default is POINTOBS=YES when supported by the output engine.
Neither	Defaults to CLONE.

CONSTRAINT=YES | NO

specifies whether to copy all integrity constraints when copying a data set.

Default NO

Tip For data sets with integrity constraints that have a foreign key, the COPY statement copies the general and referential constraints if CONSTRAINT=YES is specified and the entire library is copied. If you use the SELECT or EXCLUDE statement to copy the data sets, then the referential integrity constraints are not copied.

DATECOPY

copies the SAS internal date and time at which the SAS file was created and when it was last modified to the resulting copy of the file. Note that the operating environment date and time are not preserved.

Restrictions DATECOPY cannot be used with encrypted files or catalogs.

DATECOPY can be used only when the resulting SAS file uses the V8 or V9 engine.

Tips

You can alter the file creation date and time with the DTC= option in the MODIFY statement. See [MODIFY statement on page 446](#).

If the file that you are copying has attributes that require additional processing, the last modified date is changed to the current date. For example, when you copy a data set that has an index, the index must be rebuilt, and the last modified date changes to the current date. Other attributes that require additional processing and that could affect the last modified date include integrity constraints and a sort indicator.

ENCRYPTKEY= *key-value*

specifies the key value needed to copy data sets in the IN= library that have AES encryption.

Note If the output library does not support AES-encryption and the input data set is AES-encrypted, the COPY process will produce an error.

See [“Copying AES-Encrypted Data Files Containing Referential Integrity Constraints ” on page 426](#)

FORCE

enables you to use the MOVE option for a SAS data set on which an audit trail exists.

Note The AUDIT file is not moved with the audited data set.

IN=libref-2

names the SAS library containing SAS files to copy.

Aliases INLIB=

INDD=

Default the libref of the procedure input library

Interaction To copy only selected members, use the SELECT or EXCLUDE statements.

INDEX=YES | NO

specifies whether to copy all indexes for a data set when copying the data set to another SAS library.

Default YES

MEMTYPE=(*member-type(s)*)

restricts processing to one or more member types.

Aliases MTYPE=

MT=

- Default** If you omit MEMTYPE= in the PROC DATASETS statement, the default is MEMTYPE=ALL.
- Note** When PROC COPY processes a SAS library on tape and the MEMTYPE= option is not specified, it scans the entire sequential library for entries until it reaches the end-of-file. If the sequential library is a multivolume tape, all tape volumes are mounted. This behavior is also true for single-volume tape libraries.
- See** [“Specifying Member Types When Copying or Moving SAS Files” on page 423](#) and [“Member Types” on page 470](#)
- Example** [“Example 2: Manipulating SAS Files” on page 497](#)

MOVE

moves SAS files from the input data library (named with the IN= option) to the output data library (named with the OUT= option). And deletes the original files from the input data library.

- Restriction** The MOVE option can be used to delete a member of a SAS library only if the IN= engine supports the deletion of tables. A tape format engine does not support table deletion. If you use a tape format engine, SAS suppresses the MOVE operation and prints a warning.
- See** [“Using Passwords with the DATASETS Procedure” on page 467](#)
- Example** [“Example 2: Manipulating SAS Files” on page 497](#)

ALTER=*alter-password*

provides the Alter password for any alter-protected SAS files that you are moving from one data library to another. Because the MOVE option deletes the SAS file from the original data library, you need Alter access to move the SAS file.

OVERRIDE=(*ds-option-1=value-1* <*ds-option-2=value-2*> ...)

overrides specified output data set options copied from the input data set. Some data set options might not be appropriate in the output data set context of COPY.

- Restriction** The OVERRIDE option is ignored if the NOCLONE option is specified. However, it can be used to modify data set attributes other than those controlled by the NOCLONE option.
- Interaction** When you specify an OUTREP= value in the OVERRIDE= option, the default encoding is based on the operating environment that is represented by the OUTREP= value and the locale of the current session. To assign a nondefault encoding such as UTF-8, you must also specify an ENCODING= value in the OVERRIDE=option. For more information about locale and encoding, see [SAS National Language Support \(NLS\): Reference Guide](#).
- Tip** The default (or CLONE) behavior is to preserve many attributes, including the data representation and encoding, from the input data set. If instead you want the data representation and encoding of the

session that is executing the procedure, then specify `OVERRIDE=(OUTREP=SESSION ENCODING=SESSION)` in `PROC COPY` or in the `COPY` statement. If you do not want to preserve any attributes from the input data set, then specify `NOCLONE` instead.

ATTRIB Statement

Associates a format, informat, label, and length with one or more variables.

Syntax

ATTRIB *variable-list(s) attribute-list(s);*

Arguments

variable-list(s)

names the variables that you want to associate with the attributes.

TIP List the variables in any form that SAS allows.

attribute-list(s)

specifies one or more attributes to assign to *variable-list*. Specify one or more of these attributes in the `ATTRIB` statement:

FORMAT=*format*

associates a format with variables in *variable-list*.

Tip The format can be either a standard SAS format or a format that is defined with the `FORMAT` procedure.

INFORMAT=*informat*

associates an informat with variables in *variable-list*.

Tip The informat can be either a standard SAS informat or an informat that is defined with the `FORMAT` procedure.

LABEL=*'label'*

associates a label with variables in *variable-list*.

See Also

“`ATTRIB` Statement” in *SAS DATA Step Statements: Reference*.

EXCLUDE Statement

Excludes SAS files from copying.

Restrictions: The EXCLUDE statement must follow a COPY statement
The EXCLUDE statement cannot appear in the same COPY step with a SELECT statement

Example: [“Example 2: Manipulating SAS Files” on page 497](#)

Syntax

EXCLUDE *SAS-file(s)* </ MEMTYPE=*member-type*>;

Required Argument

SAS-file(s)

specifies one or more SAS files to exclude from the copy operation. All SAS files you name in the EXCLUDE statement must be in the library that is specified in the IN= option in the COPY statement. If the SAS files are generation groups, the EXCLUDE statement allows only selection of the base versions.

You can use the following shortcuts to list several SAS files in the EXCLUDE statement:

Table 13.7 Using the EXCLUDE Statement

Notation	Meaning
x1–xn	Specifies files X1 through Xn. The numbers must be consecutive.
x:	Specifies all files that begin with the letter X.

Optional Argument

MEMTYPE=member-type

restricts processing to one member type. You can use the option either in parentheses after the name of each SAS file or after a forward slash.

Aliases MTYPE=

MT=

Default If you do not specify MEMTYPE= in the PROC DATASETS statement, the COPY statement, or in the EXCLUDE statement, the default is MEMTYPE=ALL.

See [“Specifying Member Types When Copying or Moving SAS Files” on page 423](#) and [“Restricting Member Types for Processing” on page 469](#)

Details

Excluding Several Like-Named Files

You can use shortcuts for listing several SAS files in the EXCLUDE statement.

FORMAT Statement

Associates formats with variables.

Syntax

FORMAT *variable-1* <...*variable-n*> *format variable-1* <...*variable-n*> *format*;

Arguments

variable

names one or more variables for SAS to associate with a format. You must specify at least one *variable*.

Tip To disassociate a format from a variable, use the variable in a FORMAT statement without specifying a format in a DATA step or in PROC DATASETS. In a DATA step, place this FORMAT statement after the SET statement. See [“Removing a Format” in SAS DATA Step Statements: Reference](#). You can also use PROC DATASETS.

format

specifies the format that is listed for writing the values of the variables.

Tip Formats that are associated with variables by using a FORMAT statement behave like formats that are used with a colon modifier in a subsequent PUT statement. For more information about using a colon modifier, see [“PUT Statement: List” in SAS DATA Step Statements: Reference](#).

See [SAS Formats and Informats: Reference](#)

LABEL Statement

Assigns descriptive labels to variables.

Syntax

LABEL *variable-1* = '*text-string*' <...*variable-n* = '*text-string*'>;

LABEL *variable-1* = '' <...*variable-n*=''>; | *variable-1* = <...*variable-n*= >;

Arguments

variable

specifies the variable that you want to label. You can specify labels for multiple variables in a single LABEL statement:

```
data test;
  L = 5;
  W = 10;
  label L = 'Length' W = 'Width';
run;
```

text-string

specifies a label of up to 256 bytes.

```
data test;
  L = 5;
  label L = 'Length';
run;
```

Restrictions If the label includes a semicolon (;) or an equal sign (=), you must enclose the label in either single or double quotation marks.

```
label N = 'Number = ';
```

If the label includes single quotation marks ('), you must enclose the label in double quotation marks.

```
label Name = "Owner's Last Name";
```

There is partial support for variable name substitution with the JAVAIMG device. When the label text for the variable exceeds the space allowed, the variable name is used instead by procedures that are labeling an axis or a legend title. With the exception of the SAS/GRAPH GPLOT procedure, the variable name is not used when the JAVAIMG device is specified.

Notes Quotation marks are not required unless the label includes a semicolon, quotation marks, or an equal sign.

The LABEL statement expects an equal sign (=) after the first variable. If any other character is found, an error occurs. The LABEL statement considers everything after the equal sign, following the first variable, as part of the label until the statement comes to another variable followed by an equal sign, regardless of quotation marks. In this example, the label for x is **this is x y 4 = this is y** because the next variable that is followed by an equal sign is z.

```
data test;
  x = 1;
  y = 1;
  z = 1;
  label x = 'this is x' y 4 = 'this is y' z = 'This is z';
run;
```

In this LABEL statement, the x variable has the same label:

```
label x = this is x y 4 = this is y z = This is z;
```

See

[“Character Constants and Character Variables” in SAS Programmer’s Guide: Essentials.](#)

”

removes a label from a variable. Enclose a single blank space in quotation marks to remove an existing label. You can remove labels from multiple variables in a single LABEL statement:

```
data test;
  L=5; W=10;
  label L = ' ' W = ' ';
run;
```

Alternatively, you can remove a label by specifying nothing after the equal sign:

```
data test;
  L=5; W=10;
  label L = W = ;
run;
```

SELECT Statement

Selects SAS files for copying.

Restrictions:

The SELECT statement must follow a COPY statement

The SELECT statement cannot appear with an EXCLUDE statement in the same COPY step

Example:

[“Example 2: Manipulating SAS Files” on page 497](#)

Syntax

SELECT *SAS-file(s)*

```
</ <ENCRYPTKEY=key-value> <ALTER=alter-password> <MEMTYPE=member-type>>;
```

Required Argument

SAS-file(s)

specifies one or more SAS files that you want to copy. All of the SAS files that you name must be in the data library that is referenced by the libref named in the IN= option in the COPY statement. If the SAS files have generation groups, all the generations are copied because the SELECT statement does not allow you to select specific versions.

Optional Arguments

ALTER=*alter-password*

provides the Alter password for any alter-protected SAS files that you are moving from one data library to another. Because you are moving and thus deleting a SAS file from a SAS library, you need Alter access. You can use the option either in parentheses after the name of each SAS file or after a forward slash.

See [“Using Passwords with the DATASETS Procedure” on page 467](#)

ENCRYPTKEY=*key-value*

specifies the key value for AES encryption.

See [“Appending AES-Encrypted Data Sets” on page 395](#)

MEMTYPE=*member-type*

restricts processing to one member type. You can use the option either in parentheses after the name of each SAS file or after a forward slash.

Aliases MTYPE=

MT=

Default If you do not specify the MEMTYPE= option in the PROC DATASETS statement, in the COPY statement, or in the SELECT statement, the default is MEMTYPE=ALL.

See [“Specifying Member Types When Copying or Moving SAS Files” on page 423](#) and [“Restricting Member Types for Processing” on page 469](#)

Example [“Example 2: Manipulating SAS Files” on page 497](#)

Details

Selecting Several Like-Named Files

You can use shortcuts for listing several SAS files in the SELECT statement:

Table 13.8 *Using the SELECT Statement*

Notation	Meaning
x1–xn	Specifies files X1 through Xn. The numbers must be consecutive.
x:	Specifies all files that begin with the letter X.

WHERE Statement

Subsets the input data set by specifying certain conditions that each observation must meet before it is available for processing.

Syntax

WHERE *where-expression-1*
< *logical-operator where-expression-n*>;

Arguments

where-expression

is an arithmetic or logical expression that generally consists of a sequence of operands and operators.

Tips The operands and operators described in the next several sections are also valid for the WHERE= data set option.

You can specify multiple where-expressions.

logical-operator

can be AND, AND NOT, OR, or OR NOT.

Example

Procedures That Support the WHERE Statement

You can use the WHERE statement with any of the following Base SAS procedures that read a SAS data set:

COMPARE	REPORT
CORR	SORT
DATASETS (APPEND statement)	SQL
FREQ	STANDARD
MEANS or SUMMARY	TABULATE
PRINT	TRANSPOSE
RANK	UNIVARIATE

Details

- The COMPARE procedure and the APPEND statement in PROC DATASETS accept more than one input data set. For more information, see the documentation for the specific procedure.
- To subset the output data set, use the WHERE= data set option:

```
proc report data=debate nowd
           out=onlyfr (where=(year='1')) ;
run;
```

For more information about WHERE=, see [SAS DATA Step Statements: Reference](#).

Temporarily Dissociating a Format from a Variable

In this example, PROC PRINT prints only those observations that meet the condition of the WHERE expression.

```

options nodate pageno=1 linesize=64
      pagesize=40;

proc print data=debate noobs;
  where gpa>3.5;
  title 'Team Members with a GPA';
  title2 'Greater than 3.5';
run;

```

Output

Team Members with a GPA Greater than 3.5			
Name	Gender	Year	GPA
Capiccio	m	Freshman	3.598
Tucker	m	Freshman	3.901
Bagwell	f	Sophomore	3.722
Gold	f	Junior	3.609
Syme	f	Junior	3.883
Baglione	f	Senior	4.000
Carr	m	Senior	3.750
Hall	m	Senior	3.574

Usage: COPY Procedure

CAS Processing for PROC COPY

When the IN= option and OUT= option both reference a CAS engine library and both libraries use the same CAS session, the COPY procedure can use a CAS action to copy tables within the server.

PROC COPY options are valid when using a CAS LIBNAME engine except for the following:

- ENCRYPTKEY=

- OVERRIDE=
- PW=

When a copy occurs on the CAS server, the MVA session system options (like VALIDMEMNAME and VALIDVARNAME) will not be used.

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. The CAS LIBNAME engine can connect a Compute Server session to an existing SAS Cloud Analytic Services session through the CAS session name or the CAS session UUID. The libref then becomes your handle to communicate from SAS with the specific session. The following example shows how to use PROC COPY with CAS processing.

Here is an example of how to run PROC COPY with CAS.

```
/* Connect to a CAS server */
cas casauto host="cloud.example.com" port=5570;

/* Specify the LIBNAME statements. */
libname public cas caslib=public;
libname mycas cas caslib=casuser;

/* Loading data to Casuser. Note that you can specify OUTCASLIB=
on the PROC CASUTIL statement and it will apply to all statements
that follow in the procedure step (up to the quit). */
proc casutil;
load data=sashelp.class outcaslib=casuser;
load data=sashelp.class compress casout="class_comp" outcaslib=casuser;
load data=sashelp.class promote casout="class_prom" outcaslib=casuser;
quit;

/* Execute the PROC COPY statement. */
title 'Copy with CLONE;
proc copy in=casuser out=public;
run;
```

For information about how to use the CAS LIBNAME statement, see “Getting Started” in *SAS Cloud Analytic Services: User’s Guide*.

Using CLONE | NOCLONE on a CAS Table

The CLONE | NOCLONE option specifies whether to copy data set attributes. The only attribute that can be used with the CAS engine is COMPRESS.

Attributes are specified with data set options, system options, or LIBNAME statement options. The CAS engine supports only the COMPRESS=YES | NO option. No other attributes are supported by the CAS engine.

The following table summarizes how the COPY statement works:

Table 13.9 CLONE Interaction with Attributes

Attribute	To	CLONE or NOCLONE	Description
BUFSIZE= CAS engine does not support.	SAS data set to CAS table		
		CLONE	OVERWRITE=(BUFSIZE= <i>value other than default</i>)
		NOCLONE	Uses setting of BUFSIZE= system option
COMPRESS= SAS data set - COMPRESS=BINARY CHAR NO CAS table - COMPRESS=NO YES	SAS data set to CAS table		
		CLONE	A compressed SAS data set becomes a compressed CAS table unless OVERWRITE= is used. An uncompressed SAS data set becomes an uncompressed CAS table unless OVERWRITE= is used.
		NOCLONE	Follows the CAS LIBNAME setting.
	CAS table to SAS data set	CLONE	Compressed CAS table becomes a SAS data set CHAR unless the OVERWRITE= is used.
		NOCLONE	The COMPRESS= system option or LIBNAME option value is used.
	CAS table to CAS table	CLONE	Keeps the current setting.
		NOCLONE	Set using the CAS LIBNAME setting for the OUT= libref.
REUSE=	SAS data set to CAS table		

Attribute	To	CLONE or NOCLONE	Description
CAS engine does not support.	CAS table to SAS data set	CLONE	REUSE=NO unless OVERRIDE= or REUSE=YES system option is used.
		NOCLONE	Uses the REUSE= system option value.
	CAS table to CAS table		
POINTOBS= CAS engine does not support.	SAS data set to CAS table		
	CAS table to SAS data set	CLONE	POINTOBS=NO unless OVERRIDE= is used.
		NOCLONE	POINTOBS=NO, if the CAS table is compressed and the LIBNAME statement has POINTOBS=NO. POINTOBS=YES, if the CAS table is compressed and the LIBNAME option is missing.
OUTREP= CAS engine does not support.	CAS table to CAS table		
	SAS data set to CAS table	CLONE	Converts to LINUX_86_64 if needed. (A warning is sent to the log if the OVERRIDE= option is used.)
		NOCLONE	Converts to LINUX_86_64 if needed.
	CAS table to SAS data set		

Attribute	To	CLONE or NOCLONE	Description
		CLONE	Keeps data representation. (A warning is sent to the log if the OVERRIDE= option is used.)
	CAS table to CAS table		
ENCODING=	SAS data set to CAS table	CLONE	Converts to UTF-8 if needed. (A warning is sent to the log if the OVERRIDE= option is used.)
CAS engine supports UTF-8 only.			
Changing encoding is not implemented for the CAS engine.			
		NOCLONE	Converts to UTF-8 if needed.
	CAS table to SAS data set	CLONE	Keeps the UTF-8 encoding unless OVERRIDE= is used.
		NOCLONE	Keeps the UTF-8 encoding unless OUTENCODING= is used in the output data set LIBNAME is used.
	CAS table to CAS table		

Compressing Output CAS Tables

When copying previously compressed tables, the following occurs:

- if a SAS data set is compressed, then it retains the COMPRESS=YES value on the CAS table.
- if a CAS table is compressed, then it converts to a SAS data set with the COMPRESS=CHAR value.
- if a CAS table is copied to another CAS table, the COMPRESS=YES attribute is maintained.

Using the COPY Procedure

Copying Select Files from a Large Directory of Files

When using the COPY procedure, an in-memory directory of the library is obtained. This can be a performance issue if the library has thousands of members and only a few members are being copied. To resolve this performance issue, use a combination of the MEMTYPE= option in the COPY statement with a SELECT statement. The following is an example of this process:

```
proc copy in=work out=mylib memtype=(data catalog);  
    select mydata x1-x10 data2;  
run;
```

Note: If either MEMTYPE=ALL or a wildcard specification (":") is used, the performance code cannot be used.

TIP If the MSGLEVEL=I option is set, and the SELECT performance code can be used, the following message is sent to the SAS log:

```
INFO: COPY with SELECT performance is in use.
```

Transporting SAS Data Sets between Hosts

The COPY procedure, along with the XPORT engine and the XML engine, can create and read transport files that can be moved from one host to another. PROC COPY can create transport files only with SAS data sets, not with catalogs or other types of SAS files.

Transporting is a three-step process:

- 1 Use PROC COPY to copy one or more SAS data sets to a file that is created with either the transport (XPORT) engine or the XML engine. This file is referred to as a transport file and is always a sequential file.
- 2 After the file is created, you can move it to another operating environment via communications software, such as FTP, or tape. If you use communications software, be sure to move the file in binary format to avoid any type of conversion. If you are moving the file to a mainframe, the file must have certain

attributes. Consult the SAS documentation for your operating environment and the SAS Technical Support web page for more information.

- 3 After you have successfully moved the file to the receiving host, use PROC COPY to copy the data sets from the transport file to a SAS library.

For an example, see [“Example 1: Copying SAS Data Sets between Hosts” on page 335](#).

For details about transporting files, see *Moving and Accessing SAS Files*.

The CPORT and CIMPORT procedures also provide a way to transport SAS files. For more information, see [Chapter 14, “CPORT Procedure,” on page 341](#) and [Chapter 10, “CIMPORT Procedure,” on page 179](#).

If you need to migrate a SAS library from a previous release of SAS, see the Migration focus area at <http://support.sas.com/migration>.

For more information, see the [Details on page 420](#) section of the CONTENTS statement in PROC DATASETS.

Copying AES-Encrypted Data Files

You must use the ENCRYPTKEY= data set option when copying an AES-encrypted data file. An error occurs when you copy an AES-encrypted file to a library that does not support AES encryption. For more information about AES encryption, see [“AES Encryption” in SAS Programmer’s Guide: Essentials](#) and see [“Copying AES-Encrypted Data Files” on page 426](#).

The following is an example using the ENCRYPTKEY= data set option:

```
proc copy in=Lib1 out=Lib2;
select My-Data1 (encryptkey=key-value1) <My-Data2 (encryptkey=key-value2) ...>;
run;
```

To copy AES-encrypted data files containing referential integrity constraints, see [“Copying AES-Encrypted Data Files Containing Referential Integrity Constraints” on page 426](#).

Compressing Output Data Files

The COPY procedure does not support data set options. Therefore, you cannot use the COMPRESS= data set option in PROC COPY or a COPY statement from PROC DATASETS. To compress an OUTPUT data set generated by PROC COPY, you can use the COMPRESS=YES system option before the PROC COPY statement with the NOCLONE option.

```
options compress=yes;
proc copy in=work out=new noclone;
select x;
run;
```

Examples: COPY Procedure

Example 1: Copying SAS Data Sets between Hosts

Features: PROC COPY statement options
IN=
MEMTYPE=
OUT=
SELECT statement
XPORT engine

Details

This example demonstrates how to create a transport file on a host and read it on another host.

In order for this example to work correctly, the transport file must have certain characteristics, as described in the SAS documentation for your operating environment. In addition, the transport file must be moved to the receiving operating system in binary format.

Program

```
libname source 'SAS-library-on-sending-host';

libname xptout xport 'filename-on-sending-host';

proc copy in=source out=xptout memtype=data;
  select bonus budget salary;
run;

libname insource xport 'filename-on-receiving-host';
```

```
proc copy in=insource out=work;  
run;
```

Program Description

Assign library references. Assign a libref, such as Source, to the SAS library that contains the SAS data set that you want to transport. Also, assign a libref to the transport file and use the XPORT keyword to specify the XPORT engine.

```
libname source 'SAS-library-on-sending-host';  
  
libname xptout xport 'filename-on-sending-host';
```

Copy the SAS data sets to the transport file.

```
proc copy in=source out=xptout memtype=data;  
    select bonus budget salary;  
run;
```

Enable the procedure to read data from the transport file. The XPORT engine in the LIBNAME statement enables the procedure to read the data from the transport file.

```
libname insource xport 'filename-on-receiving-host';
```

Copy the SAS data sets to the receiving host. After you copy the files, use PROC COPY to copy the SAS data sets to the Work data library on the receiving host.

```
proc copy in=insource out=work;  
run;
```

Log Examples

Example Code 13.1 Source Library Log

```

1  LIBNAME source 'SAS-library-on-sending-host ';
NOTE: Libref SOURCE was successfully assigned as follows:
      Engine:          V9
      Physical Name:   SAS-library-on-sending-host
2  LIBNAME xptout xport 'filename-on-sending-host';
NOTE: Libref XPTOUT was successfully assigned as follows:
      Engine:          XPORT
      Physical Name:   filename-on-sending-host
3  proc copy in=source out=xptout memtype=data;
4  select bonus budget salary;
5  run;

NOTE: Copying SOURCE.BONUS to XPTOUT.BONUS (memtype=DATA).
NOTE: The data set XPTOUT.BONUS has 1 observations and 3 variables.
NOTE: Copying SOURCE.BUDGET to XPTOUT.BUDGET (memtype=DATA).
NOTE: The data set XPTOUT.BUDGET has 1 observations and 3 variables.
NOTE: Copying SOURCE.SALARY to XPTOUT.SALARY (memtype=DATA).
NOTE: The data set XPTOUT.SALARY has 1 observations

```

Example Code 13.2 Insource Library Log

```

1  LIBNAME insource xport 'filename-on-receiving-host';
NOTE: Libref INSOURCE was successfully assigned as follows:
      Engine:          XPORT
      Physical Name:   filename-on-receiving-host
2  proc copy in=insource out=work;
3  run;
NOTE: Input library INSOURCE is sequential.
NOTE: Copying INSOURCE.BUDGET to WORK.BUDGET (memtype=DATA).
NOTE: BUFSIZE is not cloned when copying across different engines.
      System Option for BUFSIZE was used.
NOTE: The data set WORK.BUDGET has 1 observations and 3 variables.
NOTE: Copying INSOURCE.BONUS to WORK.BONUS (memtype=DATA).
NOTE: BUFSIZE is not cloned when copying across different engines.
      System Option for BUFSIZE was used.
NOTE: The data set WORK.BONUS has 1 observations and 3 variables.
NOTE: Copying INSOURCE.SALARY to WORK.SALARY (memtype=DATA).
NOTE: BUFSIZE is not cloned when copying across different engines.
      System Option for BUFSIZE was used.
NOTE: The data set WORK.SALARY has 1 observations and 3 variables.

```

Example 2: Converting SAS Data Sets Encodings

Features:

PROC COPY statement options

IN=

NOCLONE

OUT=

SELECT statement

CVP engine

Details

This example demonstrates how to convert encoding from one type to another type. In order for this example to work correctly, the two encodings must be compatible. For documentation, see [“Compatible and Incompatible Encodings” in SAS National Language Support \(NLS\): Reference Guide](#).

Program

Assign library references. The two encodings must be compatible.

```
LIBNAME inlib cvp 'SAS-library';  
LIBNAME outlib 'SAS-library' outencoding="encoding value for output";  
proc copy noclone in=inlib out=outlib;  
    select car;  
run;
```


Log Examples

Example Code 13.3 InLib Library Log

```

1  LIBNAME inlib cvp 'SAS-library';
NOTE: Libref INLIB was successfully assigned as follows:
      Engine:          V9
      Physical Name: SAS-library
2  LIBNAME outlib 'SAS-library' outencoding="encoding value for output";
NOTE: Libref OUTLIB was successfully assigned as follows:
      Engine:          V9
      Physical Name: SAS-library
3  proc copy noclone in=inlib out=outlib;
4  select car;
5  run;

NOTE: Copying INLIB.CAR to OUTLIB.CAR (memtype=DATA).
NOTE: System Options for BUFSIZE and REUSE were used at user's request.

NOTE: Libname and/or system options for compress, pointobs, data representation and
encoding attributes were used at user's request.
NOTE: Data file OUTLIB.CAR.DATA is in a format that is native to another host, or
the file encoding does not match the session encoding. Cross Environment Data Access
will be used, which might require additional CPU resources and might reduce
performance.

NOTE: There were 25 observations read from the data set INLIB.CAR.

NOTE: The data set OUTLIB.CAR has 25 observations and 2 variables.

```

Example 3: Copy a SAS Data Set to a CAS Table

Features: PROC COPY statement options
 IN=
 OUT=
 SELECT statement

Details

This example demonstrates how to copy a SAS data set into a CAS table.

Program

```
libname mycas cas;
```

```

libname mylib 'BASE-engine-library';

proc copy in=mylib out=mycas;
  select monthly;
run;
quit;

```

Program Description

Assign library references. Select the data set that you want to copy into a CAS table.

```

libname mycas cas;
libname mylib 'BASE-engine-library';

```

Use PROC COPY and the SELECT statement. Copy a SAS data set into a CAS table.

```

proc copy in=mylib out=mycas;
  select monthly;
run;
quit;

```

Log Examples

Example Code 13.4 MyLib Library Log

```

57      libname mycas cas;
NOTE: Libref MYCAS was successfully assigned as follows:
      Engine:          CAS
      Physical Name: 1f436ced
58      libname mylib 'BASE-engine-library';
NOTE: Libref MYLIB was successfully assigned as follows:
      Engine:          V9
      Physical Name: BASE-engine-library
59
60      proc copy in=mylib out=mycas;
61          select monthly;
62      run;

NOTE: Copying MYLIB.MONTHLY to MYCAS.MONTHLY (memtype=DATA).
NOTE: There were 12012 observations read from the data set MYLIB.MONTHLY.
NOTE: The data set MYCAS.MONTHLY has 12012 observations and 7 variables.
NOTE: PROCEDURE COPY used (Total process time):
      real time          0.06 seconds
      cpu time           0.02 seconds

```

CPORT Procedure

Overview: CPORT Procedure	341
What Does the CPORT Procedure Do?	341
Process for Creating and Reading a Transport File	342
Syntax: CPORT Procedure	342
PROC CPORT Statement	343
EXCLUDE Statement	350
SELECT Statement	352
TRANTAB Statement	353
Usage: CPORT Procedure	354
READ= Data Set Option in the PROC CPORT Statement	354
CPORT Problems: Creating Transport Files	355
Examples: CPORT Procedure	357
Example 1: Exporting Multiple Catalogs	357
Example 2: Exporting Individual Catalog Entries	359
Example 3: Exporting a Single SAS Data Set	360
Example 4: Applying a Translation Table	361
Example 5: Exporting Entries Based on Modification Date	362

Overview: CPORT Procedure

What Does the CPORT Procedure Do?

The CPORT procedure writes SAS data sets, SAS catalogs, or SAS libraries to sequential file formats (transport files). Use PROC CPORT with the CIMPORT procedure to move files from one environment to another. *Transport files* are sequential files that each contain a SAS library, a SAS catalog, or a SAS data set in transport format. The transport format that PROC CPORT writes is the same for all

environments and for many releases of SAS. In PROC CPORT, *export* means to put a SAS library, a SAS catalog, or a SAS data set into transport format. PROC CPORT exports catalogs and data sets, either singly or as a SAS library. PROC CIMPORT restores (*imports*) the transport file to its original form as a SAS catalog, SAS data set, or SAS library.

PROC CPORT also *converts* SAS files, which means that it changes the format of a SAS file from the format appropriate for one version of SAS to the format appropriate for another version. For example, you can use PROC CPORT and PROC CIMPORT to move files from earlier releases of SAS to more recent releases. PROC CIMPORT automatically converts the transport file as it imports it.

Note: PROC CPORT and PROC CIMPORT can be used to back up graphic catalogs. PROC COPY cannot be used to back up graphic catalogs.

PROC CPORT produces no output (other than the transport files), but it does write notes to the SAS log.

Process for Creating and Reading a Transport File

Here is the process to create a transport file at the source computer and to read it at a target computer:

- 1 A transport file is created at the source computer using PROC CPORT.
- 2 The transport file is transferred from the source computer.
- 3 The transport file is read at the target computer using PROC CIMPORT.

Note: Transport files that are created using PROC CPORT are not interchangeable with transport files that are created using the XPORT engine.

For complete details about the steps to create a transport file (PROC CPORT), to transfer the transport file, and to restore the transport file (PROC CIMPORT), see *Moving and Accessing SAS Files*.

Syntax: CPORT Procedure

Tip: Use PROC CPORT or PROC CIMPORT when backing up graphic catalogs. PROC COPY cannot be used to back up graphic catalogs.

See: For complete details about the steps to create a transport file (PROC CPORT), to transfer the transport file, and to restore the transport file (PROC CIMPORT), see *Moving and Accessing SAS Files*

```

PROC CPORT source-type=libref | <libref.>member-name <options>;
EXCLUDE SAS file(s) | catalog entry(s) < / MEMTYPE=mtype >
< / ENTRYTYPE=entry-type >;
SELECT SAS file(s) | catalog entry(s) < / MEMTYPE=mtype >
< / ENTRYTYPE=entry-type >;
TRANTAB NAME=translation-table-name
<options>;

```

Statement	Task	Example
PROC CPORT	Create a transport file	Ex. 1, Ex. 2, Ex. 3, Ex. 4, Ex. 5
EXCLUDE	Exclude one or more specified files from the transport file	
SELECT	Specify one or more files or entries to include in the transport file	Ex. 2
TRANTAB	Specify one or more translation tables for characters in catalog entries to be exported	Ex. 4

PROC CPORT Statement

Creates a transport file.

Examples:

- “Example 1: Exporting Multiple Catalogs” on page 357
- “Example 2: Exporting Individual Catalog Entries” on page 359
- “Example 3: Exporting a Single SAS Data Set” on page 360
- “Example 4: Applying a Translation Table” on page 361
- “Example 5: Exporting Entries Based on Modification Date” on page 362

Syntax

```

PROC CPORT source-type=libref | <libref.>member-name <options>;

```

Summary of Optional Arguments

NOEDIT

exports SAS/AF PROGRAM and SCL entries without Edit capability when you import them.

NOSRC

specifies that exported catalog entries contain compiled SCL code, but not the source code.

OUTLIB=libref

specifies a libref associated with a SAS library.

Control the contents of the transport file

ASIS

suppresses the conversion of displayed character data to transport format.

CONSTRAINT=YES | NO

controls the exportation of integrity constraints.

DATECOPY

copies the created and modified date and time to the transport file.

INDEX=YES | NO

controls the exportation of indexes with indexed SAS data sets.

INTYPE=IBM | HITAC | FACOM | DEC | SJIS | PCIBM

specifies the type of DBCS data stored in the SAS files to be exported.

NOCOMPRESS

suppresses the compression of binary zeros and blanks in the transport file.

OUTTYPE=UPCASE

writes all alphabetic characters to the transport file in uppercase.

TRANSLATE=(translation-list)

translates specified characters from one ASCII or EBCDIC value to another.

Identify the transport file

FILE=fileref | 'filename'

specifies the transport file to write to.

TAPE

directs the output from PROC CPORT to a tape.

Select files to export

AFTER=date

exports copies of all data sets or catalog entries that have a modification date equal to or later than the date that you specify.

EET=(etype(s))

excludes specified entry types from the transport file.

ET=(etype(s))

includes specified entry types in the transport file.

GENERATION=YES | NO

specifies whether to export all generations of a data set.

MEMTYPE=ALL | CATALOG | DATA

specifies that only data sets, only catalogs, or both, be moved when a library is exported.

Required Argument

source-type=libref* | *<libref.>member-name

identifies the type of file to export and specifies the catalog, SAS data set, or SAS library to export.

source-type

identifies one or more files to export as a single catalog, as a single SAS data set, or as the members of a SAS library. The *source-type* argument can be one of the following:

CATALOG=

specifies that the file type is a SAS catalog.

Aliases C=

CAT=

DATA=

specifies that the file type is a SAS data set.

Aliases D=

DS=

LIBRARY=

specifies that the file type is a SAS library.

Aliases L=

LIB=

Note If you specify a password-protected data set as the source type, you must also include the password when creating its transport file. For details, see [“READ= Data Set Option in the PROC CPORT Statement ” on page 354.](#)

libref

<libref.>member-name

specifies the specific catalog, SAS data set, or SAS library to export. If *source-type* is CATALOG or DATA, you can specify both a *libref* and a member name. If the *libref* is omitted, PROC CPORT uses the default library as the *libref*, which is usually the WORK library. If the *source-type* argument is LIBRARY, specify only a *libref*. If you specify a library, PROC CPORT exports only data sets and catalogs from that library. You cannot export other types of files.

See Refer to [“SAS Names” in SAS Programmer’s Guide: Essentials](#) for naming conventions that you can use for names and member names.

Optional Arguments

AFTER=*date*

exports copies of all data sets or catalog entries that have a modification date later than or equal to the date that you specify. The modification date is the most recent date when the contents of the data set or catalog entry changed. Specify date as a SAS date literal or as a numeric SAS date value.

Tip You can determine the modification date of a catalog entry by using the CATALOG procedure.

Example [“Example 5: Exporting Entries Based on Modification Date” on page 362](#)

ASIS

suppresses the conversion of displayed character data to transport format. Use this option when you move files that contain DBCS (double-byte character set) data from one operating environment to another if both operating environments use the same type of DBCS data.

Interactions The ASIS option invokes the NOCOMPRESS option.

You cannot use both the ASIS option and the OUTTYPE= options in the same PROC CPORT step.

CONSTRAINT=YES | NO

controls the exportation of integrity constraints that have been defined on a data set. When you specify CONSTRAINT=YES, all types of integrity constraints are exported for a library; only general integrity constraints are exported for a single data set. When you specify CONSTRAINT=NO, indexes created without integrity constraints are ported, but neither integrity constraints nor any indexes created with integrity constraints are ported. For more information about integrity constraints, see [“Integrity Constraints” in SAS V9 LIBNAME Engine: Reference](#).

Alias CON=

Default YES

Interactions You cannot specify both CONSTRAINT= and INDEX= in the same PROC CPORT step.

If you specify INDEX=NO, no integrity constraints are exported.

DATECOPY

copies the SAS internal date and time at which the SAS file was created and the date and time at which it was last modified to the resulting transport file. Note that the operating environment date and time are not preserved.

Restriction DATECOPY can be used only when the destination file uses the V8 or V9 engine.

Note If the file that you are transporting has attributes that require additional processing, then the last modified date might be changed to the current date and time.

Tips The DATECOPY option must be specified to transport data sets with time zone offsets.

You can alter the file creation date and time with the DTC= option in the MODIFY statement in a PROC DATASETS step. For details, see [“MODIFY Statement” on page 446](#).

EET=(etype(s))

excludes specified entry types from the transport file. If *etype* is a single entry type, then you can omit the parentheses. Separate multiple values with a space.

Interaction You cannot use both the EET= option and the ET= option in the same PROC CPORT step.

ET=(etype(s))

includes specified entry types in the transport file. If *etype* is a single entry type, then you can omit the parentheses. Separate multiple values with a space.

Interaction You cannot use both the EET= option and the ET= option in the same PROC CPORT step.

FILE=fileref | 'filename'

specifies a previously defined fileref or the filename of the transport file to write to. If you omit the FILE= option, then PROC CPORT writes to the fileref SASCAT, if defined. If the fileref SASCAT is not defined, PROC CPORT writes to SASCAT.DAT in the current directory.

Note The behavior of PROC CPORT when SASCAT is undefined varies from one operating environment to another. For details, see the SAS documentation for your operating environment.

Example All examples.

GENERATION=YES | NO

specifies whether to export all generations of a SAS data set. To export only the base generation of a data set, specify GENERATION=NO in the PROC CPORT statement. To export a specific generation number, use the GENNUM= data set option when you specify a data set in the PROC CPORT statement. For more information, see [“Generation Data Sets” in SAS V9 LIBNAME Engine: Reference](#).

Alias GEN=

Default YES for libraries; NO for single data sets

Note PROC CIMPORT imports all generations of a data set that are present in the transport file. It deletes any previous generation set with the same name and replaces it with the imported generation set, even if the number of generations does not match.

INDEX=YES | NO

specifies whether to export indexes with indexed SAS data sets.

Default YES

Interactions You cannot specify both INDEX= and CONSTRAINT= in the same PROC CPORT step.

If you specify INDEX=NO, no integrity constraints are exported.

INTYPE=IBM | HITAC | FACOM | DEC | SJIS | PCIBM

specifies the type of DBCS data stored in the SAS files to be exported. Double-byte character set (DBCS) data uses up to two bytes for each character in the set. *DBCS-type* must be one of the following values:

IBM

for z/OS or VSE

HITAC

for z/OS

FACOM

for z/OS

DEC

for OpenVMS

SJIS

for OpenVMS or OS/2

PCIBM

for OS/2

Default If the INTYPE= option is not used, the DBCS type defaults to the value of the SAS system option DBCSTYPE=.

Restriction The INTYPE= option is allowed only if SAS is built with Double-Byte Character Set (DBCS) extensions. Because these extensions require significant computing resources, there is a special distribution for those sites that require it. An error is reported if this option is used at a site for which DBCS extensions are not enabled.

Interactions Use the INTYPE= option in conjunction with the OUTTYPE= option to change from one type of DBCS data to another.

The INTYPE= option invokes the NOCOMPRESS option.

You cannot use the INTYPE= option and the ASIS option in the same PROC CPORT step.

Tip You can set the value of the SAS system option DBCSTYPE= in your configuration file.

MEMTYPE=ALL | CATALOG | DATA

restricts the type of SAS file that PROC CPORT writes to the transport file.

MEMTYPE= restricts processing to one member type. Values for *mtype* can be

ALL

both catalogs and data sets

CATALOG

catalogs

Alias CAT

DATA

SAS data sets

Alias DS

Alias MT=

Default ALL

Example [“Example 1: Exporting Multiple Catalogs” on page 357](#)

NOCOMPRESS

suppresses the compression of binary zeros and blanks in the transport file.

Alias NOCOMP

Default By default, PROC CPORT compresses binary zeros and blanks to conserve space.

Interaction The ASIS, INTYPE=, and OUTTYPE= options invoke the NOCOMPRESS option.

Note Compression of the transport file does not alter the flag in each catalog and data set that indicates whether the original file was compressed.

NOEDIT

exports SAS/AF PROGRAM and SCL entries without Edit capability when you import them.

The NOEDIT option produces the same results as when you create a new catalog to contain SCL code by using the MERGE statement with the NOEDIT option in the BUILD procedure of SAS/AF software.

Alias NEDIT

Note The NOEDIT option affects only SAS/AF PROGRAM and SCL entries. It does not affect FSEDIT SCREEN or FSVIEW FORMULA entries.

NOSRC

specifies that exported catalog entries contain compiled SCL code but not the source code.

The NOSRC option produces the same results as when you create a new catalog to contain SCL code by using the MERGE statement with the NOSOURCE option in the BUILD procedure of SAS/AF software.

Alias NSRC

OUTLIB=libref

specifies a libref associated with a SAS library. If you specify the OUTLIB= option, PROC CIMPORT is invoked automatically to re-create the input library, data set, or catalog in the specified library.

Alias OUT=

Tip Use the OUTLIB= option when you change SAS files from one DBCS type to another within the same operating environment if you want to keep the original data intact.

OUTTYPE=UPCASE

writes all displayed characters to the transport file and to the OUTLIB= file in uppercase.

Interaction The OUTTYPE= option invokes the NOCOMPRESS option.

TAPE

directs the output from PROC CPORT to a tape.

Default The output from PROC CPORT is sent to disk.

TRANSLATE=(translation-list)

translates specified characters from one ASCII or EBCDIC value to another. Each element of *translation-list* has the following form:

- *ASCII-value-1* TO *ASCII-value-2*
- *EBCDIC-value-1* TO *EBCDIC-value-2*

You can use hexadecimal or decimal representation for ASCII values. If you use the hexadecimal representation, values must begin with a digit and end with an x. Use a leading zero if the hexadecimal value begins with an alphabetic character.

For example, to translate all left brackets to left braces, specify the TRANSLATE= option as follows (for ASCII characters):

```
translate=(5bx to 7bx)
```

The following example translates all left brackets to left braces and all right brackets to right braces:

```
translate=(5bx to 7bx 5dx to 7dx)
```

EXCLUDE Statement

Excludes specified files or entries from the transport file.

Interaction: You can use either EXCLUDE statements or SELECT statements in a PROC CPORT step, but not both.

Tip: There is no limit to the number of EXCLUDE statements that you can use in one invocation of PROC CPORT.

Syntax

EXCLUDE *SAS file(s) | catalog entry(s)* < / MEMTYPE=*mtype* >
< / ENTRYTYPE=*entry-type* >;

Required Arguments

SAS file(s) | catalog entry(s)

specifies one or more SAS files to be excluded from the transport file. Specify SAS filenames when you export a SAS library; specify catalog entry names when you export an individual SAS catalog. Separate multiple filenames or entry names with a space. You can use shortcuts to list many like-named files in the EXCLUDE statement. For more information, see [“SAS Data Sets” in SAS Programmer’s Guide: Essentials](#).

specifies one or more catalog entries to be excluded from the transport file. Specify catalog entry names when you export an individual SAS catalog. Separate multiple entry names with a space. You can use shortcuts to list many like-named files in the EXCLUDE statement. For more information, see [“SAS Data Sets” in SAS Programmer’s Guide: Essentials](#).

Optional Arguments

ENTRYTYPE=entry-type

specifies a single entry type for the catalog entries listed in the EXCLUDE statement. See [“Catalog Names and Catalog Entries” in SAS V9 LIBNAME Engine: Reference](#).

Aliases ETYPE=
 ET=

Restriction ENTRYTYPE= is valid only when you export an individual SAS catalog.

MEMTYPE=mtype

specifies a single member type for one or more SAS files listed in the EXCLUDE statement. Valid values are CATALOG, DATA, or ALL. If you do not specify the MEMTYPE= option in the EXCLUDE statement, then processing is restricted to those member types specified in the MEMTYPE= option in the PROC CPORT statement.

You can also specify the MEMTYPE= option, enclosed in parentheses, immediately after the name of a file. In parentheses, MEMTYPE= identifies the type of the filename that immediately precedes it. When you use this form of the option, it overrides the MEMTYPE= option that follows the slash in the EXCLUDE statement, but it must match the MEMTYPE= option in the PROC CPORT statement:

Aliases MTYPE=
 MT=

Default	If you do not specify MEMTYPE= in the PROC CPORT statement or in the EXCLUDE statement, the default is MEMTYPE=ALL.
Restrictions	MEMTYPE= is valid only when you export a SAS library. If you specify a member type for MEMTYPE= in the PROC CPORT statement, it must agree with the member type that you specify for MEMTYPE= in the EXCLUDE statement.
See	Refer to “SAS Names” in SAS Programmer’s Guide: Essentials for naming conventions that you can use for names and member names.

SELECT Statement

Includes specified files or entries in the transport file.

Interaction:	You can use either EXCLUDE statements or SELECT statements in a PROC CPORT step, but not both.
Tip:	There is no limit to the number of SELECT statements that you can use in one invocation of PROC CPORT.
Examples:	“Example 2: Exporting Individual Catalog Entries” on page 359 “Example 4: Using PROC CIMPORT to Import a French Data Set into a UTF-8 SAS Session” on page 205

Syntax

```
SELECT SAS file(s) | catalog entry(s) < / MEMTYPE=mtype >
< / ENTRYTYPE=entry-type > ;
```

Required Argument

SAS file(s) | catalog entry(s)

specifies one or more SAS files or one or more catalog entries to be included in the transport file. Specify SAS filenames when you export a SAS library; specify catalog entry names when you export an individual SAS catalog. Separate multiple filenames or entry names with a space. You can use shortcuts to list many like-named files in the SELECT statement. For more information, see [“SAS Data Sets” in SAS Programmer’s Guide: Essentials](#).

Optional Arguments

ENTRYTYPE=entry-type

specifies a single entry type for the catalog entries listed in the SELECT statement. See [“Catalog Names and Catalog Entries” in SAS V9 LIBNAME Engine: Reference](#), for a complete list of catalog entry types.

Aliases ETYPE=

ET=

Restriction ENTRYTYPE= is valid only when you export an individual SAS catalog.

MEMTYPE=*mtype*

specifies a single member type for one or more SAS files listed in the SELECT statement. Valid values are CATALOG, DATA, or ALL. If you do not specify the MEMTYPE= option in the SELECT statement, then processing is restricted to those member types specified in the MEMTYPE= option in the PROC CPORT statement.

You can also specify the MEMTYPE= option, enclosed in parentheses, immediately after the name of a member. In parentheses, MEMTYPE= identifies the type of the member name that immediately precedes it. When you use this form of the option, it overrides the MEMTYPE= option that follows the slash in the SELECT statement, but it must match the MEMTYPE= option in the PROC CPORT statement.

Aliases MTYPE=

MT=

Default If you do not specify MEMTYPE= in the PROC CPORT statement or in the SELECT statement, the default is MEMTYPE=ALL.

Restrictions MEMTYPE= is valid only when you export a SAS library.

If you specify a member type for MEMTYPE= in the PROC CPORT statement, it must agree with the member type that you specify for MEMTYPE= in the SELECT statement.

See Refer to “SAS Names” in *SAS Programmer’s Guide: Essentials* for naming conventions that you can use for names and member names.

TRANTAB Statement

Specifies translation tables for characters in catalog entries that you export.

Restriction: The TRANTAB statement does not support DBCS or UTF-8 SAS sessions.

Tip: You can specify only one translation table for each TRANTAB statement. However, you can use more than one translation table in a single invocation of PROC CPORT.

See: The TRANTAB Statement for the CPORT Procedure and the UPLOAD and DOWNLOAD Procedures in *SAS National Language Support (NLS): Reference Guide*.

Example: [“Example 4: Applying a Translation Table” on page 361](#)

Syntax

TRANTAB **NAME**=*translation-table-name* <options>;

Required Argument

NAME=

specifies the name of the translation table to use.

Usage: CPORT Procedure

READ= Data Set Option in the PROC CPORT Statement

To be authorized to create the transport file for a read-protected data set, you must include the password (clear-text or encoded). If the password is not included, the transport file cannot be created.

If you are working with a password protected data set, you can supply that password using the READ= option. If you do not supply the password using the READ= option for a read-protected data set, you are prompted for the password.

Use the READ= data set option to include the appropriate password for the read-protected data set when creating a transport file. In Example 1, PROC CPORT copies the input file that is named SOURCE.GRADES, includes the password ADMIN with the data set, and creates the transport file named GRADESOUT.

Example 1: Clear-Text Password:

```
proc cport data=source.grades(read=admin) file=gradesout;
```

In Example 2, an encoded password is specified with the READ= option. An encoded password is generated via the PWENCODE procedure. For details, see [Chapter 45, “PWENCODE Procedure,” on page 1573](#).

Example 2: Encoded Password

```
proc cport  
data=source.grades(read={SAS005}ADD8AB7108595A7D1A69190D78CDFE6145C1EB8  
49CC7A43D)  
file=gradesout;
```

If the password is omitted when referring to a password protected data set, SAS prompts for the password. If an invalid password is specified, an error message is sent to the log. Here is an example error:

```
ERROR: Invalid or missing READ password on member WORK.XYZ.DATA
```


If the data set is transported as part of a library or named in a SELECT statement, a password is not required. However, if a data set with a password is transported and the target SAS engine does not support passwords, the transport file cannot be imported.

For details about the READ= data set option, see [SAS Data Set Options: Reference](#), and for details about password-protected data sets, see [SAS Programmer's Guide: Essentials](#).

Note: PROC CIMPORT does not require a password in order to restore the transport file in the target environment. However, other SAS procedures that use the password-protected data set must include the password.

CPORT Problems: Creating Transport Files

About Transport Files and Encodings

In the SAS Viya platform, UTF-8 is the default encoding. However, the SAS Viya platform supports other legacy encodings.

The character data in a transport file is created for the following types of encodings:

- the UTF-8 encoding of the SAS session in which the transport file is created.
- When you use PROC CPORT to create a transport file that is encoded with US-ASCII on an ASCII platform, regardless of the session encoding, the US-ASCII encoding is preserved for that transport file. If you then transport that data set to an ASCII platform using PROC CIMPORT, the US-ASCII encoding for that transport file is preserved and is not transcoded. The data set that is created has the US-ASCII encoding, not the session encoding. For example, if your session encoding is WLATIN1, you use PROC CPORT to create a data set that has an encoding of US-ASCII. The US-ASCII encoding is preserved in the transport file, instead of the WLATIN1 encoding. This preservation also occurs when you use PROC CIMPORT on this data set. The US-ASCII encoding is preserved and is not transcoded when you use PROC CIMPORT to transport the data set to an ASCII platform.

Note: The preservation of the US-ASCII encoding occurs only on an ASCII platform.

For information about warnings generated by PROC CIMPORT, see [“Warning Messages Generated When Using PROC CIMPORT in the SAS Viya Platform” on page 195](#).

The following example shows how SAS applies an encoding to a transport file. For information about how to set encodings in SAS Data and AI Studio, see [“Opening a Program” in SAS Data and AI Studio: User’s Guide](#).

Table 14.1 Assignment of Encodings to Transport Files

Encoding Value of the Transport File	SAS Invocation	Explanation
utf-8 or us-ascii	encoding is utf8;	A SAS session is invoked using the UTF-8 encoding. The session encoding is applied to the transport file. If the data set is US-ASCII, the US-ASCII encoding is preserved, not the session encoding.
wlatin2	locale is pl_PL;	A SAS session is invoked encoding, LATIN2, which is associated with the Polish Poland locale.

For a complete list of encodings that are associated with each locale, see [Locale Tables](#) in *SAS National Language Support (NLS): Reference Guide*.

In order for a transport file to be imported successfully, the encodings of the source and target SAS sessions must be compatible. Here is an example of compatible source and target SAS sessions:

Table 14.2 Compatible Encodings

Source SAS Session		Transport File Encoding	Target SAS Session	
Locale	Linux SAS Session Encoding		Locale	SAS Session Encoding
es_MX (Spanish Mexico)	latin1	wlatin1	it_IT (Italian Italy)	wlatin1

The encodings of the source and target SAS sessions are compatible because the default encoding for the es_MX locale is WLATIN1 and the encoding of the target SAS session is WLATIN1.

However, if the encodings of the source and target SAS sessions are incompatible, a transport file might not be successfully imported. (See the introduction to this section.) Here is an example of incompatible encodings:

Table 14.3 Incompatible Encodings

Source SAS Session			Target SAS Session	
Locale	Linux SAS Session Encoding	Transport File Encoding	Locale	Encoding
cs_CZ (Czech Czechoslovakia)	latin2	wlatin2	de_DE (German Germany)	latin1

The encodings of the source and target SAS sessions are incompatible because the default encoding for the cs_CZ locale is WLATIN2 and the encoding of the target SAS session is latin1. A transport file cannot be imported between these locales.

The target session encoding is latin1. The CPORT transport file is WLATIN2 and can be imported in this session. However, some characters might be improperly changed. The following warning message is provided:

```
WARNING: The transport file was created in the encoding wlatin2. This session
encoding, wlatin1, can read and create transport files in the encoding wlatin1.
Data might not be imported correctly.
```

Examples: CPORT Procedure

Example 1: Exporting Multiple Catalogs

Features:

PROC CPORT statement options
FILE=
MEMTYPE=

Details

This example shows how to use PROC CPORT to export entries from all of the SAS catalogs in the SAS library that you specify.

Program

```
libname source 'sas-library';
filename tranfile 'transport-file';

host-options-for-file-characteristics;

proc cport library=source file=tranfile memtype=catalog;
run;
```

Program Description

Specify the library reference for the SAS library that contains the source files to be exported and the file reference to which the output transport file is written.

The LIBNAME statement assigns a libref for the SAS library. The FILENAME statement assigns a fileref and any operating environment options for file characteristics for the transport file that PROC CPORT creates.

```
libname source 'sas-library';
filename tranfile 'transport-file';

host-options-for-file-characteristics;
```

Create the transport file. The PROC CPORT step executes on the operating environment where the source library is located. MEMTYPE=CATALOG writes all SAS catalogs in the source library to the transport file.

```
proc cport library=source file=tranfile memtype=catalog;
run;
```

Log Examples

Example Code 14.1 Exporting Multiple Catalogs

```
NOTE: Proc CPORT begins to transport catalog SOURCE.FINANCE
NOTE: The catalog has 5 entries and its maximum logical record length is 866.
NOTE: Entry LOAN.FRAME has been transported.
NOTE: Entry LOAN.HELP has been transported.
NOTE: Entry LOAN.KEYS has been transported.
NOTE: Entry LOAN.PMENU has been transported.
NOTE: Entry LOAN.SCL has been transported.

NOTE: Proc CPORT begins to transport catalog SOURCE.FORMATS
NOTE: The catalog has 2 entries and its maximum logical record length is 104.
NOTE: Entry REVENUE.FORMAT has been transported.
NOTE: Entry DEPT.FORMATC has been transported.
```

Example 2: Exporting Individual Catalog Entries

Features: PROC CPORT statement option
FILE=
SELECT statement

Details

This example shows how to use PROC CPORT to export individual catalog entries, rather than all of the entries in a catalog.

Program

```
libname source 'sas-library';  
filename tranfile 'transport-file';  
  
host-options-for-file-characteristics;  
  
proc cport catalog=source.finance file=tranfile;  
select loan.scl;  
run;
```

Program Description

Assign library references. The LIBNAME and FILENAME statements assign a libref for the source library and a fileref for the transport file, respectively.

```
libname source 'sas-library';  
filename tranfile 'transport-file';  
  
host-options-for-file-characteristics;
```

Write an entry to the transport file. SELECT writes only the LOAN.SCL entry to the transport file for export.

```
proc cport catalog=source.finance file=tranfile;  
select loan.scl;  
run;
```

Log Examples

Example Code 14.2 Exporting Individual Catalog Entries

```
NOTE: Proc CPORT begins to transport catalog SOURCE.FINANCE
NOTE: The catalog has 5 entries and its maximum logical record length is 866.
NOTE: Entry LOAN.SCL has been transported.
```

Example 3: Exporting a Single SAS Data Set

Features: PROC CPORT statement option
FILE=

Details

This example shows how to use PROC CPORT to export a single SAS data set.

Program

```
libname source 'sas-library';
filename tranfile 'transport-file';

host-options-for-file-characteristics;

proc cport data=source.times file=tranfile;
run;
```

Program Description

Assign library references. The LIBNAME and FILENAME statements assign a libref for the source library and a fileref for the transport file, respectively.

```
libname source 'sas-library';
filename tranfile 'transport-file';
```

```
host-options-for-file-characteristics;
```

Specify the type of file that you are exporting. The DATA= specification in the PROC CPORT statement tells the procedure that you are exporting a SAS data set rather than a library or a catalog.

```
proc cport data=source.times file=tranfile;
```

```
run;
```

Log Examples

Example Code 14.3 Exporting a Single SAS Data Set

```
NOTE: Proc CPORT begins to transport data set SOURCE.TIMES
NOTE: The data set contains 2 variables and 2 observations.
      Logical record length is 16.
NOTE: Transporting data set index information.
```

Example 4: Applying a Translation Table

Features:

- PROC CPORT statement option
FILE=
- TRANTAB statement option
TYPE=

Details

This example shows how to apply a customized translation table to the transport file before PROC CPORT exports it. For this example, assume that you have already created a customized translation table called TTABLE1.

Program

```
libname source 'sas-library';
filename tranfile 'transport-file';

host-options-for-file-characteristics;

proc cport catalog=source.formats file=tranfile;
  trantab name=ttable1 type=(format);
run;
```

Program Description

Assign library references. The LIBNAME and FILENAME statements assign a libref for the source library and a fileref for the transport file, respectively.

```
libname source 'sas-library';
filename tranfile 'transport-file';

host-options-for-file-characteristics;
```

Apply the translation specifics. The TRANTAB statement applies the translation that you specify with the customized translation table TTABLE1. TYPE= limits the translation to FORMAT entries.

```
proc cport catalog=source.formats file=tranfile;
    trantab name=ttable1 type=(format);
run;
```

Log Examples

Example Code 14.4 Applying a Translation Table

```
NOTE: Proc CPORT begins to transport catalog SOURCE.FORMATS
NOTE: The catalog has 2 entries and its maximum logical record length is 104.
NOTE: Entry REVENUE.FORMAT has been transported.
NOTE: Entry DEPT.FORMATC has been transported.
```

Example 5: Exporting Entries Based on Modification Date

Features:	PROC CPORT statement options
	AFTER=
	FILE=

Details

This example shows how to use PROC CPORT to transport only the catalog entries with modification dates equal to or later than the date that you specify in the AFTER= option.

Program

```
libname source 'sas-library';
filename tranfile 'transport-file';

host-options-for-file-characteristics;
```



```
proc cport catalog=source.finance file=tranfile
          after='09sep1996'd;
run;
```

Program Description

Assign library references. The LIBNAME and FILENAME statements assign a libref for the source library and a fileref for the transport file, respectively.

```
libname source 'sas-library';
filename tranfile 'transport-file';

host-options-for-file-characteristics;
```

Specify the catalog entries to be written to the transport file. AFTER= specifies that only catalog entries with modification dates on or after September 9, 1996, should be written to the transport file.

```
proc cport catalog=source.finance file=tranfile
          after='09sep1996'd;
run;
```

Log Examples

PROC CPORT writes messages to the SAS log to inform you that it began the export process for all the entries in the specified catalog. However, PROC CPORT wrote only the entries LOAN.FRAME and LOAN.HELP in the FINANCE catalog to the transport file because only those two entries had a modification date equal to or later than September 9, 1996. That is, of all the entries in the specified catalog, only two met the requirement of the AFTER= option.

Example Code 14.5 Exporting Entries Based on Modification Date

```
NOTE: Proc CPORT begins to transport catalog SOURCE.FINANCE
NOTE: The catalog has 5 entries and its maximum logical record length is 866.
NOTE: Entry LOAN.FRAME has been transported.
NOTE: Entry LOAN.HELP has been transported.
```


DATASETS Procedure

Overview: DATASETS Procedure	365
Managing Data Sets Using the DATASETS Procedure	366
Notes	367
Concepts: DATASETS Procedure	368
Procedure Execution	368
Extended Attributes	371
Contents of a Library at the Directory Level	372
CONTENTS Statement with VARCHAR	373
Differences in the DATASETS Procedure Output When Using SAS/ACCESS LIBNAME Engines	375
CAS Processing for the DATASETS Procedure	376
PROC DATASETS and the Output Delivery System (ODS)	376
Syntax: DATASETS Procedure	378
PROC DATASETS Statement	382
AGE Statement	387
APPEND Statement	389
ATTRIB Statement	400
AUDIT Statement	401
CHANGE Statement	403
CONTENTS Statement	405
COPY Statement	413
DELETE Statement	427
EXCHANGE Statement	432
EXCLUDE Statement	433
FORMAT Statement	434
IC CREATE Statement	435
IC DELETE Statement	438
IC REACTIVATE Statement	438
INDEX CENTILES Statement	439
INDEX CREATE Statement	440
INDEX DELETE Statement	441
INFORMAT Statement	442
INITIATE Statement	443
LABEL Statement	444
LOG Statement	444
MODIFY Statement	446
QUIT Statement	450
REBUILD Statement	451

RENAME Statement	453
REPAIR Statement	454
RESUME Statement	456
SAVE Statement	457
SELECT Statement	458
SUSPEND Statement	460
TERMINATE Statement	460
USER_VAR Statement	461
XATTR ADD Statement	461
XATTR DELETE Statement	462
XATTR OPTIONS Statement	463
XATTR REMOVE Statement	464
XATTR SET Statement	464
XATTR UPDATE Statement	466
Usage: DATASETS Procedure	467
Using Passwords with the DATASETS Procedure	467
Restricting Processing for Generation Data Sets	468
Restricting Member Types for Processing	469
Sample PROC DATASETS Output	472
Results: DATASETS Procedure	474
Directory Listing to the SAS Log	474
Directory Listing as SAS Output	474
Procedure Output	475
PROC DATASETS and the Output Delivery System (ODS)	482
ODS Table Names	483
Output Data Sets	484
Examples: DATASETS Procedure	492
Example 1: Removing All Labels and Formats in a Data Set	492
Example 2: Manipulating SAS Files	497
Example 3: Saving SAS Files from Deletion	502
Example 4: Modifying SAS Data Sets	505
Example 5: Describing a SAS Data Set	508
Example 6: Concatenating Two SAS Data Sets	511
Example 7: Aging SAS Data Sets	514
Example 8: Initiating an Audit File	516
Example 9: Extended Attributes	522

Overview: DATASETS Procedure

Managing Data Sets Using the DATASETS Procedure

The DATASETS procedure is a utility procedure that manages your SAS files. With PROC DATASETS, you can do the following:

- copy SAS files from one SAS library to another
- rename SAS files
- repair SAS files
- delete SAS files
- list the SAS files that are contained in a SAS library
- list the attributes of a SAS data set:
 - the date on which the data was last modified
 - whether the data is compressed
 - whether the data is indexed
- manipulate passwords on SAS files
- append SAS data sets
- modify attributes of SAS data sets and variables within the data sets
- create and delete indexes on SAS data sets
- create and manage audit files for SAS data sets
- create and delete integrity constraints on SAS data sets
- create and manage extended attributes of data sets

Notes

- Although the DATASETS procedure can perform some operations on catalogs, generally the CATALOG procedure is the best utility to use for managing catalogs.
- The term *member* often appears as a synonym for *SAS file*.
- PROC DATASETS cannot work with sequential data libraries.
- You cannot change the length of a variable using the LENGTH statement or the LENGTH= option in an ATTRIB statement.
- There can be a discrepancy between the modified date in PROC DATASETS, PROC CONTENTS, and other components of SAS, such as SAS Explorer. The two modified dates and times are distinctly different:
 - Operating-environment modified date and time is reported by the SAS Explorer and the PROC DATASETS LIST option.
 - The modified date and time reported by the CONTENTS statement is the date and time that the data within the data set was actually modified.
- If you have a library containing a large number of members, the DATASETS procedure might show an increase in process time. You might want to reorganize your library into smaller libraries for better performance.
- Extended attributes are supported in the Compute Server, but not in CAS.

- If you want to use the KILL functionality of the DATASETS procedure and the DBMS=Redshift, then you need to use a SCHEMA= option in the LIBNAME statement.

Concepts: DATASETS Procedure

Procedure Execution

Execution of Statements

When you start the DATASETS procedure, you specify the procedure input library in the PROC DATASETS statement. If you omit a procedure input library, the procedure processes the current default SAS library (usually the Work library). To specify a new procedure input library, issue the DATASETS procedure again.

Statements execute in the order in which they are written. Use CONTENTS, COPY, CONTENTS if you want to see the contents of a data set, copy a data set, and then visually compare the contents of the second data set with the first.

RUN-Group Processing

PROC DATASETS supports RUN-group processing. RUN-group processing enables you to submit RUN groups without ending the procedure.

The DATASETS procedure supports four types of RUN groups. Each RUN group is defined by the statements that compose it and by what causes it to execute.

Some statements in PROC DATASETS act as implied RUN statements because they cause the RUN group preceding them to execute.

The following list discusses what statements compose a RUN group and what causes each RUN group to execute:

- The PROC DATASETS statement always executes immediately. No other statement is necessary to cause the PROC DATASETS statement to execute. Therefore, the PROC DATASETS statement alone is a RUN group.
- The MODIFY statement, and any of its subordinate statements, form a RUN group. These RUN groups always execute immediately. No other statement is necessary to cause a MODIFY RUN group to execute.

The MODIFY statement runs on the Compute Server, but not in CAS.

- The APPEND, CONTENTS, and COPY statements (including EXCLUDE and SELECT, if present), form their own separate RUN groups. Every APPEND statement forms a single-statement RUN group; every CONTENTS statement forms a single-statement RUN group; and every COPY step forms a RUN group. Any other statement in the procedure, except those that are subordinate to either the COPY or MODIFY statement, causes the RUN group to execute.
- One or more of the following statements form a RUN group:
 - ☐ AGE
 - ☐ CHANGE
 - ☐ DELETE
 - ☐ EXCHANGE
 - ☐ REPAIR
 - ☐ SAVE

If any of these statements appear in sequence in the PROC step, the sequence forms a RUN group. For example, if a REPAIR statement appears immediately after a SAVE statement, the REPAIR statement does not force the SAVE statement to execute; it becomes part of the same RUN group. To execute the RUN group, submit one of the following statements:

- ☐ PROC DATASETS
- ☐ APPEND
- ☐ CONTENTS
- ☐ COPY
- ☐ MODIFY
- ☐ QUIT
- ☐ RUN
- ☐ another DATA or PROC step

SAS reads the program statements that are associated with one task until it reaches a RUN statement or an implied RUN statement. SAS executes all of the preceding statements immediately and continues reading until it reaches another RUN statement or implied RUN statement. To execute the last task, you must use a RUN statement or a statement that stops the procedure.

The following PROC DATASETS step contains five RUN groups:

```
LIBNAME dest 'SAS-library';
  /* RUN group */

proc datasets;
  /* RUN group */
  change nutr=fatg;
  delete bldtest;
  exchange xray=chest;
  /* RUN group */
  copy out=dest;
  select report;
```

```

        /* RUN group */
    modify bp;
        label dias='Taken at Noon';
        rename weight=bodyfat;
        /* RUN group */
    append base=tissue data=newtiss;
quit;

```

Note: If you are running in interactive line mode, you can receive messages that statements have already executed before you submit a RUN statement. Plan your tasks carefully if you are using this environment for running PROC DATASETS.

Error Handling

Generally, if an error occurs in a statement, the RUN group containing the error does not execute. RUN groups preceding or following the one containing the error execute normally. The MODIFY RUN group is an exception. If a syntax error occurs in a statement subordinate to the MODIFY statement, only the statement containing the error fails. The other statements in the RUN group execute.

Note: If the first word of the statement (the statement name) is in error and the procedure cannot recognize it, the procedure treats the statement as part of the preceding RUN group.

Password Errors

If there is an error involving an incorrect or omitted password in a statement, the error affects only the statement containing the error. The other statements in the RUN group execute.

Forcing a RUN Group with Errors to Execute

The FORCE option in the PROC DATASETS statement forces execution of the RUN group even if one or more of the statements contain errors. Only the statements that are error-free execute.

Ending the Procedure

To stop the DATASETS procedure, you must issue a QUIT statement, a RUN CANCEL statement, a new PROC statement, or a DATA statement. Submitting a

QUIT statement executes any statements that have not executed. Submitting a RUN CANCEL statement cancels any statements that have not executed.

Extended Attributes

You can use extended attributes on the Compute Server, but not in CAS.

Extended attributes are customized metadata for your SAS files. They are user-defined characteristics that you associate with a SAS data set or variable. Whereas common SAS attributes, such as length for variables for data sets are predefined SAS system attributes, extended attributes are attributes that you define yourself. Extended attributes are organized into (name, value) pairs.

Use the MODIFY statement to add, delete, remove, set, and update extended attributes. When using the COPY statement, if the OUT= library engine supports extended attributes, they are copied. Extended attributes are not appended when using the APPEND statement, unless the BASE= data set does not exist.

Extended attributes can be used to automate tasks that require a custom attribute to be associated with a variable or a data set.

An extended attribute can have numeric or character values. There is no maximum length to an extended attribute character value. By default, each value is stored in 256-byte segments. The length of the segment can be changed with the SEGLEN= option of the XATTR OPTIONS statement. This option indicates the length of the storage element that will hold the character attribute value. Should the segment size not be large enough for some particular character attribute value, another segment is allocated. To minimize processing time, choose a length that would accommodate most attribute values for the data set.

The following output shows a data set and variables with extended attributes.

Output 15.1 Contents of a Data Set with Extended Attributes

Alphabetic List of Variables and Attributes			
#	Variable	Type	Len
2	age	Num	8
5	cars	Num	8
3	income	Num	8
4	kids	Num	8
1	purchase	Char	3

Alphabetic List of Data Set Extended Attributes		
Extended Attribute	Numeric Value	Character Value
attrib	-	table
role	-	train

Alphabetic List of Extended Attributes on Variables			
Extended Attribute	Attribute Variable	Numeric Value	Character Value
level	income	-	interval
level	purchase	-	nominal
role	age	-	reject
role	income	-	input
role	purchase	-	target

For information about using extended attributes, see “[XATTR ADD Statement](#)” on page 461, “[XATTR DELETE Statement](#)” on page 462, “[XATTR REMOVE Statement](#)” on page 464, “[XATTR SET Statement](#)” on page 464, and “[XATTR UPDATE Statement](#)” on page 466.

Contents of a Library at the Directory Level

In a concatenated library, each physical piece of the library is a level. Level 1 is the first location in the libref definition, Level 2 is the second location, and so on. PROC DATASETS reports the number of levels in the library as a value under the LEVEL specification.

CONTENTS Statement with VARCHAR

If there is a VARCHAR data type in the table, PROC CONTENTS or PROC DATASETS CONTENTS statement shows the Length in bytes and characters, as well as the max bytes used. PROC CONTENTS reports metadata about the table and the metadata about the variables. The CAS engine is the only engine supporting VARCHAR.

The bottom portion of PROC CONTENTS (related to the variable metadata) is the metadata that is represented in the Compute Server session. The variable byte length used in Viya is dependent on the encoding of the session. The variable byte length used in the session might differ from the byte length.

Output 15.2 Contents of Mycas.French2 Table

Data Set Name	MYCAS.FRENCH2	Observations	11
Member Type	DATA	Variables	2
Engine	CAS	Indexes	0
Created	08/16/2017 18:05:10	Observation Length	40
Last Modified	08/16/2017 18:05:10	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64		
Encoding	utf-8 Unicode (UTF-8)		

Engine/Host Dependent Information	
Data Limit	100MB
Caslib	CASUSER.
Scope	Session

Alphabetic List of Variables and Attributes					
#	Variable	Type	Len		Max Bytes Used
			Bytes	Chars	
1	nterm	Varchar	80	20	19
2	term	Char	19		

Use the following code to show the contents of the Mycas library:

```
proc datasets lib=mycas;
  contents data=cars;
run;
```

Output 15.3 Mycas Library and Mycas.Cars Table

Data Set Name	MYCAS.CARS	Observations	428
Member Type	DATA	Variables	15
Engine	CAS	Indexes	0
Created	08/16/2017 17:55:43	Observation Length	160
Last Modified	08/16/2017 17:55:43	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64		
Encoding	utf-8 Unicode (UTF-8)		

Engine/Host Dependent Information	
Data Limit	100MB
Caslib	CASUSER
Scope	Session

Alphabetic List of Variables and Attributes					
#	Variable	Type	Len	Format	Label
9	Cylinders	Num	8		
5	DriveTrain	Char	5		
8	EngineSize	Num	8		Engine Size (L)
10	Horsepower	Num	8		
7	Invoice	Num	8	DOLLAR8.	
15	Length	Num	8		Length (IN)
11	MPG_City	Num	8		MPG (City)
12	MPG_Highway	Num	8		MPG (Highway)
6	MSRP	Num	8	DOLLAR8.	
1	Make	Char	13		
2	Model	Char	40		
4	Origin	Char	6		
3	Type	Char	8		
13	Weight	Num	8		Weight (LBS)
14	Wheelbase	Num	8		Wheelbase (IN)

Use the following code to show the contents of the Mycas library:

```
proc datasets lib=mycas;
  contents data=mycs.cars directory details;
run;
```

Using the DIRECTORY and DETAILS Option

Directory	
Libref	MYCAS
Engine	CAS
Physical Name	eb4545e1
Session UUID	eb4545e1
Session Name	SASCAS1
Server Host	rdoesx
Server Port	55
Caslib	CASUSER

#	Name	Member Type	Obs, Entries or Indexes	Vars	Label	Number of Rows	Number of Columns	Last Modified	Data Encoding	Scope
1	CARS	DATA	428	15		428	15	08/16/2017 19:29:50	utf-8	Session
2	LASTMONTH	DATA	12	7		12	7	08/16/2017 18:58:09	utf-8	Session

Differences in the DATASETS Procedure Output When Using SAS/ACCESS LIBNAME Engines

When you use a SAS/ACCESS LIBNAME engine to access a database, some of the information that is available in the header of a SAS data set is not available to the procedure. Therefore, the procedure does not have the information to display. The CONTENTS procedure, the DATASETS COPY statement, and the DATASETS CONTENTS statement do not provide information about the indexes, integrity constraints, and number of observations. For example, copying a table with indexes and integrity constraints results in a table with no indexes or integrity constraints if the engine used to access the table is a SAS/ACCESS LIBNAME engine. In the following code, the output from the CONTENTS statement displays a 0 (zero) in the index field although the database has indexes, and displays a . (period) for the unknown number of observations.

```
proc datasets lib=mylib details;
    contents data=table-name;
run;
quit;
```

To obtain the number of observations, run an SQL query to obtain the count using the LIBNAME engine or explicit-pass through on the DBMS table.

To obtain the information about indexes and integrity constraints, you can use explicit pass-through of database-specific syntax to query the system tables. For documentation on explicit pass-through, see [“Connecting to a DBMS By Using the SQL Procedure Pass-Through Facility” in SAS SQL Procedure User's Guide](#).

CAS Processing for the DATASETS Procedure

Accessing CAS tables through the CAS LIBNAME engine with the DATASETS procedure results in some behavior that differs from other LIBNAME engines. The following is a summary of these behavior differences:

- Indexes, integrity constraints, and extended attributes are not supported in CAS tables, but are supported on the Compute Server.
- The MODIFY statement results in an error because UPDATE access is not supported.
- If there is a VARCHAR data type in the table, the CONTENTS statement shows the length in bytes, length in characters, and maximum bytes used.
- The encoding shown by the CONTENTS statement is the encoding of the CAS table.
- The block I/O method cannot be used when appending a CAS table to a SAS data set.
- You cannot use the APPEND statement to add data to a CAS table because UPDATE access is not supported. Data from a CAS table can be appended to a SAS data set.
- If the CAS LIBNAME engine is used for both the input and output libraries with the COPY statement, the COPY task is executed on the CAS server. The NOACCEL option disables this behavior, causing the COPY task to be executed in the Compute Server session.

PROC DATASETS and the Output Delivery System (ODS)

Most procedures send their messages to the log and their procedure results to the output. PROC DATASETS is unique because it sends procedure results to both the log and the procedure output table. When the interface to ODS was created, it was decided that all procedure results (from both the log and the procedure output table) should be available to ODS. In order to implement this feature and maintain compatibility with earlier releases, the interface to ODS had to be slightly different from the usual interface.

By default, the PROC DATASETS statement itself produces two output objects: Members and Directory. These objects are routed to the log. The CONTENTS statement produces three output objects by default: Attributes, EngineHost, and Variables. (The use of various options adds other output objects.) These objects are routed to the procedure output table. If you open an ODS destination (such as HTML, RTF, or PRINTER), all of these objects are, by default, routed to that destination.

You can use the ODS SELECT and ODS EXCLUDE statements to control which objects go to which destination, just as you can for any other procedure.

PROC DATASETS and PROC CONTENTS assign a name to each table that they create. You can use these names to reference the table when using the Output Delivery System (ODS) to select tables and create output tables.

PROC CONTENTS generates the same ODS tables as PROC DATASETS with the CONTENTS statement.

Table 15.1 ODS Tables Produced by the DATASETS Procedure without the CONTENTS Statement

ODS Table	Description	Generates Table
Directory	General library information	Unless you specify the NOLIST option
Members	Library member information	Unless you specify the NOLIST option

Table 15.2 ODS Table Names Produced by PROC CONTENTS and PROC DATASETS with the CONTENTS Statement

ODS Table	Description	Generates Table
Attributes	Data set attributes	Unless you specify the SHORT option
Directory	General library information	If you specify DATA=<libref.>_ALL_ or the DIRECTORY option
Members	Library member information	If you specify DATA=<libref.>_ALL_ or the DIRECTORY option
Position	A detailed listing of variables by logical position in the table	If you specify the VARNUM option and you do not specify the SHORT option
PositionShort	A concise listing of variables by logical position in the table	If you specify the VARNUM option and the SHORT option
Variables	A detailed listing of variables in alphabetical order	Unless you specify the SHORT option
VariablesShort	A concise listing of variables in alphabetical order	If you specify the SHORT option

Syntax: DATASETS Procedure

- Notes:** Data set names and variable names that end with large numeric values that are larger than a long integer cannot be used in numbered-range lists. For more information, see [“Definitions for SAS Variables” in SAS Programmer’s Guide: Essentials..](#)
- NOTE:** _LAST_ is the name of the last data set that is created in your session. When the BASE= data set does not exist and PROC APPEND creates it, PROC APPEND sets _LAST_ to the name of the BASE= data set.
- Tips:** Supports RUN-group processing.
- Supports the Output Delivery System. For details, see [“Output Delivery System: Basic Concepts” in SAS Output Delivery System: User’s Guide.](#)
- See:** For more information, see [“Statements with the Same Function in Multiple Procedures” on page 23.](#)

```
PROC DATASETS <options>;
  AGE current-name related-SAS-file(s)
    </ <ALTER=alter-password> <MEMTYPE=member-type>>;
  APPEND BASE=<libref.>SAS-data-set
    <APPENDVER=V6>
    <DATA=<libref.>SAS-data-set>
    <ENCRYPTKEY=key-value>
    <FORCE>
    <GETSORT>
    <NOWARN>;
  AUDIT SAS-file <(SAS-password <ENCRYPTKEY=key-value>
    <GENNUM=integer>)>;
    INITIATE <AUDIT_ALL=NO | YES>;
    LOG <ADMIN_IMAGE=YES | NO>
      <BEFORE_IMAGE=YES | NO>
      <DATA_IMAGE=YES | NO>
      <ERROR_IMAGE=YES | NO>;
    <SUSPEND | RESUME | TERMINATE>; >
    <USER_VAR> variable-name-1 <$> <length> <LABEL='variable-label' >
      <variable-name-2 <$> <length> <LABEL='variable-label' > ...>;
  CHANGE old-name-1=new-name-1
    <old-name-2=new-name-2 ...>
    </ <ENCRYPTKEY=key-value>
    <ALTER=alter-password> <GENNUM=ALL | integer> <MEMTYPE=member-
      type>>;
  CONTENTS <options>;
  COPY <ACCEL | NOACCEL>OUT=libref-1
```



```

<CLONE | NOCLONE>
<CONSTRAINT=YES | NO>
<DATECOPY>
<ENCRYPTKEY=key-value>
<FORCE>
<IN=libref-2>
<INDEX=YES | NO>
<MEMTYPE=(member-type(s))>
<MOVE <ALTER=alter-password>>
<OVERRIDE=(ds-option-1=value-1 <ds-option-2=value-2 ...> ) >;

SELECT SAS-file(s)
  </ <ENCRYPTKEY=key-value> <ALTER=alter-password>
  <MEMTYPE=member-type>>;

```

```

DELETE SAS-file(s)
  </ <ALTER=alter-password>
  <ENCRYPTKEY=key-value>
  <GENNUM=ALL | HIST | REVERT | integer>
  <MEMTYPE=member-type>>;

```

```

EXCHANGE name-1=other-name-1 <name-2=other-name-2 ...>
  </ <ALTER=alter-password> <MEMTYPE=member-type>>;

```

```

MODIFY SAS-file <(options)>
  </ <CORRECTENCODING=encoding-value> <DTC=SAS-date-time>
  <GENNUM=integer> <MEMTYPE=member-type>>;

```

```

ATTRIB variable-list(s) attribute-list(s);

```

```

FORMAT variable-1 <format-1>
  <variable-2 <format-2> ...>;

```

```

IC CREATE <constraint-name=> constraint <MESSAGE='message-string'
  <MSGTYPE=USER>>;

```

```

IC DELETE constraint-name(s) | _ALL_;

```

```

IC REACTIVATE foreign-key-name REFERENCES libref;

```

```

INDEX CENTILES index(s)
  </ <REFRESH> <UPDATECENTILES=ALWAYS | NEVER | integer>>;

```

```

INDEX CREATE index-specification(s)
  </ <NOMISS> <UNIQUE> <UPDATECENTILES=ALWAYS | NEVER |
  integer>>;

```

```

INDEX DELETE index(s) | _ALL_;

```

```

INFORMAT variable-1 <informat-1>
  <variable-2 <informat-2> ...>;

```

```

LABEL variable-1=<'label-1' | '>
  <variable-2=<'label-2' | '> ...>;

```

```

RENAME old-name-1=new-name-1
  <old-name-2=new-name-2 ...>;

```

```

XATTR ADD DS attribute-name-1=attribute-value-1
  <attribute-name-2=attribute-value-2 ...>;

```

or

```

XATTR ADD VAR variable-name-1 (attribute-name-1=attribute-value-1
  < attribute-name-2=attribute-value-2 ...>)

```

```
<variable-name-2 (attribute-name-1=attribute-value-1
<attribute-name-2=attribute-value-2 ...>) >;
```

XATTR DELETE;

XATTR OPTIONS <SEGLen=*number-of-bytes*>;

XATTR REMOVE DS *attribute-name(s)* ;

or

XATTR REMOVE VAR *variable-name-1 (attribute-name(s))*

```
<variable-name-2 (attribute-name(s) ...)>;
```

XATTR SET DS *attribute-name-1=attribute-value-1*

```
<attribute-name-2="attribute-value-2" ...>;
```

or

XATTR SET VAR *variable-name-1 (attribute-name-1=attribute-value-1*

```
<attribute-name-2=attribute-value-2 ... >)
```

```
<variable-name-2 (attribute-name-1=attribute-value-1
```

```
<attribute-name-2=attribute-value-2> ...)>;
```

XATTR UPDATE DS *attribute-name-1=attribute-value-1*

```
<attribute-name-2=attribute-value-2 ...>;
```

or

XATTR UPDATE VAR *variable-name-1 (attribute-name-1=attribute-value-1*

```
<attribute-name-2=attribute-value-2 ...>)
```

```
<variable-name-2 (attribute-name-1=attribute-value-1
```

```
<attribute-name-2=attribute-value-2 ...>)>;
```

REBUILD SAS-file </ <ENCRYPTKEY=*key-value*>

```
<ALTER=password> <GENNUM=n> <MEMTYPE=member-type>
```

```
<NOINDEX>>;
```

REPAIR </ <ENCRYPTKEY=*key-value*>

```
<ALTER=alter-password> <GENNUM=integer> <MEMTYPE=member-type>>;
```

SAVE SAS-file(s) </ MEMTYPE=*member-type*>;

Statement	Task	Example
PROC DATASETS	Manage SAS files	
AGE	Rename a group of related SAS files	Ex. 7
APPEND	Add observations from one SAS data set to the end of another SAS data set	Ex. 6
ATTRIB	Associate a format, informat, or label with variables in the SAS data set specified in the MODIFY statement	Ex. 1
AUDIT	Initiate, control, suspend, resume, or terminate event logging to an audit file	
CHANGE	Rename one or more SAS files	Ex. 2
CONTENTS	Describe the contents of one or more SAS data sets and prints a directory of the SAS library	Ex. 5

Statement	Task	Example
COPY	Copy all or some of the SAS files	Ex. 2
DELETE	Delete SAS files	Ex. 2
EXCHANGE	Exchange the names of two SAS files	Ex. 2
EXCLUDE	Exclude SAS files from copying	Ex. 2
FORMAT	Permanently assign, change, and remove variable formats	Ex. 4
IC CREATE	Create an integrity constraint	
IC DELETE	Delete an integrity constraint	
IC REACTIVATE	Reactivate a foreign key integrity constraint	
INDEX CENTILES	Update centiles statistics for indexed variables	
INDEX CREATE	Create simple or composite indexes	Ex. 4
INDEX DELETE	Delete one or more indexes	
INFORMAT	Permanently assign, change, and remove variable informats	Ex. 4
INITIATE	Create an audit file that has the same name as the SAS data file and a data set type of AUDIT	Ex. 8
LABEL	Assign, change, and remove variable labels	Ex. 4
LOG	Specify the settings for an audit file	Ex. 8
MODIFY	Change the attributes of a SAS file and the attributes of variables	Ex. 4
REBUILD	Specify whether to restore or delete the disabled indexes and integrity constraints	
RENAME	Rename variables in the SAS data set	Ex. 4
REPAIR	Attempt to restore damaged SAS data sets or catalogs	
RESUME	Resume event logging to the audit file, if the audit file was suspended	Ex. 8
SAVE	Delete all the SAS files except the ones listed in the SAVE statement	Ex. 3
SELECT	Select SAS files for copying	Ex. 2
SUSPEND	Suspend event logging to the audit file	Ex. 8
TERMINATE	Terminate event logging and deletes the audit file	Ex. 8

Statement	Task	Example
USER_VAR	Define optional variables to be logged in the audit file with each update to an observation	Ex. 8
XATTR ADD	Add an extended attribute to a variable or a data set	Ex. 9
XATTR DELETE	Delete all of the extended attributes from a data set	
XATTR OPTIONS	Specify options as needed for extended attributes	Ex. 9
XATTR REMOVE	Remove an extended attribute from a variable or a data set	
XATTR SET	Update or add an extended attribute to a variable or a data set	
XATTR UPDATE	Update an extended attribute of a variable or a data set	Ex. 9

PROC DATASETS Statement

Manages SAS files.

Syntax

```
PROC DATASETS <options>;
```

Summary of Optional Arguments

[ALTER=alter-password](#)

provides Alter access to any alter-protected SAS file in the SAS library.

[DETAILS](#)

[NODETAILS](#)

includes information in the log about the number of observations, number of variables, number of indexes, and data set labels.

[ENCRYPTKEY=key-value](#)

specifies the key value for AES encryption.

[FORCE](#)

forces either a RUN group to execute even when there are errors or forces an Append operation.

[GENNUM=ALL | HIST | REVERT | integer](#)

restricts processing for generation data sets.

KILL

deletes SAS files.

LIBRARY=*libref*

specifies the procedure input/output library.

MEMTYPE=(*member-type(s)*)

restricts processing to a certain type of SAS file.

NOLIST

suppresses the printing of the directory.

NOPRINT

suppresses the printing of the output to the log and listing.

NOWARN

suppresses error processing.

PW=*password*

provides Read, Write, or Alter access.

READ=*read-password*

provides Read access.

Optional Arguments

ALTER=*alter-password*

provides the Alter password for any alter-protected SAS files in the SAS library.

See [“Using Passwords with the DATASETS Procedure” on page 467](#)

DETAILS | NODETAILS

determines whether the following columns are written to the log:

Obs, Entries, or Indexes	gives the number of observations for SAS files of type AUDIT, DATA, and VIEW; the number of entries for type CATALOG; and the number of files of type INDEX that are associated with a data file, if any. If SAS cannot determine the number of observations in a SAS data set, the value in this column is set to missing. For example, in a very large data set, if the number of observations or deleted observations exceeds the number that can be stored in a double-precision integer, the count shows as missing. The value for type CATALOG is the total number of entries. For other types, this column is blank.
Vars	gives the number of variables for types AUDIT, DATA, and VIEW. If SAS cannot determine the number of variables in the SAS data set, the value in this column is set to missing. For other types, this column is blank.
Label	contains the label associated with the SAS data set. This column prints a label only for the type DATA.

TIP The value for files of type INDEX includes both user-defined indexes and indexes created by integrity constraints. To view index ownership and attribute information, use PROC DATASETS with the CONTENTS statement and the OUT2 option.

Note: The DETAILS option affects output only when a directory is specified and requires Read access to all read-protected SAS files in the SAS library. If you do not supply the Read password, the directory listing contains missing values for the columns produced by the DETAILS option.

Default If neither DETAILS or NODETAILS is specified, the default is the system option setting. The default system option setting is NODETAILS.

Tip If you are using the SAS windowing environment and specify the DETAILS option for a library that contains read-protected SAS files, a dialog box prompts you for each Read password that you do not specify in the PROC DATASETS statement. Therefore, you might want to assign the same Read password to all SAS files in the same SAS library.

Example [“Example 2: Manipulating SAS Files” on page 497](#)

ENCRYPTKEY=key-value

specifies the key value for AES encryption.

See [“Appending AES-Encrypted Data Sets” on page 395](#)

FORCE

performs two separate actions:

- forces a RUN group to execute even if errors are present in one or more statements in the RUN group. See [“RUN-Group Processing” on page 368](#) for a discussion of RUN-group processing and error handling.
- forces all APPEND statements to concatenate two data sets even when the variables in the data sets are not exactly the same. The APPEND statement drops the extra variables and issues a warning message to the SAS log unless the NOWARN option is specified (either with the APPEND statement or PROC DATASETS). For more information about the FORCE option, see [APPEND statement on page 389](#).

GENNUM=ALL | HIST | REVERT | integer

restricts processing for generation data sets. Valid values are as follows:

- | | |
|------|---|
| ALL | for subordinate CHANGE and DELETE statements, refers to the base version and all historical versions in a generation group. |
| HIST | for a subordinate DELETE statement, refers to all historical versions, but excludes the base version in a generation group. |

REVERT 0	for a subordinate DELETE statement, refers to the base version in a generation group and changes the most current historical version, if it exists, to the base version.
integer	for subordinate AUDIT, CHANGE, MODIFY, DELETE, and REPAIR statements, refers to a specific version in a generation group. Specifying a positive number is an absolute reference to a specific generation number that is appended to a data set name (that is, gennum=2 specifies MYDATA#002). Specifying a negative number is a relative reference to a historical version in relation to the base version, from the youngest to the oldest (that is, gennum=-1 refers to the youngest historical version).

See [“Restricting Processing for Generation Data Sets” on page 468](#)

KILL

deletes all SAS files in the SAS library that are available for processing. The MEMTYPE= option subsets the member types that the statement deletes. The following example deletes all the data files in the Work library:

```
proc datasets lib=work kill memtype=data; run; quit;
```

CAUTION

The KILL option deletes the SAS files immediately after you submit the statement. If the SAS file has an ALTER= password assigned, it must be specified in order to delete the SAS file.

LIBRARY=*libref*

names the library that the procedure processes. This library is the *procedure input/output library*.

Aliases DDNAME=

DD=

LIB=

Default Work or User

Note A SAS library that is accessed via a sequential engine (such as a tape format engine) cannot be specified as the value of the LIBRARY= option.

Example [“Example 2: Manipulating SAS Files” on page 497](#)

MEMTYPE=(*member-type(s)*)

restricts processing to one or more member types and restricts the listing of the data library directory to SAS files of the specified member types. For example, the following PROC DATASETS statement limits processing to SAS data sets in the default data library and limits the directory listing in the SAS log to SAS files of member type DATA:

```
proc datasets memtype=data;
```

Aliases **MTYPE=**

MT=

Default **ALL**

See [“Restricting Member Types for Processing” on page 469](#)

NOLIST

suppresses the printing of the directory of the SAS files in the SAS log and any open non-LISTING destination.

Note If you have ODS LISTING turned on and open a non-LISTING ODS destination, PROC DATASETS output goes to both the SAS log and the ODS destination. The NOLIST option suppresses output to both. To see the output in the SAS log only, use the ODS EXCLUDE statement by specifying the Members and Directory output objects. For example, if the RTF and LISTING destinations are both open and Directory and Members information is desired in the LOG window only, use the following:

```
ods rtf file="listing_nolist.rtf";
ods trace on;
ods rtf exclude directory members;

proc datasets lib=work;
run;
quit;

ods rtf close;
```

Example [“Example 4: Modifying SAS Data Sets” on page 505](#)

NOPRINT

suppresses the printing of the output and the printing of the directory of the SAS files in the log and any open non-LISTING destination. The NOPRINT option is a combination of the NOLIST option and the NOPRINT option in the CONTENTS statement.

NOWARN

suppresses the error processing that occurs when a SAS file that is specified in a SAVE, CHANGE, EXCHANGE, REPAIR, DELETE, or COPY statement or listed as the first SAS file in an AGE statement, is not in the procedure input library. When an error occurs and the NOWARN option is in effect, PROC DATASETS continues processing that RUN group. If NOWARN is not in effect, PROC DATASETS stops processing that RUN group and issues a warning for all operations except DELETE, for which it does not stop processing.

PW= password

provides the password for any protected SAS files in the SAS library. PW= can act as an alias for READ=, WRITE=, or ALTER=.

See [“Using Passwords with the DATASETS Procedure” on page 467](#)

READ=*read-password*

provides the Read password for any read-protected SAS files in the SAS library.

See [“Using Passwords with the DATASETS Procedure” on page 467](#)

AGE Statement

Renames a group of related SAS files in a library.

Example: [“Example 7: Aging SAS Data Sets” on page 514](#)

Syntax

AGE *current-name related-SAS-file(s)*

</ <**ALTER=***alter-password*> <**MEMTYPE=***member-type*>>;

Required Arguments

current-name

is a SAS file that the procedure renames. *current-name* receives the name of the first name in *related-SAS-file(s)*.

related-SAS-file(s)

is one or more SAS files in the SAS library.

Optional Arguments

ALTER=*alter-password*

provides the Alter password for any alter-protected SAS files named in the AGE statement. Because an AGE statement renames and deletes SAS files, you need Alter access to use the AGE statement. You can use the option either in parentheses after the name of each SAS file or after a forward slash.

See [“Using Passwords with the DATASETS Procedure” on page 467](#)

MEMTYPE=*member-type*

restricts processing to one member type. All of the SAS files that you name in the AGE statement must be the same member type. You can use the option after a forward slash after the name of each SAS file.

Aliases **MTYPE=**

MT=

Default If you do not specify **MEMTYPE=** in the PROC DATASETS statement, the default is DATA.

See [“Restricting Member Types for Processing” on page 469](#)

Details

The AGE statement renames the *current-name* to the name of the first name in the *related-SAS-files*, renames the first name in the *related-SAS-files* to the second name in the *related-SAS-files*, and so on, until it changes the name of the next-to-last SAS file in the *related-SAS-files* to the last name in the *related-SAS-files*. The AGE statement then deletes the last file in the *related-SAS-files*.

If the first SAS file named in the AGE statement does not exist in the SAS library, PROC DATASETS stops processing the RUN group containing the AGE statement and issues an error message. The AGE statement does not age any of the *related-SAS-files*. To override this behavior, use the NOWARN option in the PROC DATASETS statement.

If one of the *related-SAS-files* does not exist, the procedure prints a warning message to the SAS log but continues to age the SAS files that it can.

If you age a data set that has an index, the index continues to correspond to the data set.

You can age only entire generation groups. For example, if data sets A and B have generation groups, then the following statement deletes generation group B and ages (renames) generation group A to the name B:

```
age a b;
```

For example, suppose the generation group for data set A has three historical versions and the generation group for data set B has two historical versions. Then aging A to B has this effect:

Table 15.3 Generation Groups

Old Name	Version	New Name	Version
A	base	B	base
A	1	B	1
A	2	B	2
A	3	B	3
B	base	is deleted	
B	1	is deleted	
B	2	is deleted	

APPEND Statement

Adds the observations from one SAS data set to the end of another SAS data set.

Default:	If the BASE= data set is accessed through a SAS server and if no other user has the data set open at the time the APPEND statement begins processing, the BASE= data set defaults to CNTLLEV=MEMBER (member-level locking). When this behavior happens, no other user can update the file while the data set is processed.
Restriction:	The BASE= option cannot be an existing CAS table.
Requirement:	The BASE= data set must be a member of a SAS library that supports update processing.
Notes:	If you use the DROP=, KEEP=, or RENAME= options on the BASE= data set, the options ONLY affect the APPEND processing and does not change the variables in the appended BASE= data set. Variables that are dropped or not kept using the DROP= and KEEP= options still exist in the appended BASE= data set. Variables that are renamed using the RENAME= option remain with their original name in the appended BASE= data set. The CAS engine does not support update processing and therefore an existing CAS table cannot be specified with BASE=.
Tips:	You can specify most data set options for the BASE= argument and DATA= option. However, if you specify DROP=, KEEP=, or RENAME= data set option for the BASE= data set, the option is ignored. You can use any global statements as well. If a failure occurs during processing, the data set is marked as damaged and is reset to the pre-append condition at the next REPAIR statement. If the data set has an index, the index is not updated with each observation but is updated once at the end. (This behavior is Version 7 and later, as long as APPENDVER=V6 is not set.) Each password and encryption key option must be coded on a separate line to ensure that they are properly blotted in the log.
Example:	“Example 6: Concatenating Two SAS Data Sets” on page 511

Syntax

```

APPEND BASE=<libref.>SAS-data-set
<APPENDVER=V6>
<DATA=<libref.>SAS-data-set>
<ENCRYPTKEY=key-value>
<FORCE>
<GETSORT>
<NOWARN>;

```

Required Argument

BASE=<libref.> SAS-data-set

names the data set to which you want to add observations.

libref

specifies the library that contains the SAS data set. If you omit the libref, the default is the libref for the procedure input library. If you are using PROC APPEND, the default for libref is either Work or User.

SAS-data-set

names a SAS data set. If the APPEND statement cannot find an existing data set with this name, it creates a new data set in the library. That is, you can use the APPEND statement to create a data set by specifying a new data set name in the BASE= argument.

Whether you are creating a new data set or appending to an existing data set, the BASE= data set is the current SAS data set after all Append operations.

Alias OUT=

Example [“Example 6: Concatenating Two SAS Data Sets” on page 511](#)

Optional Arguments

APPENDVER=V6

uses the Version 6 behavior for appending observations to the BASE= data set, which is to append one observation at a time. Beginning in Version 7, to improve performance, the default behavior changed so that all observations are appended after the data set is processed.

See [“Appending to an Indexed Data Set – Fast-Append Method” on page 396](#)

DATA=<libref.> SAS-data-set

names the SAS data set containing observations that you want to append to the end of the SAS data set specified in the BASE= argument.

libref

specifies the library that contains the SAS data set. If you omit libref, the default is the libref for the procedure input library. The DATA= data set can be from any SAS library. You must use the two-level name if the data set resides in a library other than the procedure input library.

SAS-data-set

names a SAS data set. If the APPEND statement cannot find an existing data set with this name, it stops processing.

Alias NEW=

Default the most recently created SAS data set, from any SAS library

See [“Appending with Generation Groups” on page 399](#)

Example [“Example 6: Concatenating Two SAS Data Sets” on page 511](#)

ENCRYPTKEY=key-value

specifies the key value for AES encryption.

See [“Appending AES-Encrypted Data Sets” on page 395](#)

FORCE

forces the APPEND statement to concatenate data sets when the DATA= data set contains variables that meet one of the following criteria:

- are not in the BASE= data set
- do not have the same type as the variables in the BASE= data set
- are longer than the variables in the BASE= data set

Tip You can use the GENNUM= data set option to append to or from a specific version in a generation group. Here are some examples:

```
/* appends historical version to base A */
proc datasets;
  append base=a
         data=a (gennum=2);

/* appends current version of A to historical version */
proc datasets;
  append base=a (gennum=1)
         data=a;
```

See [“Appending to Data Sets with Different Variables” on page 397](#) and [“Appending to Data Sets That Contain Variables with Different Attributes” on page 398](#)

Example [“Example 6: Concatenating Two SAS Data Sets” on page 511](#)

GETSORT

copies the sort indicator from the DATA= data set to the BASE= data set. The sort indicator is established by either a PROC SORT or an ORDERBY clause in PROC SQL if the following criteria are met:

- The BASE= data set must meet the following criteria:
 - ☐ be SAS Version 7 or later
 - ☐ contain no observations
 - ☐ accept sort indicators

CAUTION

Any pre-existing sort indicator on the BASE= data set is overwritten with no warning, even if the DATA= data set is not sorted at all.

- The DATA= data set must meet the following criteria:
 - ☐ contain a sort indicator established by PROC SORT
 - ☐ be the same data representation as the BASE= data set

Restrictions The GETSORT option has no effect on the data sets if the BASE= data set has an audit trail associated with it. This restriction causes a WARNING in the output while the APPEND process continues.

The GETSORT option has no effect on the data sets if there are dropped, kept, or renamed variables in the DATA= data file.

NOWARN

suppresses the warning when used with the FORCE option to concatenate two data sets with different variables.

Details

Using the APPEND Procedure Instead of the APPEND Statement

The only difference between the APPEND procedure and the APPEND statement in PROC DATASETS, is the default for libref in the BASE= and DATA= arguments. For PROC APPEND, the default is either Work or User. For the APPEND statement, the default is the libref of the procedure input library.

You can use the WHERE= data set option with the DATA= data set to restrict the observations that are appended. Similarly, you can use the WHERE statement to restrict the observations from the DATA= data set. The WHERE statement has no effect on the BASE= data set.

CAUTION

If there is a WHERE option used on an existing BASE= data set, it is used only if the WHEREUP= option is set to YES. If the BASE= data set does not exist, then no WHEREUP= option is needed for the WHERE option to take effect.

Appending Sorted Data Sets

You can append sorted data sets and maintain the sort using the following guidelines:

- The DATA= data set and the BASE= data set contain sort indicators from the SORT procedure.
- The DATA= data set and the BASE= data set are sorted using the same variables.
- The observations added from the DATA= data set do not violate the sort order of the BASE= data set.

The sort indicator from the BASE= data set is retained.

Using the Block I/O Method to Append

Note: The block I/O method cannot be used when appending a CAS table to a SAS data set.

The block I/O method is used to append blocks of data instead of one observation at a time. This method increases performance when you are appending large data sets. SAS determines whether to use the block I/O method. Not all data sets can use the block I/O method. There are restrictions set by the APPEND statement and the Base SAS engine.

To display information in the SAS log about the append method that is being used, you can specify the MSGLEVEL= system option as follows:

```
options msglevel=i;
```

The following message is written to the SAS log, if the block I/O method is not used:

INFO: Data set block I/O cannot be used because:

If the APPEND statement determines that the block I/O will not be used, one of the following explanations is written to the SAS log:

INFO: - The data sets use different engines, have different variables or have attributes that might differ.

INFO: - There is a WHERE clause present.

INFO: - There is no member level locking.

INFO: - The OBS option is active.

INFO: - The FIRSTOBS option is active.

If the Base SAS engine determines that the block I/O method will not be used, one of the following explanations is written to the SAS log:

INFO: - Referential Integrity Constraints exist.

INFO: - Cross Environment Data Access is being used.

INFO: - The file is compressed.

INFO: - The file has an audit file which is not suspended.

Restricting the Observations That Are Appended

You can use the WHERE= data set option with the DATA= data set in order to restrict the observations that are appended. Likewise, you can use the WHERE statement in order to restrict the observations from the DATA= data set. The WHERE statement has no effect on the BASE= data set. If you use the WHERE= data set option with the BASE= data set, WHERE= has no effect.

CAUTION

For an existing BASE= data set: If there is a WHERE statement in the BASE= data set, it takes effect only if the WHEREUP= option is set to YES.

CAUTION

For the non-existent BASE= data set: If there is a WHERE statement in the non-existent BASE= data set, regardless of the WHEREUP option setting, you use the WHERE statement.

Note: You cannot append a data set to itself by using the WHERE= data set option.

Choosing between the SET Statement and the APPEND Statement

If you use the SET statement in a DATA step to concatenate two data sets, SAS must process all the observations in both data sets to create a new one. The APPEND statement bypasses the processing of data in the original data set and adds new observations directly to the end of the original data set. Using the APPEND statement can be more efficient than using a SET statement if any of the following occurs:

- The BASE= data set is large.
- All variables in the BASE= data set have the same length and type as the variables in the DATA= data set and if all variables exist in both data sets.

Note: You can use the CONTENTS statement to see the variable lengths and types.

The APPEND statement is especially useful if you frequently add observations to a SAS data set (for example, in production programs that are constantly appending data to a journal-type data set).

Appending Password-Protected SAS Data Sets

In order to use the APPEND statement, you need Read access to the DATA= data set and Write access to the BASE= data set. To gain access, use the READ= and WRITE= data set options in the APPEND statement in parentheses immediately after the data set name. When you are appending password-protected data sets, use the following guidelines:

- If you do not give the Read password for the DATA= data set in the APPEND statement, by default the procedure looks for the Read password for the DATA= data set in the PROC DATASETS statement. However, the procedure does not look for the Write password for the BASE= data set in the PROC DATASETS statement. Therefore, you must specify the Write password for the BASE= data set in the APPEND statement.
- If the BASE= data set is read-protected only, you must specify its Read password in the APPEND statement.

Appending AES-Encrypted Data Sets

When appending two AES-encrypted data sets, you must use the ENCRYPTKEY= data set option for access to the data sets. Here is an example using the ENCRYPTKEY= option on both data sets:

```
proc datasets;
  append base=a (encryptkey=secret)
    data=a (encryptkey=j1gh56);
run;
```

When appending to a non-encrypted data set, you must specify the ENCRYPTKEY= on the DATA= data set. Then, you can append to the data set even if the BASE= data set engine does not support AES encryption. The data appended will not be encrypted.

```
proc datasets;
  append base=a
    data=a (encryptkey=key-value);
run;
```

For more information about AES encryption, see [“AES or AES2” in SAS Data Set Options: Reference](#). For more information about the ENCRYPTKEY= data set option, see [“ENCRYPTKEY= Data Set Option” in SAS Data Set Options: Reference](#).

Appending to a Compressed Data Set

You can concatenate compressed SAS data sets. Either or both of the BASE= and DATA= data sets can be compressed. If the BASE= data set allows the reuse of space from deleted observations, the APPEND statement might insert the observations into the middle of the BASE= data set to use available space.

Either or both of the BASE= SAS data set and DATA= data set or table can be compressed. If the BASE= data set allows the reuse of space from deleted rows, the APPEND statement might insert the rows into the middle of the BASE= data set.

For information about the COMPRESS= and REUSE= data set and system options, see [SAS Data Set Options: Reference](#) and [SAS System Options: Reference](#).

PROC APPEND places all of the new observations at the end of the BASE= data set and tries to reuse space, which is part of the fast-append behavior that was added in SAS 7. The fast-append behavior was added for better performance writing to the BASE data set and updating indexes. If you want to reuse space, then you need to add the APPENDVER=v6 option to the PROC DATASETS APPEND statement. PROC DATASETS APPEND reuses space only when the new observations fit into the space previously occupied by the deleted observations.

```
proc append base=libref.data-set data=librefdata-set appendver=v6;
run;
```

Appending to Data Sets That Contain Variables with Different Attributes

If a variable has different attributes in the BASE= SAS data set than it does in the DATA= data set or table, the attributes in the BASE= data set prevail.

If the formats in the DATA= SAS data set or table are different from those in the BASE= SAS data set, then the formats in the BASE= data set are used. However, the data from the DATA= data set or table is not converted in order to be consistent with the formats in the BASE= data set. The result could be data that seems to be incorrect. A warning message is displayed in the log.

Use the FORCE option if one of the following occurs:

- if the length of a variable is longer in the DATA= SAS data set or table than in the BASE= data set
- if the same variable is a character variable in one data set or table and a numeric variable in the other

Using FORCE has the following consequences:

- The length of the variables in the BASE= SAS data set takes precedence. The values might be truncated from the DATA= data set or table to fit them into the length that is specified in the BASE= data set.
- The type of the variables in the BASE= data set takes precedence. The APPEND statement replaces values of the wrong type (all values for the variable in the DATA= data set or table) with missing values.

Appending to an Indexed Data Set — Fast-Append Method

Beginning with Version 7, the behavior of appending to an indexed data set changed to improve performance.

- In Version 6, when you appended to an indexed data set, the index was updated for each added observation. Index updates tend to be random. Therefore, disk I/O could have been high.
- Currently, SAS does not update the index until all observations are added to the data set. After the append, SAS internally sorts the observations and inserts the data into the index sequentially. The behavior reduces most of the disk I/O and results in a faster append method.

The fast-append method is used by default when the following requirements are met. Otherwise, the Version 6 method is used:

- The BASE= data set is open for member-level locking. If CNTLLEV= is set to record, then the fast-append method is not used.
- The BASE= data set does not contain referential integrity constraints.
- The BASE= data set is not accessed using the Cross Environment Data Access (CEDA) facility.
- The BASE= data set is not using a WHERE= data set option.

To display information in the SAS log about the append method that is being used, you can specify the MSGLEVEL= system option as follows:

```
options msglevel=i;
```

Either a message appears if the fast-append method is in use, or a message or messages appear as to why the fast-append method is not in use.

The current append method initially adds observations to the BASE= data set regardless of the restrictions that are determined by the index. For example, a

variable that has an index that was created with the UNIQUE option does not have its values validated for uniqueness until the index is updated. Then, if a nonunique value is detected, the offending observation is deleted from the data set. After observations are appended, some of them might subsequently be deleted.

For a simple example, consider that the BASE= data set has ten observations numbered from 1 to 10 with a UNIQUE index for the variable ID. You append a data set that contains five observations numbered from 1 to 5, and observations 3 and 4 both contain the same value for ID. The following actions occur:

- 1 After the observations are appended, the BASE= data set contains 15 observations numbered from 1 to 15.
- 2 SAS updates the index for ID, validates the values, and determines that observations 13 and 14 contain the same value for ID.
- 3 SAS deletes one of the observations from the BASE= data set, resulting in 14 observations that are numbered from 1 to 15. For example, observation 13 is deleted. Note that you cannot predict which observation is deleted, because the internal sort might place either observation first. (In Version 6, you could predict that observation 13 would be added and observation 14 would be rejected.)

If you do not want the current behavior (which could result in deleted observations) or if you want to be able to predict which observations are appended, request the Version 6 append method by specifying the APPENDVER=V6 option:

```
proc datasets;
  append base=a data=b appendver=v6;
run;
```

Note: In Version 6, deleting the index and then re-creating it after the append could improve performance. The current method might eliminate the need to do that. However, the performance depends on the nature of your data.

Appending to Data Sets with Different Variables

If the DATA= data set contains variables that are not in the BASE= data set, use the FORCE option in the APPEND statement to force the concatenation of the two data sets. The APPEND statement drops the extra variables and issues a warning message. You can use the NOWARN option to suppress the warning message.

If the BASE= data set contains a variable that is not in the DATA= data set, the APPEND statement concatenates the data sets, but the observations from the DATA= data set have a missing value for the variable that was not present in the BASE= data set. The FORCE option is not necessary in this case.

If you use the DROP=, KEEP=, or RENAME= options on the BASE= data set, the options ONLY affect the APPEND processing and does not change the variables in the appended BASE= data set. Variables that are dropped or not kept using the DROP= and KEEP= options still exist in the appended BASE= data set. Variables that are renamed using the RENAME= option remain with their original name in the appended BASE= data set.

Appending to Data Sets That Contain Variables with Different Attributes

If a variable has different attributes in the BASE= data set than it does in the DATA= data set, the attributes in the BASE= data set prevail.

If the SAS formats in the DATA= data set are different from those in the BASE= data set, then the SAS formats in the BASE= data set are used. However, SAS does not convert the data from the DATA= data set in order to be consistent with the SAS formats in the BASE= data set. The result could be data that seems to be incorrect. A warning message is displayed in the SAS log. The following example illustrates appending data by using different SAS formats:

```
data format1;
    input Date date9.;
    format Date date9.;
datalines;
24sep1975
22may1952
;

data format2;
    input Date datetime20.;
    format Date datetime20.;
datalines;
25aug1952:11:23:07.4
;

proc append base=format1 data=format2;
run;
```

The following messages are displayed in the SAS log.

```
NOTE: Appending WORK.FORMAT2 to WORK.FORMAT1.
WARNING: Variable Date has format DATE9. on the BASE data set
        and format DATETIME20. on the DATA data set. DATE9. used.
NOTE: There were 1 observations read from the data set WORK.FORMAT2.
NOTE: 1 observations added.
NOTE: The data set WORK.FORMAT1 has 3 observations and 1 variables.
```

If the length of a variable is longer in the DATA= data set than in the BASE= data set, or if the same variable is a character variable in one data set and a numeric variable in the other, use the FORCE option. Using FORCE has the following consequences:

- The length of the variables in the BASE= data set takes precedence. SAS truncates values from the DATA= data set to fit them into the length that is specified in the BASE= data set.
- The type of the variables in the BASE= data set takes precedence. The APPEND statement replaces values of the wrong type (all values for the variable in the DATA= data set) with missing values.

Note: If a character variable's transcoding attribute is opposite in the BASE= and DATA= data sets (for example, one is YES and the other is NO), then a warning is issued. To determine the transcoding attributes, use the CONTENTS procedure for

each data set. You set the transcoding attribute with the `TRANSCODE=` option in the `ATTRIB` statement or with the `TRANSCODE=` column modifier in `PROC SQL`.

Appending Data Sets That Contain Integrity Constraints

If the `DATA=` data set contains integrity constraints and the `BASE=` data set does not exist, the `APPEND` statement copies the general constraints. Note that the referential constraints are not copied. If the `BASE=` data set exists, the `APPEND` action copies only observations.

Appending to a Data Set with Integrity Constraints or Indexes

Here is a simple way to get the indexes and integrity constraints from an existing data set and then apply them to another data set. Note that `Work.Model` should already exist and have observations:

```
Proc contents data=sashelp.zipcode out2=work.ConstraintsIndexes;
run;

proc sql noprint;
    select recreate into :macroVarCIs separated by ' '
        from work.ConstraintsIndexes;
quit;

data work.test_zipcode;
    set sashelp.zipcode;
run;

proc datasets lib=work nolist;
    modify test_zipcode;
    &macroVarCIs
quit;

proc contents data=work.test_zipcode;
run;
```

Appending with Generation Groups

You can use the `GENNUM=` data set option to append to a specific version in a generation group. Here are examples:

Table 15.4 *Appending with Generation Groups*

SAS Statements	Result
<pre>proc datasets; append base=a data=b(gennum=2) ;</pre>	Appends historical version B#002 to base A
<pre>proc datasets; append base=a(gennum=2) data=b(gennum=2) ;</pre>	Appends historical version B#002 to historical version A#002

System Failures

If a system failure or some other type of interruption occurs while the procedure is executing, the Append operation might not be successful; it is possible that not all, perhaps none, of the observations are added to the BASE= data set. In addition, the BASE= data set might suffer damage. The Append operation performs an update in place, which means that it does not make a copy of the original data set before it begins to append observations. If you want to be able to restore the original observations, you can initiate an audit trail for the base data file and choose to store a before-update image of the observations. Then you can write a DATA step to extract and reapply the original observations to the data file. For information about initiating an audit trail, see [“AUDIT Statement” on page 401](#).

ATTRIB Statement

Associates a format, informat, or label with variables in the SAS data set specified in the MODIFY statement.

- Restriction: The ATTRIB statement must appear in a MODIFY RUN group
- Note: The ATTRIB statement does not affect the CONTENTS statement output. CONTENTS reports the labels, informats, and formats on the actual member.
- Example: [AUDIT statement on page 492](#)

Syntax

ATTRIB *variable-list(s) attribute-list(s);*

Required Arguments

variable-list(s)

names the variables that you want to associate with the attributes. You can list the variables in any form that SAS allows.

attribute-list(s)

specifies one or more attributes to assign to *variable-list*. Specify one or more of the following attributes in the ATTRIB statement:

FORMAT=*format*

associates a format with variables in *variable-list*.

Tip The format can be either a standard SAS format or a format that is defined with the FORMAT procedure.

INFORMAT=*informat*

associates an informat with variables in *variable-list*.

Tip The informat can be either a standard SAS informat or an informat that is defined with the FORMAT procedure.

LABEL=*'label'*

associates a label with variables in the *variable-list*.

Details

Within the DATASETS procedure, the ATTRIB statement must be used in a MODIFY RUN group and can use only the FORMAT, INFORMAT, and LABEL options. The ATTRIB statement is the simplest way to remove or change all variable labels, formats, or informats in a data set using the keyword `_ALL_`. For an example, see [“Example 1: Removing All Labels and Formats in a Data Set” on page 492](#).

If you are not deleting or changing all attributes, it is easier to use the following statements, [LABEL statement on page 444](#), [FORMAT Statement on page 434](#), and [INFORMAT statement on page 442](#).

AUDIT Statement

Initiates and controls event logging to an audit file as well as suspends, resumes, or terminates event logging in an audit file.

Tips:

The AUDIT statement takes one of two forms, depending on whether you are initiating the audit trail or suspending, resuming, or terminating event logging in an audit file.

You can define attributes such as format and informat for the user variables in the data file by using the PROC DATASETS MODIFY statement.

Syntax

AUDIT *SAS-file* <(SAS-password <ENCRYPTKEY=*key-value*>
<GENNUM=*integer*>)>;

INITIATE <AUDIT_ALL=NO | YES>;

LOG<ADMIN_IMAGE=YES | NO>

<BEFORE_IMAGE=YES | NO>

<DATA_IMAGE=YES | NO>

<ERROR_IMAGE=YES | NO>;

<SUSPEND | RESUME | TERMINATE; >

<USER_VAR *variable(s)* >;

Required Argument

SAS-file

specifies the SAS data file in the procedure input library that you want to audit.

Optional Arguments

SAS-password

specifies the password for the SAS data file, if one exists. The parentheses are required.

ENCRYPTKEY=key-value

specifies the key value for AES encryption.

GENNUM=integer

specifies that the SUSPEND, RESUME, or TERMINATE action be performed on the audit trail of a generation file. You cannot initiate an audit trail on a generation file. Valid values for GENNUM= are *integers*, which is a number that references a specific version from a generation group. Specifying a positive number is an absolute reference to a specific generation number that is appended to a data set's name (that is, **gennum=2** specifies MYDATA#002). Specifying a negative number is a relative reference to a historical version in relation to the base version, from the youngest to the oldest (that is, **gennum=-1** refers to the youngest historical version). Specifying 0, which is the default, refers to the base version. The parentheses are required.

Restriction The GENNUM= option cannot be specified before an INITIATE or USER_VAR statement.

Details

Creating an Audit File

Note: The initiation of an audit trail is possible only with the Base SAS engine.

The following example creates the audit file MyLib.MyFile.audit to log updates to the data file MyLib.MyFile.data, storing all available record images:

```
proc datasets library=mylib;
  audit myfile (alter=password);
  initiate;
run;
```

The following example creates the same audit file but stores only error record images:

```
proc datasets library=mylib;
  audit myfile (alter=password);
  initiate;
  log data_image=no
    before_image=no
    data_image=no;
run;
```

The following example initiates an audit file using AUDIT_ALL=YES:

```
proc datasets lib=mylib; /* all audit image types will be logged
```



```

                                and the file cannot be suspended */
audit myfile (alter=password);
initiate audit_all=yes;
quit;

```

The following example terminates an audit file:

```

proc datasets lib=mylib;
    audit myfile (alter=password);
    terminate;
quit;

```

The AUDIT statement starts an *audit run group* for a file. Multiple audit run groups for that file can be submitted in the following ways:

- in the same PROC DATASETS step
- in separate PROC DATASETS steps
- in separate Compute Server sessions

All audit file related statements (INITIATE, USER_VAR, LOG, SUSPEND, RESUME, TERMINATE) must be preceded by an AUDIT statement, which identifies the file that they apply to.

The INITIATE statement creates an audit file and must be submitted in the first AUDIT statement occurrence. No other audit-related statement, such as USER_VAR, LOG, SUSPEND, RESUME, or TERMINATE will be valid for that audit file until the INITIATE statement has been submitted. You can initiate an audit file on a generation data set, but it must be the latest generation of the generation group. You cannot specify a GENNUM to identify the latest generation. You will get the latest generation by default.

Here is an example of the AUDIT statement in the first AUDIT RUN group for a given file:

```
AUDIT file <(SAS-password)>;
```

Once an audit file has been initiated, the AUDIT statements that follow the INITIATE statement for that file can specify a GENNUM:

```
AUDIT file <(<SAS-password><GENNUM=integer>)>;
```

The USER_VAR statement must directly follow the INITIATE statement in the same AUDIT RUN group.

CHANGE Statement

Renames one or more SAS files in the same SAS library.

Example: [“Example 2: Manipulating SAS Files” on page 497](#)

Syntax

```
CHANGE old-name-1=new-name-1
<old-name-2=new-name-2 ...>
</ <ENCRYPTKEY=key-value>
<ALTER=alter-password> <GENNUM=ALL | integer> <MEMTYPE=member-type>>;
```

Required Argument

old-name=new-name

changes the name of a SAS file in the input data library. *old-name* must be the name of an existing SAS file in the input data library.

Example [“Example 2: Manipulating SAS Files” on page 497](#)

Optional Arguments

ENCRYPTKEY=key-value

specifies the key value for AES encryption. This option is needed only if RELATIVE GENNUM is specified. For more information, see [“Library Contents and AES Encryption” on page 410](#).

ALTER=alter-password

provides the Alter password for any alter-protected SAS files named in the CHANGE statement. Because a CHANGE statement changes the names of SAS files, you need Alter access to use the CHANGE statement for *new-name*. You can use the option either in parentheses after the name of each SAS file or after a forward slash.

See [“Using Passwords with the DATASETS Procedure” on page 467](#)

GENNUM=ALL | integer

restricts processing for generation data sets. You can use the option either in parentheses after the name of each SAS file or after a forward slash. The following list shows valid values:

ALL

0

refers to the base version and all historical versions of a generation group.

integer

refers to a specific version from a generation group. Specifying a positive number is an absolute reference to a specific generation number that is appended to a data set's name (that is, **gennum=2** specifies MYDATA#002). Specifying a negative number is a relative reference to a historical version in relation to the base version, from the youngest to the oldest (that is, **gennum=-1** refers to the youngest historical version).

For example, the following statements change the name of version A#003 to base B:

```
proc datasets;
  change A=B / gennum=3;
```

```
proc datasets;
  change A (gennum=3) =B;
```

The following CHANGE statement produces an error:

```
proc datasets;
  change A (gennum=3) =B (gennum=3) ;
```

See [“Restricting Processing for Generation Data Sets” on page 468](#).

MEMTYPE=member-type

restricts processing to one member type. You can use the option either in parentheses after the name of each SAS file or after a forward slash.

Aliases MTYPE=

MT=

Default If you do not specify MEMTYPE= in the PROC DATASETS statement, the default is MEMTYPE=ALL.

See [“Restricting Member Types for Processing” on page 469](#)

Details

The CHANGE statement changes names by the order in which the *old-names* occur in the directory listing, not in the order in which you list the changes in the CHANGE statement.

If the *old-name* SAS file does not exist in the SAS library, PROC DATASETS stops processing the RUN group containing the CHANGE statement and issues an error message. To override this behavior, use the NOWARN option in the PROC DATASETS statement.

If you change the name of a data set that has an index, the index continues to correspond to the data set.

CONTENTS Statement

Describes the contents of one or more SAS data sets and prints the directory of the SAS library.

Restriction: You cannot use the WHERE option to affect the output because PROC CONTENTS does not process any observations.

Notes: The ATTRIB statement does not affect the CONTENTS statement output. CONTENTS reports the labels, informats, and formats on the actual member. When using the CONTENTS statement with SAS/ACCESS LIBNAME engines, see [“Differences in the DATASETS Procedure Output When Using SAS/ACCESS LIBNAME Engines” on page 375](#).

Tip: You can use data set options with the DATA=, OUT=, and OUT2= options. You can use any global statements as well.

Example: [“Example 5: Describing a SAS Data Set” on page 508](#)

Syntax

CONTENTS *<options>*;

Optional Arguments

CENTILES

prints centiles information for indexed variables.

The following additional fields are printed in the default report of PROC CONTENTS when the CENTILES option is selected and an index exists on the data set. Note that the additional fields depend on whether the index is simple or complex.

#	number of the index on the data set.
Index	name of the index.
Update Centiles	percentage of the data values that must be changed before the CENTILES for the indexed variables are automatically updated.
Current Update Percentage	percentage of index updated since CENTILES were refreshed.
# of Unique Values	number of unique indexed values.
Variables	names of the variables used to make up the index. Centile information is listed below the variables.

DATA=SAS-file-specification

specifies an entire library or a specific SAS data set within a library. *SAS-file-specification* can take one of the following forms:

*<libref.>*SAS-data-set

names one SAS data set to process. The default for libref is the libref of the procedure input library. For example, to obtain the contents of the SAS data set HtWt from the procedure input library, use the following CONTENTS statement:

```
contents data=HtWt;
```

To obtain the contents of a specific version from a generation group, use the GENNUM= data set option as shown in the following CONTENTS statement:

```
contents data=HtWt (gennum=3) ;
```

*<libref.>*_ALL_

gives you information about all SAS data sets that have the type or types specified by the MEMTYPE= option. *libref* refers to the SAS library. The default for libref is the libref of the procedure input library.

- If you are using the `_ALL_` keyword, you need Read access to all read-protected SAS data sets in the SAS library.
- `DATA=_ALL_` automatically prints a listing of the SAS files that are contained in the SAS library. Note that for SAS views, all librefs that are associated with the views must be assigned in the current session in order for them to be processed for the listing.

Default most recently created data set in your job or session, from any SAS library.

Tip If you specify a read-protected data set in the `DATA=` option but do not give the Read password, by default the procedure looks in the PROC DATASETS statement for the Read password. However, if you do not specify the `DATA=` option and the default data set (last one created in the session) is Read protected, the procedure does not look in the PROC DATASETS statement for the Read password.

Example [“Example 5: Describing a SAS Data Set” on page 508](#)

DETAILS | NODETAILS

includes information in the output about the number of observations, number of variables, number of indexes, and data set labels. DETAILS includes these additional columns of information in the output, but only if DIRECTORY is also specified.

Default If neither DETAILS nor NODETAILS is specified, the defaults are as follows: for the CONTENTS procedure, the default is the system option setting, which is NODETAILS; for the CONTENTS statement, the default is whatever is specified in the PROC DATASETS statement, which also defaults to the system option setting.

See a description of the additional columns in the Optional Argument section of [PROC DATASETS Statement on page 382](#)

DIRECTORY

prints a list of all SAS files in the specified SAS library. If DETAILS is also specified, using DIRECTORY causes the additional columns described in [DETAILS | NODETAILS on page 407](#) to be printed.

ENCRYPTKEY=key-value

specifies the key value for AES encryption. For more information, see [“Library Contents and AES Encryption” on page 410](#).

FMTLEN

prints the length of the informat or format. If you do not specify a length for the informat or format when you associate it with a variable, the length does not appear in the output of the CONTENTS statement unless you use the FMTLEN option. The length also appears in the FORMATL or INFORML variable in the output data set.

MEMTYPE=(member-type(s))

restricts processing to one or more member types. The CONTENTS statement produces output only for member types DATA, VIEW, and ALL, which includes DATA and VIEW.

MEMTYPE= in the CONTENTS statement differs from MEMTYPE= in most of the other statements in the DATASETS procedure in the following ways:

- A slash does not precede the option.
- You cannot enclose the MEMTYPE= option in parentheses to limit its effect to only the SAS file immediately preceding it.

MEMTYPE= results in a directory of the library in which the DATA= member is located. However, MEMTYPE= does not limit the types of members whose contents are displayed unless the _ALL_ keyword is used in the DATA= option. For example, the following statements produce the contents of only the SAS data sets with the member type DATA:

```
proc datasets memtype=data;
    contents data=_all_;
run;
```

Aliases MTYPE=

 MT=

Default DATA

NODS

suppresses printing the contents of individual files when you specify _ALL_ in the DATA= option. The CONTENTS statement prints only the SAS library directory. You cannot use the NODS option when you specify only one SAS data set in the DATA= option.

NOPRINT

suppresses printing the output of the CONTENTS statement.

ORDER=COLLATE | CASECOLLATE | IGNORECASE | VARNUM

COLLATE prints a list of variables in alphabetical order beginning with uppercase and then lowercase names.

CASECOLLATE prints a list of variables in alphabetical order even if they include mixed-case names and numerics.

IGNORECASE prints a list of variables in alphabetical order ignoring the case of the letters.

VARNUM is the same as the VARNUM option.

Note The ORDER= option does not affect the order of the OUT= and OUT2= data sets.

See [“VARNUM” on page 409](#)

Example See [“Example 3: Using the ORDER= Option” on page 302](#) to compare the default and the four options for ORDER=.

OUT=SAS-data-set

names an output SAS data set.

Tip OUT= does not suppress the printed output from the statement. If you want to suppress the printed output, you must use the NOPRINT option.

See [“The OUT= Data Set” on page 485](#) for a description of the variables in the OUT= data set.

OUT2=SAS-data-set

names the output data set to contain information about indexes and integrity constraints.

Note When you use the OUT2=*PermanentLibrary_ALL_* option within PROC CONTENTS or PROC DATASETS with the CONTENTS statement, you must also set the REPLACE=YES data set option or the REPLACE system option.

Tips If UPDATECENTILES was not specified in the index definition, then the default value of 5 is used in the re-create variable of the OUT2 data set.

OUT2= does not suppress the printed output from the statement. To suppress the printed output, use the NOPRINT option.

See [“The OUT2= Data Set” on page 491](#) for a description of the variables in the OUT2= data set.

SHORT

prints only the list of variable names, the index information, and the sort information for the SAS data set.

Restriction If the list of variables is more than 32,767 characters, the list is truncated and a WARNING is written to the SAS log. To get a complete list of the variables, request an alphabetical listing of the variables.

VARNUM

prints a list of the variable names in the order of their logical position in the data set. By default, the CONTENTS statement lists the variables alphabetically. The physical position of the variable in the data set is engine-dependent.

Details

Using the CONTENTS Procedure Instead of the CONTENTS Statement

The only difference between the CONTENTS procedure and the CONTENTS statement in PROC DATASETS is the default for libref in the DATA= option. For PROC CONTENTS, the default is Work. For the CONTENTS statement, the default is the libref of the procedure input library.

Printing Variables

The CONTENTS statement prints an alphabetical listing of the variables by default, except for variables in the form of a numbered range list. Numbered range lists,

such as x1–x100, are printed in incrementing order, that is, x1–x100. For more information, see [“Alphabetic List of Variables and Attributes” on page 479](#).

Note: If a label is changed after a view is created from a data set with variable labels, the CONTENTS or DATASETS procedure output shows the original labels. The view must be recompiled in order for the CONTENTS or DATASETS procedure output to reflect the new variable labels.

Displaying the ICU Revision Number

The CONTENTS statement prints the International Components for Unicode (ICU) revision number when you use the SORT procedure for a linguistic sort on a data set. For more information about linguistic sorting, see [Chapter 55, “SORT Procedure,” on page 2039](#).

Library Contents and AES Encryption

When requesting the contents of all data files in a library, use the `_ALL_` option. If the library contains data files that are AES -encrypted, the `ENCRYPTKEY=` data set option must be used to access the data files. Here is an example using the `ENCRYPTKEY=` option:

```
proc contents data=MyLib._all_ (encryptkey=key-value);
run;
```

If the key value does not match the key value for a particular data file in the library, then you will be prompted to enter the correct key value.

For more information about AES encryption, see [“AES Encryption” in SAS Programmer’s Guide: Essentials](#). For more information about the `ENCRYPTKEY=` data set option, see [“ENCRYPTKEY= Data Set Option” in SAS Data Set Options: Reference](#).

Observation Length, Alignment, and Padding for a SAS Data Set

There are three different cases for alignment.

- Observations within a SAS data set are aligned on double-byte boundaries whenever possible. As a result, 8-byte and 4-byte numeric variables are positioned at 8-byte boundaries at the front of the data set. They are followed by character variables in the order in which they are encountered. If the data set contains only 4-byte numeric data, the alignment is based on 4-byte boundaries. Since numeric doubles can be operated upon directly rather than being moved and aligned before doing comparisons or increments, the boundaries cause better performance.

Since there are many observations contained within a given disk data page buffer, there might be padding between observations. The padding is to let each observation be aligned on a double-byte boundary. See the following example:

```
data a;
  length aa 7 bb 6 cc $10 dd 8 ee 3;
  aa = 1;
```



```

bb = 2;
cc = 'abc';
dd = 3;
ee = 4;
ff = 5;
output;
run;

proc contents data=a out=a1;
run;

proc print data=a1(keep=name length varnum npos);
run;

```

Figure 15.1 Observation Length

The CONTENTS Procedure			
Data Set Name	WORK.A	Observations	1
Member Type	DATA	Variables	6
Engine	V9	Indexes	0
Created	08/16/2016 12:08:25	Observation Length	48
Last Modified	08/16/2016 12:08:25	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	WINDOWS_64		
Encoding	wlatin1 Western (Windows)		

PROC CONTENTS shows an observation length of 48. PROC PRINT displays the internal layout of the variables within the observation where NPOS is the zero-based offset for each variable.

Figure 15.2 Observation and Variable Boundaries

Obs	NAME	LENGTH	VARNUM	NPOS
1	aa	7	1	16
2	bb	6	2	23
3	cc	10	3	32
4	dd	8	4	0
5	ee	3	5	29
6	ff	8	6	8

Variables DD and FF, the only true numeric doubles, are at offsets 0 and 8, respectively, so they are automatically aligned. The rest of the observation contains the remaining numeric variables and then character variables.

The last physical variable in this layout is CC with an offset of 32 and a length of 10. This gives you an internal length of 42, even though PROC CONTENTS reports the observation length as 48. The difference is the 6 bytes of padding so that the next observation is aligned on a double-byte boundary within the disk page buffer.

- No alignment is done when the observation does not contain 8-byte numeric variables as demonstrated in the next example, which gives you an observation length of 7 and no padding between observations within disk page buffers:

```
data b;
  length aa 6 cc $1;
  aa = 1;
  cc = 'x';
  output;
run;

proc contents data=b out=b1;
run;

proc print data=b1(keep=name length varnum npos);
run;
```

Figure 15.3 Variables and Attributes

Obs	NAME	LENGTH	VARNUM	NPOS
1	aa	6	1	0
2	cc	1	2	6

- Observations for compressed data sets are not aligned within the disk page buffer, but the same algorithm is used for positioning the variables within the

observations. Compressed observations must be uncompressed and moved into a work buffer. The 8-byte numeric values will be aligned and ready for use immediately after uncompressing. The observation length in the PROC CONTENTS output might be larger due to operating system-specific overhead.

COPY Statement

Copies all or some of the SAS files in a SAS library.

Restriction: The COPY statement does not support data set options.

Notes: When using the COPY statement with SAS/ACCESS LIBNAME engines, see [“Differences in the DATASETS Procedure Output When Using SAS/ACCESS LIBNAME Engines” on page 375](#).

When the DATA= input table is stored as a table or view in a data source, the COPY statement of PROC DATASETS can use in-database processing to perform most of its work within the data source. PROC DATASETS supports Greenplum, PostgreSQL, and Vertica. [“In-Database Processing for PROC COPY” on page 310](#). For CAS engine specifics, see [“CAS Processing for PROC COPY” on page 328](#).

Tip: The COPY statement defaults to the encoding and data representation of the output library when you use Remote Library Services (RLS) such as SAS/SHARE or SAS/CONNECT. If you are not using RLS, you must use the PROC COPY option NOCLONE for the output files to take on the encoding and data representation of the output library. Using the NOCLONE option results in a copy with the data representation of the data library (if specified in the OUTREP= LIBNAME option) or the native data representation of the operating environment.

Example: [“Example 2: Manipulating SAS Files” on page 497](#)

Syntax

```
COPY <ACCEL | NOACCEL>OUT=libref-1
<CLONE | NOCLONE>
<CONSTRAINT=YES | NO>
<DATECOPY>
<ENCRYPTKEY=key-value>
<FORCE>
<IN=libref-2>
<INDEX=YES | NO>
<MEMTYPE=(member-type(s))>
<MOVE <ALTER=alter-password>>
<OVERRIDE=(ds-option-1=value-1 <ds-option-2=value-2 ...> ) >;
```

Required Argument

OUT=libref-1

names the SAS library to copy SAS files to.

Aliases OUTLIB=

OUTDD=

Example [“Example 2: Manipulating SAS Files” on page 497](#)

Optional Arguments

ACCEL

specifies to perform the copy operation in CAS. Both the IN= and OUT= libraries must be CAS engine libraries and they must use the same CAS session.

Default ACCEL

See [“NOACCEL” on page 414](#)

[“Copying a CAS Table to Another CAS Table” on page 422](#)

NOACCEL

specifies to perform the copy operation in the SAS Compute Server. Both the IN= and OUT= libraries must be CAS engine libraries and they must use the same CAS session. specifies to perform the copy operation in the SAS Compute Server. Both the IN= and OUT= libraries must be CAS engine libraries and they must use the same CAS session.

See [“ACCEL” on page 414](#)

[“Copying a CAS Table to Another CAS Table” on page 422](#)

CLONE

specifies whether to copy the following data set attributes:

- size of input/output buffers
- whether the data set is compressed
- whether free space is reused
- data representation of input data set, library, or operating environment
- encoding value
- whether a compressed data set can be randomly accessed by an observation number

These attributes are specified with data set options, SAS system options, and LIBNAME statement options:

- BUFSIZE= value for the size of the input/output buffers
- COMPRESS= value for whether the data set is compressed
- REUSE= value for whether free space is reused
- OUTREP= value for data representation
- ENCODING= or INENCODING= for encoding value

- POINTOBS= value for whether a compressed data set can be randomly accessed by an observation number

For the BUFSIZE= attribute, the following table summarizes how the COPY statement works:

Table 15.5 CLONE and the Buffer Page Size Attribute

Option	COPY Statement
CLONE	Uses the BUFSIZE= value from the input data set for the output data set. However, specifying BUFSIZE= value in the OVERRIDE= option list results in a copy that uses the specified value.
NOCLONE	Uses the current setting of the SAS system option BUFSIZE= for the output data set.
Neither	Determines the type of access method, sequential or random, used by the engine for the input data set and the engine for the output data set. If both engines use the same type of access, the COPY statement uses the BUFSIZE= value from the input data set for the output data set. If the engines do not use the same type of access, the COPY statement uses the setting of SAS system option BUFSIZE= for the output data set.

For the COMPRESS= attribute, the following table summarizes how the COPY statement works:

Table 15.6 CLONE and the Compression Attribute

Option	COPY Statement
CLONE	Uses the values from the input data set for the output data set. However, specifying COMPRESS= value in the OVERRIDE= option list results in a copy that uses the specified encoding.
NOCLONE	Results in a copy with the compression of the operating environment or, if specified, the value of the COMPRESS= option in the LIBNAME statement for the library.
Neither	Defaults to CLONE.

For the REUSE= attribute, the following table summarizes how the COPY statement works:

Table 15.7 *CLONE and the Reuse Space Attribute*

Option	COPY Statement
CLONE	Uses the values from the input data set for the output data set. If the engine for the input data set does not support the reuse space attribute, then the COPY statement uses the current setting of the corresponding SAS system option. However, specifying REUSE= value in the OVERRIDE= option list results in a copy that uses the specified value.
NOCLONE	Uses the current setting of the SAS system options COMPRESS= and REUSE= for the output data set.
Neither	Defaults to CLONE.

For the OUTREP= attribute, the following table summarizes how the COPY statement works:

Table 15.8 *CLONE and the Data Representation Attribute*

Option	COPY Statement
CLONE	Results in a copy with the same data representation of the input data set. However, if you specify an OUTREP= value in the OVERRIDE= option list, the data representation is changed, and its encoding will change to an encoding that is compatible with the data representation and the locale of the current session. The encoding will also change if it is specified in the OVERRIDE= option list.
NOCLONE	Results in a copy with the data representation of the operating environment or, if specified, the value of the OUTREP= option in the LIBNAME statement for the OUT= library.
Neither	Defaults to CLONE.

Data representation is the form in which data is stored in a particular operating environment. Different operating environments use the following different standards or conventions:

- for storing floating-point numbers
- for character encoding (ASCII or EBCDIC)
- for the ordering of bytes in memory (big Endian or little Endian)
- for word alignment (4-byte boundaries or 8-byte boundaries)
- for data-type length (16-bit, 32-bit, or 64-bit)

Native data representation is when the data representation of a file is the same as the CPU operating environment.

For the ENCODING= attribute, the following table summarizes how the COPY statement works.

Table 15.9 CLONE and the Encoding Attribute

Option	COPY Statement
CLONE	Results in a copy that uses the encoding of the input data set or, if specified, the value of the INENCODING= option in the LIBNAME statement for the input library. However, specifying ENCODING= value in the OVERRIDE= option list results in a copy that uses the specified encoding.
NOCLONE	Results in a copy that uses the encoding of the current session encoding or, if specified, the value of the OUTENCODING= option in the LIBNAME statement for the output library.
Neither	Defaults to CLONE.

All text (character) data that is stored, transmitted, or processed by a computer is in an encoding. An encoding maps each character to a unique numeric representation. An encoding is a combination of a character set with an encoding method. A character set is the repertoire of characters and symbols that are used by a language or group of languages. An encoding method is the set of rules that are used to assign the numbers to the set of characters that are used in an encoding.

For the POINTOBS= attribute, the following table summarizes how the COPY statement works. To use POINTOBS=, the output data set must be compressed.

Table 15.10 CLONE and the POINTOBS= Attribute

Option	COPY Statement
CLONE	Uses the POINTOBS= value from the input data set for the output data set. However, specifying POINTOBS= value in the OVERRIDE= option list results in a copy that uses the specified value.
NOCLONE	Uses the LIBNAME statement if the output data set is compressed and the POINTOBS= option is specified and supported by the output engine. If the LIBNAME statement is not specified and the data set is compressed, the default is POINTOBS=YES when supported by the output engine.
Neither	Defaults to CLONE.

CONSTRAINT=YES | NO

specifies whether to copy all integrity constraints when copying a data set.

Default NO

Tip For data sets with integrity constraints that have a foreign key, the COPY statement copies the general and referential constraints if CONSTRAINT=YES is specified and the entire library is copied. If you use the SELECT or EXCLUDE statement to copy the data sets, then the referential integrity constraints are not copied.

DATECOPY

copies the SAS internal date and time at which the SAS file was created and when it was last modified to the resulting copy of the file. Note that the operating environment date and time are not preserved.

Restriction DATECOPY cannot be used with encrypted files or catalogs.

Tips You can alter the file creation date and time with the DTC= option in the MODIFY statement. See [MODIFY statement on page 446](#).

If the file that you are copying has attributes that require additional processing, the last modified date is changed to the current date. For example, when you copy a data set that has an index, the index must be rebuilt, and the last modified date changes to the current date. Other attributes that require additional processing and that could affect the last modified date include integrity constraints and a sort indicator.

ENCRYPTKEY= *key-value*

specifies the key value needed to copy data sets in the IN= library that have AES encryption.

Note If the output library does not support AES-encryption and the input data set is AES-encrypted, the COPY process will produce an error.

See [“Copying AES-Encrypted Data Files Containing Referential Integrity Constraints” on page 426](#)

FORCE

enables you to use the MOVE option for a SAS data set on which an audit trail exists.

Note The AUDIT file is not moved with the audited data set.

IN=libref-2

names the SAS library containing SAS files to copy.

Aliases INLIB=

INDD=

Default the libref of the procedure input library

Interaction To copy only selected members, use the SELECT or EXCLUDE statements.

INDEX=YES | NO

specifies whether to copy all indexes for a data set when copying the data set to another SAS library.

Default YES

MEMTYPE=(*member-type(s)*)

restricts processing to one or more member types.

Aliases MTYPE=

MT=

Default If you omit MEMTYPE= in the PROC DATASETS statement, the default is MEMTYPE=ALL.

Note When PROC COPY processes a SAS library on tape and the MEMTYPE= option is not specified, it scans the entire sequential library for entries until it reaches the end-of-file. If the sequential library is a multivolume tape, all tape volumes are mounted. This behavior is also true for single-volume tape libraries.

See [“Specifying Member Types When Copying or Moving SAS Files” on page 423](#) and [“Member Types” on page 470](#)

Example [“Example 2: Manipulating SAS Files” on page 497](#)

MOVE

moves SAS files from the input data library (named with the IN= option) to the output data library (named with the OUT= option). And deletes the original files from the input data library.

Restriction The MOVE option can be used to delete a member of a SAS library only if the IN= engine supports the deletion of tables. A tape format engine does not support table deletion. If you use a tape format engine, SAS suppresses the MOVE operation and prints a warning.

See [“Using Passwords with the DATASETS Procedure” on page 467](#)

Example [“Example 2: Manipulating SAS Files” on page 497](#)

ALTER=*alter-password*

provides the Alter password for any alter-protected SAS files that you are moving from one data library to another. Because the MOVE option deletes the SAS file from the original data library, you need Alter access to move the SAS file.

OVERRIDE=(*ds-option-1=value-1 <ds-option-2=value-2> ...*)

overrides specified output data set options copied from the input data set. Some data set options might not be appropriate in the output data set context of COPY.

- Restriction** The **OVERWRITE** option is ignored if the **NOCLONE** option is specified. However, it can be used to modify data set attributes other than those controlled by the **NOCLONE** option.
- Interaction** When you specify an **OUTREP=** value in the **OVERWRITE=** option, the default encoding is based on the operating environment that is represented by the **OUTREP=** value and the locale of the current Compute Server session. To assign a nondefault encoding such as UTF-8, you must also specify an **ENCODING=** value in the **OVERWRITE=** option. For more information about locale and encoding, see *SAS National Language Support (NLS): Reference Guide*.
- Tip** The default (or **CLONE**) behavior is to preserve many attributes, including the data representation and encoding, from the input data set. If instead you want the data representation and encoding of the Compute Server session that is executing the procedure, then specify **OVERWRITE=(OUTREP=SESSION ENCODING=SESSION)** in **PROC COPY** or in the **COPY** statement. If you do not want to preserve any attributes from the input data set, then specify **NOCLONE** instead.

NOCLONE

See the description of [“CLONE”](#) on page 414.

Details

Performance Improvement Using SELECT with MEMTYPE

When using the **COPY** statement, an in-memory directory of the library is obtained. This can be a performance issue if the library has thousands of members and only a few members are being copied. To resolve this performance issue, use a combination of the **MEMTYPE=** option in the **COPY** statement with a **SELECT** statement. Here is an example of this process:

```
proc datasets lib=work;
  copy out=mylib memtype=(data catalog);
  select mydata x1-x10 data2;
run;
```

Note: If either **MEMTYPE=ALL** or a wildcard specification (":") is used, the performance code cannot be used.

Using the Block I/O Method to Copy

The block I/O method is used to copy blocks of data instead of one observation at a time. This method can increase performance when you are copying large data sets. SAS determines whether to use this method. Not all data sets can use the block I/O method. There are restrictions set by the **COPY** statement and the Base SAS engine.

To display information in the SAS log about the copy method that is being used, you can specify the MSGLEVEL= system option as follows:

```
options msglevel=i;
```

The following message is written to the SAS log, if the block I/O method is not used:

INFO: Data set block I/O cannot be used because:

If the COPY statement determines that the block I/O will not be used, one of the following explanations is written to the SAS log:

INFO: - The data sets use different engines, have different variables or have attributes that might differ.

INFO: - There is no member level locking.

INFO: - The OBS option is active.

INFO: - The FIRSTOBS option is active.

If the Base SAS engine determines that the block I/O method will not be used, one of the following explanations is written to the SAS log:

INFO: - Referential Integrity Constraints exist.

INFO: - Cross Environment Data Access is being used.

INFO: - The file is compressed.

INFO: - The file has an audit file which is not suspended.

If you are having performance issues and want to create a subset of a large data set for testing, you can use the OBS=0 option. In this case, you want to reduce the use of system resources by disabling the block I/O method.

The following example uses the OBS=0 option to reduce the use of system resources:

```
options obs=0 msglevel=i;
proc copy in=old out=lib;
select a;
run;
```

You get the same results when you use the SET statement:

```
data lib.new;
  if 0 then set old.a;
  stop;
run;
```

Copying an Entire Library

To copy an entire SAS library, simply specify an input data library and an output data library following the COPY statement. For example, the following statements copy all the SAS files in the Source data library into the Dest data library:

```
proc datasets library=source;
  copy out=dest;
run;
```

Copying Selected SAS Files

To copy selected SAS files, use a SELECT or EXCLUDE statement. For more discussion of using the COPY statement with a SELECT statement or an EXCLUDE statement, see [“Specifying Member Types When Copying or Moving SAS Files” on page 423](#) and see [Manipulating SAS Files on page 497](#) for an example. Also, see [EXCLUDE statement on page 433](#) and [SELECT statement on page 458](#).

You can also select or exclude an abbreviated list of members. For example, the following statement selects members Tabs, Test1, Test2, and Test3:

```
select tabs test1-test3;
```

Also, you can select a group of members whose names begin with the same letter or letters by entering the common letters followed by a colon (:). For example, you can select the four members in the previous example and all other members having names that begin with the letter T by specifying the following statement:

```
select t::;
```

You specify members to exclude in the same way that you specify those to select. That is, you can list individual member names, use an abbreviated list, or specify a common letter or letters followed by a colon (:). For example, the following statement excludes the members Stats, Teams1, Teams2, Teams3, Teams4 and all the members that begin with the letters RBI from the copy operation:

```
exclude stats teams1-teams4 rbi::;
```

Note that the MEMTYPE= option affects which types of members are available to be selected or excluded.

When a SELECT or EXCLUDE statement is used with CONSTRAINT=YES, only the general integrity constraints on the data sets are copied. Any referential integrity constraints are not copied.

Copying a CAS Table to Another CAS Table

When PROC COPY IN= and OUT= options are set to a CAS libref, and the same CAS session is used by both librefs by default the copy occurs in CAS.

This is a change to the default behavior. If options MSGLEVEL=I system option is set, when the COPY occurs in CAS, an INFO message is sent to the log: INFO: Running COPY in Cloud Analytic Services.

The PROC COPY option ACCEL | NOACCEL determines where the COPY procedure executes. ACCEL is the default, which means the COPY operation runs on the CAS server. If the COPY procedure fails on the CAS server, it does not attempt to copy the file.

If the OVERRIDE option is specified on the PROC COPY invocation, the option is ignored, and a note is sent to the log.

You cannot copy between sessions with PROC COPY. If the CAS libraries are defined with two different sessions, the following error is displayed: ERROR: The CAS sessions SESS1 and SESS2 used for this action do not match.

Specifying Member Types When Copying or Moving SAS Files

The MEMTYPE= option in the COPY statement differs from the MEMTYPE= option in other statements in the procedure in several ways:

- A slash does not precede the option.
- You cannot limit its effect to the member immediately preceding it by enclosing the MEMTYPE= option in parentheses.
- The SELECT and EXCLUDE statements and the IN= option (in the COPY statement) affect the behavior of the MEMTYPE= option in the COPY statement according to the following rules:

- 1 MEMTYPE= in a SELECT or EXCLUDE statement takes precedence over the MEMTYPE= option in the COPY statement. The following statements copy only Vision.catalog and Nutr.data from the default data library to the Dest data library; the MEMTYPE= value in the first SELECT statement overrides the MEMTYPE= value in the COPY statement.

```
proc datasets;
  copy out=dest memtype=data;
    select vision(memtype=catalog) nutr;
run;
```

- 2 If you do not use the IN= option, or you use it to specify the library that happens to be the procedure input library, the value of the MEMTYPE= option in the PROC DATASETS statement limits the types of SAS files that are available for processing. The procedure uses the order of precedence described in rule 1 to further subset the types available for copying. The following statements do not copy any members from the default data library to the Dest data library. Instead, the procedure issues an error message because the MEMTYPE= value specified in the SELECT statement is not one of the values of the MEMTYPE= option in the PROC DATASETS statement.

```
/* This step fails! */
proc datasets memtype=(data program);
  copy out=dest;
    select apples / memtype=catalog;
run;
```

- 3 If you specify an input data library in the IN= option other than the procedure input library, the MEMTYPE= option in the PROC DATASETS statement has no effect on the copy operation. Because no subsetting has yet occurred, the procedure uses the order of precedence described in rule 1 to subset the types available for copying. The following statements successfully copy Bodyfat.data to the Dest data library because the Source library specified in the IN= option in the COPY statement is not affected by the MEMTYPE= option in the PROC DATASETS statement.

```
proc datasets library=work memtype=catalog;
  copy in=source out=dest;
    select bodyfat / memtype=data;
run;
```

When using the COPY statement, an in-memory directory of the library is obtained. This can be a performance issue if the library has thousands of members and only a few members are being copied. To resolve this performance issue, use a

combination of the MEMTYPE= option in the COPY statement with a SELECT statement. Here is an example of this process:

```
proc datasets lib=work;
    copy out=mylib memtype=(data catalog);
    select mydata x1-x10 data2;
run;
```

Note: If either MEMTYPE=ALL or a wildcard specification (":") is used, the performance code cannot be used.

Copying Views

The COPY statement with NOCLONE specified supports the OUTREP= and ENCODING= LIBNAME options for SQL views, DATA step views, and some views (Oracle and Sybase). When you use the COPY statement with Remote Library Services (RLS) such as SAS/SHARE or SAS/CONNECT, the COPY statement defaults to the encoding and data representation of the output library.

CAUTION

If you use the DATA statement's SOURCE=NOSAVE option when creating a DATA step view, the view cannot be copied from one version of SAS to another version.

Copying Password-Protected SAS Files

You can copy a password-protected SAS file without specifying the password. In addition, because the password continues to correspond to the SAS file, you must know the password in order to access and manipulate the SAS file after you copy it.

Copying Data Sets with Long Variable Names

If the VALIDVARNAME=V6 system option is set and the data set has long variable names, the long variable names are truncated, unique variables names are generated, and the copy succeeds. The same is true for index names. If VALIDVARNAME=ANY, the copy fails with an error if the OUT= engine does not support long variable names.

When a variable name is truncated, the variable name is shortened to eight bytes. If this name has already been defined in the data set, the name is shortened and a digit is added, starting with the number 2. The process of truncation and adding a digit continues until the variable name is unique. For example, a variable named LONGVARNAME becomes LONGVARN, provided that a variable with that name does not already exist in the data set. In that case, the variable name becomes LONGVAR2.

CAUTION

Truncated variable names can collide with names already defined in the input data set. This behavior is possible when the variable name that is already defined is exactly eight bytes long and ends in a digit. In the following example, the truncated name is defined in the output data set and the name from the input data set is changed:

```

options validvarname=any;
data test;
    longvar10='aLongVariableName';
    retain longvar1-longvar5 0;
run;

options validvarname=v6;
proc copy in=work out=sasuser;
    select test;
run;

```

In this example, LONGVAR10 is truncated to LONGVAR1 and placed in the output data set. Next, the original LONGVAR1 is copied. Its name is no longer unique. Therefore, it is renamed LONGVAR2. The other variables in the input data set are also renamed according to the renaming algorithm. The following example is from the SAS log:

```

1  options validvarname=any;
2  data test;
3      longvar10='aLongVariableName';
4      retain longvar1-longvar5 0;
5  run;

```

NOTE: The data set WORK.TEST has 1 observations and 6 variables.

NOTE: DATA statement used (Total process time):

real time	2.60 seconds
cpu time	0.07 seconds

```

6
7  options validvarname=v6;
8  proc copy in=work out=sasuser;
9      select test;
10 run;

```

NOTE: Copying WORK.TEST to SASUSER.TEST (memtype=DATA).

NOTE: The variable name longvar10 has been truncated to longvar1.

NOTE: The variable longvar1 now has a label set to longvar10.

NOTE: Variable LONGVAR1 already exists on file SASUSER.TEST, using LONGVAR2 instead.

NOTE: The variable LONGVAR2 now has a label set to LONGVAR1.

NOTE: Variable LONGVAR2 already exists on file SASUSER.TEST, using LONGVAR3 instead.

NOTE: The variable LONGVAR3 now has a label set to LONGVAR2.

NOTE: Variable LONGVAR3 already exists on file SASUSER.TEST, using LONGVAR4 instead.

NOTE: The variable LONGVAR4 now has a label set to LONGVAR3.

NOTE: Variable LONGVAR4 already exists on file SASUSER.TEST, using LONGVAR5 instead.

NOTE: The variable LONGVAR5 now has a label set to LONGVAR4.

NOTE: Variable LONGVAR5 already exists on file SASUSER.TEST, using LONGVAR6 instead.

NOTE: The variable LONGVAR6 now has a label set to LONGVAR5.

NOTE: There were 1 observations read from the data set WORK.TEST.

NOTE: The data set SASUSER.TEST has 1 observations and 6 variables.

NOTE: PROCEDURE COPY used (Total process time):

real time	13.18 seconds
cpu time	0.31 seconds

```

11
12  proc print data=test;
13  run;

ERROR: The value LONGVAR10 is not a valid SAS name.
NOTE: The SAS System stopped processing this step because of errors.
NOTE: PROCEDURE PRINT used (Total process time):
      real time          0.15 seconds
      cpu time           0.01 seconds

```

Copying AES-Encrypted Data Files

In order to copy data files with AES encryption, you must specify the `ENCRYPTKEY=key-value` in one of two ways:

- if all the AES-encrypted data sets in the library have the same `ENCRYPTKEY=key-value`, use the following example:

```

proc copy in=lib1 out=lib2 encryptkey=key-value;
run;

```

- if the `ENCRYPTKEY=key-value` is different for each data set, use the `SELECT` statement as shown in the following example:

```

proc copy in=lib1 out=lib2;
  select mydata1 (encryptkey=key-value1) mydata2 (encryptkey=key-value2);
run;

```

For more information about AES encryption, see [“AES Encryption” in SAS Programmer’s Guide: Essentials](#). For more information about the `ENCRYPTKEY=` data set option, see [“ENCRYPTKEY= Data Set Option” in SAS Data Set Options: Reference](#).

Copying AES-Encrypted Data Files Containing Referential Integrity Constraints

In order to copy data files with AES encryption and with referential integrity constraints, you must do the following:

- copy the entire contents of the `IN=library`
- specify the `CONSTRAINT=YES` statement option
- specify the `ENCRYPTKEY=key-value` option or the library must be metadata bound with the encryption key recorded for the library

```

proc copy in=Lib1 out=Lib2 constraint=yes encryptkey=key-value;
run;

```

However, the above code will not work if there are multiple data primary key and referential data set pairs in the `IN=LIB` and each pair has different `ENCRYPTKEY=` values. For example, if there are two pairs:


```
primarydset1/foreigndset1 having ENCRYPTKEY=secret1
primarydset2/foreigndset2 having ENCRYPTKEY=secret2
```

then the above scheme would work for only one pair. All pairs must have the same key value.

For more information, see [“ENCRYPTKEY= Data Set Option” in SAS Data Set Options: Reference](#).

Copying Generation Groups

You can use the COPY statement to copy an entire generation group. However, you cannot copy a specific version in a generation group.

Transporting SAS Data Sets between Hosts

You use the COPY procedure, along with the XPORT engine or a REMOTE engine, to transport SAS data sets between hosts. For more information, see “Strategies for Moving and Accessing SAS Files” in *Moving and Accessing SAS Files*.

Using the COPY Procedure Instead of the COPY Statement

Generally, the COPY procedure functions the same as the COPY statement in the DATASETS procedure. Here is a list of differences:

- The IN= argument is required with PROC COPY. In the COPY statement, IN= is optional. If omitted, the default value is the libref of the procedure input library.
- PROC DATASETS cannot work with libraries that allow only sequential data access.
- The COPY statement honors the NOWARN option but PROC COPY does not.

DELETE Statement

Deletes SAS files from a SAS library.

Example: [“Example 2: Manipulating SAS Files” on page 497](#)

Syntax

```
DELETE SAS-file(s)
</ <ALTER=alter-password>
<ENCRYPTKEY=key-value>
<GENNUM=ALL | HIST | REVERT | integer>
<MEMTYPE=member-type>>;
```

Required Argument

SAS-file(s)

specifies one or more SAS files that you want to delete. If the SAS file has an ALTER= password assigned, it must be specified in order to delete the SAS file. You can also use a numbered range list or colon list. For more information, see [“Data Set Name Lists” in SAS Programmer’s Guide: Essentials](#).

Optional Arguments

ALTER=alter-password

provides the Alter password for any alter-protected SAS files that you want to delete. You can use the option either in parentheses after the name of each SAS file or after a forward slash.

See [“Using Passwords with the DATASETS Procedure” on page 467](#)

ENCRYPTKEY=key-value

specifies the key value for AES encryption. This option is required when specifying GENNUM=REVERT (which is the same as GENNUM=0) or GENNUM=relative-generation-number. The key value for the ENCRYPTKEY= option must be the key value for the base version. For more information, see [“Library Contents and AES Encryption” on page 410](#).

GENNUM=ALL | HIST | REVERT | integer

restricts processing for generation data sets. You can use the option either in parentheses after the name of each SAS file or after a forward slash. Here is a list of valid values:

ALL	refers to the base version and all historical versions in a generation group.
HIST	refers to all historical versions, but excludes the base version in a generation group.
REVERT 0	deletes the base version and changes the most current historical version, if it exists, to the base version.
integer	is a number that references a specific version from a generation group. Specifying a positive number is an absolute reference to a specific generation number that is appended to a data set’s name (that is, gennum=2 specifies MYDATA#002). Specifying a negative number is a relative reference to a historical version in relation to the base version, from the youngest to the oldest (that is, gennum=-1 refers to the youngest historical version).

See [“Restricting Processing for Generation Data Sets” on page 468](#)

MEMTYPE=member-type

restricts processing to one member type. You can use the option either in parentheses after the name of each SAS file or after a forward slash.

Aliases MTYPE=

MT=

Default DATA

See [“Restricting Member Types for Processing” on page 469](#)

Example [“Example 2: Manipulating SAS Files” on page 497](#)

Details

The Basics

SAS immediately deletes SAS files when the RUN group executes. You do not have an opportunity to verify the Delete operation before it begins.

If the SAS file has an ALTER= password assigned, it must be specified in order to delete the SAS file.

If you attempt to delete a SAS file that does not exist in the procedure input library, PROC DATASETS issues a message and continues processing. If NOWARN is used, no message is issued.

When you use the DELETE statement to delete a data set that has indexes associated with it, the statement also deletes the indexes.

You cannot use the DELETE statement to delete a data file that has a foreign key integrity constraint or a primary key with foreign key references. For data files that have foreign keys, you must remove the foreign keys before you delete the data file. For data files that have primary keys with foreign key references, you must remove the foreign keys that reference the primary key before you delete the data file.

If you attempt to delete a CAS table that does not exist in the input CAS engine libref, a message is written to the log and processing continues. If NOWARN is used, no message is issued.

Working with Generation Groups

Overview of Working with Generation Groups

When you are working with generation groups, you can use the DELETE statement to delete the following versions:

- delete the base version and all historical versions
- delete the base version and rename the youngest historical version to the base version
- delete an absolute version
- delete a relative version
- delete all historical versions and leave the base version

Deleting the Base Version and All Historical Versions

The following statements delete the base version and all historical versions where the data set name is A:

```
proc datasets;
    delete A(gennum=all);

proc datasets;
    delete A / gennum=all;

proc datasets gennum=all;
    delete A;
```

The following statements delete the base version and all historical versions where the data set name begins with the letter A:

```
proc datasets;
    delete A:(gennum=all);

proc datasets;
    delete A: / gennum=all;

proc datasets gennum=all;
    delete A::;
```

Deleting the Base Version and Renaming the Youngest Historical Version to the Base Version

The following statements delete the base version and rename the youngest historical version to the base version, where the data set name is A:

```
proc datasets;
    delete A(gennum=revert);

proc datasets;
    delete A / gennum=revert;

proc datasets gennum=revert;
    delete A;
```

The following statements delete the base version and rename the youngest historical version to the base version, where the data set name begins with the letter A:

```
proc datasets;
    delete A:(gennum=revert);

proc datasets;
    delete A: / gennum=revert;

proc datasets gennum=revert;
    delete A::;
```

Deleting a Version with an Absolute Number

The following statements use an absolute number to delete the first historical version:

```
proc datasets;
    delete A(gennum=1);

proc datasets;
    delete A / gennum=1;

proc datasets gennum=1;
    delete A;
```

The following statements delete a specific historical version, where the data set name begins with the letter A:

```
proc datasets;
    delete A:(gennum=1);

proc datasets;
    delete A: / gennum=1;

proc datasets gennum=1;
    delete A::;
```

Deleting a Version with a Relative Number

The following statements use a relative number to delete the youngest historical version, where the data set name is A:

```
proc datasets;
    delete A(gennum=-1);

proc datasets;
    delete A / gennum=-1;

proc datasets gennum=-1;
    delete A;
```

The following statements use a relative number to delete the youngest historical version, where the data set name begins with the letter A:

```
proc datasets;
    delete A:(gennum=-1);

proc datasets;
    delete A: / gennum=-1;

proc datasets gennum=-1;
    delete A::;
```

Deleting All Historical Versions and Leaving the Base Version

The following statements delete all historical versions and leave the base version, where the data set name is A:

```
proc datasets;
```

```

delete A(gennum=hist);

proc datasets;
  delete A / gennum=hist;

proc datasets gennum=hist;
  delete A;

```

The following statements delete all historical versions and leave the base version, where the data set name begins with the letter A:

```

proc datasets;
  delete A:(gennum=hist);

proc datasets;
  delete A: / gennum=hist;

proc datasets gennum=hist;
  delete A;;

```

EXCHANGE Statement

Exchanges the names of two SAS files in a SAS library.

Example: [“Example 2: Manipulating SAS Files” on page 497](#)

Syntax

EXCHANGE *name-1=other-name-1* <*name-2=other-name-2* ...>
</ <**ALTER**=*alter-password*> <**MEMTYPE**=*member-type*>>;

Required Argument

name=other-name

exchanges the names of SAS files in the procedure input library. Both *name* and *other-name* must already exist in the procedure input library.

Optional Arguments

ALTER=alter-password

provides the Alter password for any alter-protected SAS files whose names you want to exchange. You can use the option either in parentheses after the name of each SAS file or after a forward slash.

See [“Using Passwords with the DATASETS Procedure” on page 467](#)

MEMTYPE=member-type

restricts processing to one member type. You can exchange only the names of SAS files of the same type. You can use the option either in parentheses after the name of each SAS file or after a forward slash.

Default If you do not specify MEMTYPE= in the PROC DATASETS statement, the default is ALL.

See [“Restricting Member Types for Processing” on page 469](#)

Details

When you exchange more than one pair of names in one EXCHANGE statement, PROC DATASETS performs the exchanges in the order in which the names of the SAS files occur in the directory listing, not in the order in which you list the exchanges in the EXCHANGE statement.

If the *name* SAS file does not exist in the SAS library, PROC DATASETS stops processing the RUN group that contains the EXCHANGE statement and issues an error message. To override this behavior, specify the NOWARN option in the PROC DATASETS statement.

The EXCHANGE statement also exchanges the associated indexes so that they correspond with the new name.

The EXCHANGE statement allows only two existing generation groups to exchange names. You cannot exchange a specific generation number with either an existing base version or another generation number.

EXCLUDE Statement

Excludes SAS files from copying.

Restrictions: The EXCLUDE statement must follow a COPY statement
The EXCLUDE statement cannot appear in the same COPY step with a SELECT statement

Example: [“Example 2: Manipulating SAS Files” on page 497](#)

Syntax

```
EXCLUDE SAS-file(s) </ MEMTYPE=member-type>;
```

Required Argument

SAS-file(s)

specifies one or more SAS files to exclude from the copy operation. All SAS files you name in the EXCLUDE statement must be in the library that is specified in the IN= option in the COPY statement. If the SAS files are generation groups, the EXCLUDE statement allows only selection of the base versions.

You can use the following shortcuts to list several SAS files in the EXCLUDE statement:

Table 15.11 Using the EXCLUDE Statement

Notation	Meaning
x1–xn	Specifies files X1 through Xn. The numbers must be consecutive.
x:	Specifies all files that begin with the letter X.

Optional Argument

MEMTYPE=*member-type*

restricts processing to one member type. You can use the option either in parentheses after the name of each SAS file or after a forward slash.

Aliases MTYPE=

MT=

Default If you do not specify MEMTYPE= in the PROC DATASETS statement, the COPY statement, or in the EXCLUDE statement, the default is MEMTYPE=ALL.

See [“Specifying Member Types When Copying or Moving SAS Files” on page 423](#) and [“Restricting Member Types for Processing” on page 469](#)

Details

Excluding Several Like-Named Files

You can use shortcuts for listing several SAS files in the EXCLUDE statement.

FORMAT Statement

Assigns, changes, and removes variable formats in the SAS data set specified in the MODIFY statement permanently.

Restriction: The FORMAT statement must appear in a MODIFY RUN group

Example: [“Example 4: Modifying SAS Data Sets” on page 505](#)

Syntax

FORMAT *variable-1* <*format-1*>
<*variable-2* <*format-2*> ...>;

Required Argument

variable

specifies one or more variables whose format you want to assign, change, or remove. If you want to disassociate a format with a variable, list the variable last in the list with no format following:

```
format x1-x3 4.1 time hhmm2.2 age;
```

Optional Argument

format

specifies a format to apply to the variable or variables listed before it. If you do not specify a format, the FORMAT statement removes any format associated with the variables in *variable-list*.

Tip To remove all formats from a data set, use the [ATTRIB Statement on page 400](#) and the `_ALL_` keyword.

IC CREATE Statement

Creates an integrity constraint.

Restriction: The IC CREATE statement must appear in a MODIFY RUN group

Note: In order for a referential constraint to be established, the foreign key must specify the same number of variables as the primary key, in the same order, and the variables must be of the same type (character or numeric) and length.

Syntax

IC CREATE <*constraint-name*=> *constraint* <MESSAGE='message-string'
<MSGTYPE=USER>>;

Required Argument

constraint

is the type of constraint. Here is a list of valid values:

NOT NULL (<i>variable</i>)	specifies that <i>variable</i> does not contain a SAS missing value, including special missing values.
---------------------------------	--

UNIQUE (<i>variables</i>)	specifies that the values of <i>variables</i> must be unique. This constraint is identical to DISTINCT.
DISTINCT (<i>variables</i>)	specifies that the values of <i>variables</i> must be unique. This constraint is identical to UNIQUE.
CHECK (WHERE- <i>expression</i>)	limits the data values of variables to a specific set, range, or list of values. This behavior is accomplished with a WHERE expression.
PRIMARY KEY (<i>variables</i>)	specifies a primary key, that is, a set of variables that do not contain missing values and whose values are unique.

When defining overlapping primary key and foreign key constraints, which means that variables in a data file are part of both a primary key and a foreign key definition, if you use exactly the same variables, then the variables must be defined in a different order.

A primary key affects the values of an individual data file until it has a foreign key referencing it.

Note: If a not null constraint exists for a variable that is being used to define a new primary key constraint, then the primary key constraint replaces the existing not null constraint.

FOREIGN KEY (<i>variables</i>) REFERENCES <i>table-</i> <i>name</i> <ON DELETE <i>referential-action</i> > <ON UPDATE <i>referential-action</i> >	specifies a foreign key, that is, a set of variables whose values are linked to the values of the primary key variables in another data file. The referential actions are enforced when updates are made to the values of a primary key variable that is referenced by a foreign key. There are three types of referential actions: RESTRICT, SET NULL, and CASCADE:
---	--

The following operations can be done with the RESTRICT referential action:

a delete operation	deletes the primary key row, but only if no foreign key values match the deleted value.
an update operation	updates the primary key value, but only if no foreign key values match the current value to be updated.

The following operations can be done with the SET NULL referential action:

a delete operation	deletes the primary key row and sets the corresponding foreign key values to NULL.
an update operation	modifies the primary key value and sets all matching foreign key values to NULL.

The following operations can be done with the CASCADE referential action:

an update operation	modifies the primary key value, and also modifies any matching foreign key values to the same value. CASCADE is not supported for Delete operations.
---------------------	--

Default	RESTRICT is the default action if no referential action is specified.
Requirements	When defining overlapping primary key and foreign key constraints, which means that variables in a data file are part of both a primary key and a foreign key definition, the following actions are required: If you use exactly the same variables, then the variables must be defined in a different order. The foreign key's update and delete referential actions must both be RESTRICT.
Interaction	Before it enforces a SET NULL or CASCADE referential action, SAS checks to see whether there are other foreign keys that reference the primary key and that specify RESTRICT for the intended operation. If RESTRICT is specified, or if the constraint reverts to the default values, then RESTRICT is enforced for all foreign keys, unless no foreign key values match the values to updated or deleted.

Optional Arguments

constraint-name

is an optional name for the constraint. The name must be a valid SAS name. When you do not supply a constraint name, a default name is generated. This default constraint name has the following form:

Table 15.12 Using *RESTRICT=*

Default Name	Constraint Type
NMxxxx	Not Null
UNxxxx	Unique
CKxxxx	Check
PKxxxx	Primary key
FKxxxx	Foreign key

xxxx is a counter beginning at 0001.

Note: The names PRIMARY, FOREIGN, MESSAGE, UNIQUE, DISTINCT, CHECK, and NOT cannot be used as values for *constraint-name*.

MESSAGE='message-string'

message-string is the text of an error message to be written to the log when the data fails the constraint:

```
ic create not null(socsec)
    message='Invalid Social Security number';
```

Length The maximum length of the message is 250 characters.

Example The following examples show how to Create integrity constraints:

```
ic create a = not null(x);
ic create Unique_D = unique(d);
ic create Distinct_DE = distinct(d e);
ic create E_less_D = check(where=(e < d or d = 99));
ic create primkey = primary key(a b);
ic create forkey = foreign key (a b) references table-name
    on update cascade on delete set null;
ic create not null (x);
```

MSGTYPE=USER

controls the format of the integrity constraint error message. By default when the MESSAGE= option is specified, the message that you define is inserted into the SAS error message for the constraint, separated by a space. MSGTYPE=USER suppresses the SAS portion of the message.

IC DELETE Statement

Deletes an integrity constraint.

Restriction: IC DELETE must be in a MODIFY RUN group

Syntax

IC DELETE *constraint-name(s)* | **_ALL_**;

Required Arguments

constraint-name(s)

names one or more constraints to delete. For example, to delete the constraints Unique_D and Unique_E, use the following statement: `ic delete Unique_D Unique_E;`

ALL

deletes all constraints for the SAS data file specified in the preceding MODIFY statement.

IC REACTIVATE Statement

Reactivates a foreign key integrity constraint that is inactive.

Restriction: IC REACTIVATE must be in a MODIFY RUN group

Syntax

IC REACTIVATE *foreign-key-name* REFERENCES *libref*;

Required Arguments

foreign-key-name

is the name of the foreign key to reactivate.

libref

refers to the SAS library containing the data set that contains the primary key that is referenced by the foreign key.

Example

Suppose that you have the foreign key FKEY defined in data set MyLib.MyOwn and that FKEY is linked to a primary key in data set MainLib.Main. If the integrity constraint is inactivated by a copy or move operation, you can reactivate the integrity constraint by using the following code:

```
proc datasets library=mylib;
    modify myown;
    ic reactivate fkey references mainlib;
run;
```

INDEX CENTILES Statement

Updates centiles statistics for indexed variables.

Restriction: INDEX CENTILES must be in a MODIFY RUN group

Syntax

INDEX CENTILES *index(s)*

</ <REFRESH> <UPDATECENTILES=ALWAYS | NEVER | *integer*>>;

Required Argument

index(s)

names one or more indexes.

Optional Arguments

REFRESH

updates centiles immediately, regardless of the value of UPDATECENTILES.

UPDATECENTILES=ALWAYS | NEVER | *integer*

specifies when centiles are to be updated. It is not practical to update centiles after every data set update. Therefore, you can specify as the value of UPDATECENTILES the percentage of the data values that can be changed before centiles for the indexed variables are updated.

Here is a list of valid values:

ALWAYS 0	updates centiles when the data set is closed if any changes have been made to the data set index. You can specify ALWAYS or 0 and produce the same results.
NEVER 101	does not update centiles. You can specify NEVER or 101 and produce the same results.
<i>integer</i>	is the percentage of values for the indexed variable that can be updated before centiles are refreshed. The alias is UPDCEN. The default is 5 (percent).

INDEX CREATE Statement

Creates simple or composite indexes for the SAS data set specified in the MODIFY statement.

Restriction: INDEX CREATE must be in a MODIFY RUN group

Example: [“Example 4: Modifying SAS Data Sets” on page 505](#)

Syntax

```
INDEX CREATE index-specification(s)
</ <NOMISS> <UNIQUE> <UPDATECENTILES=ALWAYS | NEVER | integer>>;
```

Required Argument

index-specification(s)

can be one or both of the following forms:

variable

creates a simple index on the specified variable.

index=(variables)

creates a composite index. The name that you specify for *index* is the name of the composite index. It must be a valid SAS name and cannot be the same as any variable name or any other composite index name. You must specify at least two variables.

Note The index name must follow the same rules as a SAS variable name, including avoiding the use of reserved names for automatic variables, such as `_N_`, and special variable list names, such as `_ALL_`.

Optional Arguments

NOMISS

excludes from the index all observations with missing values for all index variables.

When you create an index with the NOMISS option, SAS uses the index only for WHERE processing and only when missing values fail to satisfy the WHERE expression. For example, if you use the following WHERE statement, SAS does not use the index, because missing values satisfy the WHERE expression:

```
where dept ne '01';
```

BY-group processing ignores indexes that are created with the NOMISS option.

Example [“Example 4: Modifying SAS Data Sets” on page 505](#)

UNIQUE

specifies that the combination of values of the index variables must be unique. If you specify UNIQUE and multiple observations have the same values for the index variables, the index is not created.

Example [“Example 4: Modifying SAS Data Sets” on page 505](#)

UPDATECENTILES=ALWAYS | NEVER | *integer*

specifies when centiles are to be updated. It is not practical to update centiles after every data set update. Therefore, you can specify the percentage of the data values that can be changed before centiles for the indexed variables are updated. Here is a list of valid values:

ALWAYS 0	updates centiles when the data set is closed if any changes have been made to the data set index. You can specify ALWAYS or 0 and produce the same results.
NEVER 101	does not update centiles. You can specify NEVER or 101 and produce the same results.
<i>integer</i>	specifies the percentage of values for the indexed variable that can be updated before centiles are refreshed. The alias is UPDCEN and the default is 5 (percent).

INDEX DELETE Statement

Deletes one or more indexes associated with the SAS data set specified in the MODIFY statement.

Restriction: The INDEX DELETE statement must appear in a MODIFY RUN group

Note: You can use the CONTENTS statement to produce a list of all indexes for a data set.

Syntax

INDEX DELETE *index(s)* | **_ALL_**;

Required Arguments

index(s)

names one or more indexes to delete. The index(es) must be for variables in the SAS data set that is named in the preceding MODIFY statement. You can delete both simple and composite indexes.

ALL

deletes all indexes, except for indexes that are owned by an integrity constraint. When an index is created, it is marked as owned by the user, by an integrity constraint, or by both. If an index is owned by both a user and an integrity constraint, the index is not deleted until both an IC DELETE statement and an INDEX DELETE statement are processed.

INFORMAT Statement

Assigns, changes, and removes variable informats in the data set specified in the MODIFY statement permanently.

Restriction: The INFORMAT statement must appear in a MODIFY RUN group

Example: [“Example 4: Modifying SAS Data Sets” on page 505](#)

Syntax

INFORMAT *variable-1* *<informat-1>*
<variable-2 <informat-2> ...>;

Required Argument

variable

specifies one or more variables whose informats you want to assign, change, or remove. If you want to disassociate an informat with a variable, list the variable last in the list with no informat following:

```
informat a b 2. x1-x3 4.1 c;
```

Optional Argument

informat

specifies an informat for the variables immediately preceding it in the statement. If you do not specify an informat, the INFORMAT statement removes any existing informats for the variables in *variable-list*.

Tip To remove all informat from a data set, use the [ATTRIB statement on page 400](#) and the `_ALL_` keyword.

INITIATE Statement

Creates an audit file that has the same name as the SAS data file and a data set type of AUDIT.

Restrictions: The INITIATE statement must appear in an AUDIT RUN group.
An INITIATE statement is a required statement that can occur only once per audit file and must be placed directly after the AUDIT statement in the first AUDIT RUN group for that file.

Example: [“Example 8: Initiating an Audit File” on page 516](#)

Syntax

INITIATE <[AUDIT_ALL=NO | YES](#)>;

Optional Argument

AUDIT_ALL=NO | YES

specifies whether logging can be suspended and audit settings can be changed. `AUDIT_ALL=YES` specifies that all images are logged and cannot be suspended. That is, you cannot use the LOG statement to turn off logging of particular images, and you cannot suspend event logging by using the SUSPEND statement. To turn off logging, you must use the TERMINATE statement, which terminates event logging and deletes the audit file.

Default NO

Example [“Example 8: Initiating an Audit File” on page 516](#)

Details

The audit file logs additions, deletions, and updates to the SAS data file. You must initiate an audit trail before you can suspend, resume, or terminate it. Although the AUDIT statement immediately preceding the INITIATE statement cannot specify a GENNUM= option, if the specified file identifies a generation data set group, the audit file created by the INITIATE statement will be attached to the most recently created generation in the generation group.

The following example creates the audit file MyLib.MyFile.audit to log updates to the data file MyLib.MyFile.data, storing all available record images:

```
proc datasets library=mylib;
    audit myfile (alter=password);
```

```
initiate;
run;
```

The following example initiates an audit file using AUDIT_ALL=YES:

```
proc datasets lib=mylib; /* all audit image types will be logged
                           and the file cannot be suspended */
    audit myfile (alter=password);
    initiate audit_all=yes;
quit;
```

LABEL Statement

Assigns, changes, and removes variable labels for the SAS data set specified in the MODIFY statement.

Restriction: The LABEL statement must appear in a MODIFY RUN group

Example: [“Example 4: Modifying SAS Data Sets” on page 505](#)

Syntax

```
LABEL variable-1='label-1' | ' '
<variable-2=<'label-2' | ' '> ...>;
```

Required Argument

***variable*=<*'label'*>**

specifies a text string of up to 256 characters. If the label text contains single quotation marks, use double quotation marks around the label, or use two single quotation marks in the label text and enclose the string in single quotation marks. To remove a label from a data set, assign a label that is equal to a blank that is enclosed in quotation marks.

Range 1 - 256 characters

Tip To remove all variable labels in a data set, use the [ATTRIB statement on page 400](#) and the _ALL_ keyword.

LOG Statement

specifies the audit file settings.

Restriction: The LOG statement must appear after the INITIATE statement in an AUDIT RUN group.

Example: [“Example 8: Initiating an Audit File” on page 516](#)

Syntax

```
LOG <ADMIN_IMAGE=YES | NO>
<BEFORE_IMAGE=YES | NO>
<DATA_IMAGE=YES | NO>
<ERROR_IMAGE=YES | NO>;
```

Optional Arguments

ADMIN_IMAGE=YES | NO

specifies whether the administrative events are logged to the audit file (that is, the SUSPEND and RESUME actions).

Default YES

Tip If you do not want to log a particular image, specify NO for the image type. For example, the following code turns off logging the error images, but the administrative, before, and data images continue to be logged: `log error_image=no;`

BEFORE_IMAGE=YES | NO

specifies whether the before-update record images are logged to the audit file.

Default YES

DATA_IMAGE=YES | NO

specifies whether the added, deleted, and after-update record images are logged to the audit file.

Default YES

ERROR_IMAGE=YES | NO

specifies whether the after-update record images are logged to the audit file.

Default YES

Details

The following example creates the same audit file but stores only error record images:

```
proc datasets library=mylib;
  audit myfile (alter=password);
  initiate;
  log admin_image=no
    before_image=no
    data_image=no;
run;
```

MODIFY Statement

Changes the attributes of a SAS file and, through the use of subordinate statements, the attributes of variables in the SAS file.

Restrictions: Use in the Compute Server, but not in CAS.
You cannot change the length of a variable using the LENGTH= option in an ATTRIB statement.

Example: [“Example 4: Modifying SAS Data Sets” on page 505](#)

Syntax

```
MODIFY SAS-file <(options)>
</ <CORRECTENCODING=encoding-value> <DTC=SAS-date-time>
<GENNUM=integer> <MEMTYPE=member-type>>;
```

Required Argument

SAS-file

specifies a SAS file that exists in the procedure input library.

Optional Arguments

ALTER=password-modification

assigns, changes, or removes an Alter password for the SAS file named in the MODIFY statement. *password-modification* is one of the following:

- *new-password*
- *old-password* / *new-password*
- / *new-password*
- *old-password* /
- /

See [“Manipulating Passwords” on page 450](#)

CORRECTENCODING=encoding-value

enables you to change the encoding indicator, which is recorded in the file's descriptor information, in order to match the actual encoding of the file's data.

See [“CORRECTENCODING= Option” in SAS National Language Support \(NLS\): Reference Guide](#)

ENCRYPTKEY=key-value

specifies a key value for AES encryption.

Requirement ENCRYPTKEY= data set option is required if the data file has AES encryption.

DTC=SAS-date-time

specifies a date and time to substitute for the date and time stamp placed on a SAS file at the time of creation. You cannot use this option in parentheses after the name of each SAS file; you must specify DTC= after a forward slash:

```
modify mydata / dtc='03MAR00:12:01:00'dt;
```

Restrictions A SAS file's creation date and time cannot be set later than the date and time the file was actually created.

DTC= cannot be used with encrypted files or sequential files.

Tip Use DTC= to alter a SAS file's creation date and time before using the DATECOPY option in the COPY procedure, CPORT procedure, SORT procedure, and the COPY statement in the DATASETS procedure.

GENMAX=number-of-generations

specifies the maximum number of versions. Use this option in parentheses after the name of SAS file.

Default 0

Range 0 to 1,000

GENNUM=integer

restricts processing for generation data sets. You can specify GENNUM= either in parentheses after the name of each SAS file or after a forward slash. Valid value is *integer*, which is a number that references a specific version from a generation group. Specifying a positive number is an absolute reference to a specific generation number that is appended to a data set's name (that is, **gennum=2** specifies MYDATA#002). Specifying a negative number is a relative reference to a historical version in relation to the base version, from the youngest to the oldest (that is, **gennum=-1** refers to the youngest historical version). Specifying 0, which is the default, refers to the base version.

LABEL='data-set-label' | ''

assigns, changes, or removes a data set label for the SAS data set named in the MODIFY statement. If a single quotation mark appears in the label, write it as two single quotation marks. LABEL= or LABEL='' removes the current label.

Range 1 - 256 characters

Restriction A view label cannot be updated after the label is created.

Tip To remove all variable labels in a data set, use the [ATTRIB statement on page 400](#).

See [“Example 4: Modifying SAS Data Sets” on page 505](#)

MEMTYPE=member-type

restricts processing to one member type. You cannot specify MEMTYPE= in parentheses after the name of each SAS file; you must specify MEMTYPE= after a forward slash.

Aliases MTYPE=

MT=

Default If you do not specify the MEMTYPE= option in the PROC DATASETS statement or in the MODIFY statement, the default is MEMTYPE=DATA.

PW=password-modification

assigns, changes, or removes a Read, Write, or Alter password for the SAS file named in the MODIFY statement. *password-modification* is one of the following:

- *new-password*
- *old-password / new-password*
- */ new-password*
- *old-password /*
- */*

READ=password-modification

assigns, changes, or removes a Read password for the SAS file named in the MODIFY statement. *password-modification* is one of the following:

- *new-password*
- *old-password / new-password*
- */ new-password*
- *old-password /*
- */*

See [“Manipulating Passwords” on page 450](#)

Example [“Example 4: Modifying SAS Data Sets” on page 505](#)

SORTEDBY=sort-information

specifies how the data are currently sorted. SAS stores the sort information with the file but does not verify that the data are sorted the way you indicate. *sort-information* can be one of the following:

by-clause </collate-name> indicates how the data are currently sorted. Values for *by-clause* are the variables and options that you can use in a BY statement in a PROC SORT step. *collate-name* names the collating sequence used for the sort. By default, the collating sequence is that of your host operating environment.

NULL removes any existing sort indicator.

Restriction The data must be sorted in the order in which you specify. If the data is not in the specified order, SAS does not sort it for you.

Tip When using the MODIFY SORTEDBY option, you can also use a numbered range list or colon list. For more information, see [“Data Set Name Lists” in SAS Programmer’s Guide: Essentials](#).

Example [“Example 4: Modifying SAS Data Sets” on page 505](#)

TYPE=*special-type*

assigns or changes the special data set type of a SAS data set. SAS does not verify the following:

- the SAS data set type that you specify in the TYPE= option (except to check if it has a length of eight or fewer characters)
- that the SAS data set’s structure is appropriate for the type that you have designated

Note Do not confuse the TYPE= option with the MEMTYPE= option. The TYPE= option specifies a type of special SAS data set. The MEMTYPE= option specifies one or more types of SAS files in a SAS library.

Tip Most SAS data sets have no special type. However, certain SAS procedures, like the CORR procedure, can create a number of special SAS data sets. In addition, SAS/STAT software and SAS/EIS software support special data set types.

WRITE=*password-modification*

assigns, changes, or removes a Write password for the SAS file named in the MODIFY statement. *password-modification* is one of the following:

- *new-password*
- *old-password / new-password*
- */ new-password*
- *old-password /*
- */*

See [“Manipulating Passwords” on page 450](#)

Details

Changing Data Set Labels and Variable Labels

The LABEL option can change either the data set label or a variable label within the data set. To change a data set label, use the following syntax:

```
modify datasetname(label='Label for Data Set');
run;
```

You can change one or more variable labels within a data set. To change a variable label within the data set, use the following syntax:

```
modify datasetname;
    label variablename='Label for Variable';
run;
```

For an example of changing both a data set label and a variable label in the same PROC DATASETS, see [“Example 4: Modifying SAS Data Sets” on page 505](#).

Manipulating Passwords

In order to assign, change, or remove a password, you must specify the password for the highest level of protection that currently exists on that file.

Assigning Passwords

```
/* assigns a password to an unprotected file */
modify colors (pw=green);

/* assigns an Alter password to an already read-protected SAS data set */
modify colors (read=green alter=red);
```

Changing Passwords

```
/* changes the Write password from YELLOW to BROWN */
modify cars (write=yellow/brown);

/* uses Alter access to change unknown Read password to BLUE */
modify colors (read=/blue alter=red);
```

Removing Passwords

```
/* removes the Alter password RED from STATES */
modify states (alter=red/);

/* uses Alter access to remove the Read password */
modify zoology (read=green/ alter=red);

/* uses PW= as an alias for either WRITE= or ALTER= to remove unknown
   Read password */
modify biology (read=/ pw=red);
```

Working with Generation Groups

Changing the Number of Generations

```
/* changes the number of generations on data set A to 99 */
modify A (genmax=99);
```

Removing Passwords

```
/* removes the Alter password RED from STATES#002 */
modify states (alter=red/) / gennum=2;
```

QUIT Statement

Executes any statements that have not executed and ends the procedure.

Syntax

QUIT;

REBUILD Statement

Specifies whether to restore or delete the disabled indexes and integrity constraints.

Default: Rebuild indexes and integrity constraints

Restriction: The REBUILD statement applies only to data sets created in Version 7 or later

Syntax

```
REBUILD SAS-file </ <ENCRYPTKEY=key-value>
<ALTER=password> <GENNUM=n> <MEMTYPE=member-type> <NOINDEX>>;
```

Required Argument

SAS-file

specifies a SAS data file that contains the disabled indexes and integrity constraints. You can also use a numbered range list or colon list.

See [“Data Set Name Lists” in SAS Programmer’s Guide: Essentials](#).

Optional Arguments

ENCRYPTKEY=*key-value*

specifies a key value for AES encryption.

Requirement ENCRYPTKEY= data set option is required if the data file has AES encryption.

ALTER=*alter-password*

provides the Alter password for any alter-protected SAS files that are named in the REBUILD statement. You can use the option either in parentheses after the name of each SAS file or after a forward slash.

GENNUM=*integer*

restricts processing for generation data sets. You can use the option either in parentheses after the name of each SAS file or after a forward slash. Valid value is *integer*, which is a number that references a specific version from a generation group. Specifying a positive number is an absolute reference to a specific generation number that is appended to a data set’s name (that is, **gennum=2** specifies MYDATA#002). Specifying a negative number is a relative reference to a historical version in relation to the base version, from the youngest to the oldest (that is, **gennum=-1** refers to the youngest historical version). Specifying 0, which is the default, refers to the base version.

MEMTYPE=member-type

restricts processing to one member type.

Aliases MTYPE=

MT=

Default If you do not specify the MEMTYPE= option in the PROC DATASETS statement or in the REBUILD statement, the default is MEMTYPE=ALL.

NOINDEX

specifies to delete the disabled indexes and integrity constraints.

Restriction The NOINDEX option cannot be used for data files that contain one or more referential integrity constraints.

Details

When the DLDMGACTION=NOINDEX data set or system option is specified and SAS encounters a damaged data file, SAS does the following:

- repairs the data file without indexes and integrity constraints
- deletes the index file
- updates the data file to reflect the disabled indexes and integrity constraints
- limits the data file to be opened only in INPUT mode
- writes the following warning to the SAS log:

```
WARNING: SAS data file MYLIB.MYFILE.DATA was damaged and
        has been partially repaired. To complete the repair,
        execute the DATASETS procedure REBUILD statement.
```

The REBUILD statement completes the repair of a damaged SAS data file by rebuilding or deleting all of the data file's disabled indexes and integrity constraints. The REBUILD statement establishes and uses member-level locking in order to process the new index file and to restore the indexes and integrity constraints.

To rebuild the index file and restore the indexes and integrity constraints, use the following code:

```
proc datasets library=mylib;
  rebuild myfile
  /alter=password
  gennum=n
  memtype=mytype;
```

To delete the disabled indexes and integrity constraints, use the following code:

```
proc datasets library=mylib;
  rebuild myfile /noindex;
```

After you execute the REBUILD statement, the data file is no longer restricted to INPUT mode.

The REBUILD statement default is to rebuild the indexes and integrity constraints and the index file.

If a data file contains one or more referential integrity constraints and you use the NOINDEX option with the REBUILD statement, the following error message is written to the SAS log:

```
Error: Unable to rebuild data file MYLIB.MYFILE.DATA using the
      NOINDEX option because the data file contains referential
      constraints. Resubmit the REBUILD statement without the
      NOINDEX option to restore the data file.
```

RENAME Statement

Renames variables in the SAS data set specified in the MODIFY statement.

Restriction: The RENAME statement must appear in a MODIFY RUN group

Example: [“Example 4: Modifying SAS Data Sets” on page 505](#)

Syntax

```
RENAME old-name-1=new-name-1
<old-name-2=new-name-2 ...>;
```

Required Argument

old-name=new-name

changes the name of a variable in the data set specified in the MODIFY statement. *old-name* must be a variable that already exists in the data set. *new-name* cannot be the name of a variable that already exists in the data set or the name of an index, and the new name must be a valid SAS name.

For example, assume the Oxygen data set includes a variable named OXYGEN. The following code renames the OXYGEN variable to intake:

```
modify oxygen;
      rename oxygen=intake;
```

Alphabetic List of Variables and Attributes				
#	Variable	Type	Len	Label
1	AGE	Num	8	
6	MAXPULSE	Num	8	
4	RSTPULSE	Num	8	
5	RUNPULSE	Num	8	
3	RUNTIME	Num	8	
2	WEIGHT	Num	8	
7	intake	Num	8	Intake Measurement

Details

If *old-name* does not exist in the SAS data set or *new-name* already exists, PROC DATASETS stops processing the RUN group containing the RENAME statement and issues an error message.

When you use the RENAME statement to change the name of a variable for which there is a simple index, the statement also renames the index.

If the variable that you are renaming is used in a composite index, the composite index automatically references the new variable name. However, if you attempt to rename a variable to a name that has already been used for a composite index, you receive an error message.

REPAIR Statement

Attempts to restore damaged SAS data sets or catalogs, in permanent libraries, to a usable condition.

Note: The REPAIR statement is not a replacement for having a current backup.

Syntax

```
REPAIR SAS-file(s)
  </ <ENCRYPTKEY=key-value>
  <ALTER=alter-password> <GENNUM=integer> <MEMTYPE=member-type>>;
```

Required Argument

SAS-file(s)

specifies one or more SAS data sets or catalogs in the procedure input library. You can also use a numbered range list or colon list.

See [“Data Set Name Lists” in SAS Programmer’s Guide: Essentials](#)

Optional Arguments

ALTER=alter-password

provides the Alter password for any alter-protected SAS files that are named in the REPAIR statement. You can use the option either in parentheses after the name of each SAS file or after a forward slash.

See [“Using Passwords with the DATASETS Procedure” on page 467](#)

ENCRYPTKEY=key-value

specifies a key value for AES encryption.

Requirement ENCRYPTKEY= data set option is required if the data file has AES encryption.

GENNUM=integer

restricts processing for generation data sets. You can use the option either in parentheses after the name of each SAS file or after a forward slash. Valid value is *integer*, which is a number that references a specific version from a generation group. Specifying a positive number is an absolute reference to a specific generation number that is appended to a data set's name (that is, **gennum=2** specifies MYDATA#002). Specifying a negative number is a relative reference to a historical version in relation to the base version, from the youngest to the oldest (that is, **gennum=-1** refers to the youngest historical version). Specifying 0, which is the default, refers to the base version.

See [“Restricting Processing for Generation Data Sets” on page 468](#)

MEMTYPE=member-type

restricts processing to one member type.

Aliases MTYPE=

MT=

Default If you do not specify the MEMTYPE= option in the PROC DATASETS statement or in the REPAIR statement, the default is MEMTYPE=ALL.

See [“Restricting Member Types for Processing” on page 469](#)

Details

CAUTION

If you have extensive damage to your data set, the REPAIR statement will not correct it. If the device on which a SAS data set or an auxiliary file (index, audit, or extended attribute file) resides is damaged, then you must restore the damaged data set and auxiliary files from a backup device.

The most common situations where the REPAIR statement might be helpful are as follows:

- A system failure occurs while you are updating a SAS data set or catalog.

When you use the REPAIR statement for SAS data sets, it re-creates all indexes for the data set. It also attempts to restore the data set to a usable condition, but the restored data set might not include the last several updates that occurred before the system failed. You cannot use the REPAIR statement to re-create indexes that were destroyed by using the FORCE option in a PROC SORT step.

- An I/O error occurs while you are writing a SAS data set or catalog entry.

If the disk that stores the SAS data set becomes full before the file is completely written to disk, the step that writes the data set will fail, and an error is written to the SAS log. The REPAIR statement can repair the data set header and perhaps offer a workable data set. The data set will likely be incomplete and its integrity questionable. The best recourse is to re-create the data set from a backup.

Note that observations might be lost during the repair process.

When you use the REPAIR statement for a catalog, you receive a message stating whether the REPAIR statement restored the entry. If the entire catalog is potentially damaged, the REPAIR statement attempts to restore all the entries in the catalog. If only a single entry is potentially damaged, for example, when a single entry is being updated and a disk-full condition occurs, on most systems only the entry that is open when the problem occurs is potentially damaged. In this case, the REPAIR statement attempts to repair only that entry. Some entries within the restored catalog might not include the last updates that occurred before a system crash or an I/O error. The REPAIR statement issues warning messages for entries that might have truncated data.

To repair a damaged catalog, you must use a version of SAS that can update the catalog.

If you issue a REPAIR statement for a SAS file that does not exist in the specified library, PROC DATASETS stops processing the run group that contains the REPAIR statement, and issues an error message. To override this behavior, use the NOWARN option in the PROC DATASETS statement.

If you are using Cross-Environment Data Access (CEDA) to process a foreign SAS data set that has become damaged, you must move the data set back to its native environment before you attempt to repair it using the PROC DATASETS REPAIR statement. CEDA does not support update processing, which is required in order to repair a damaged data set.

For more information about CEDA, see [“Definitions for Cross-Environment Data Access \(CEDA\)” in SAS Programmer’s Guide: Essentials](#).

RESUME Statement

resumes event logging to the audit file, if it was suspended.

Restriction: The RESUME statement must appear in an AUDIT RUN group.

Example: [“Example 8: Initiating an Audit File” on page 516](#)

Syntax

RESUME;

Details

No other audit-related statement, such as RESUME, SUSPEND, TERMINATE, USER_VAR, or LOG, will be valid for an audit file until the INITIATE statement has been submitted. For more information, see [“INITIATE Statement” on page 443](#).

The RESUME statement requires the LOG statement ADMIN_IMAGE=YES option. This option specifies whether the administrative events are logged to the audit file (that is, the SUSPEND and RESUME actions). For more information, see [“LOG Statement” on page 444](#).

SAVE Statement

Deletes all the SAS files in a library except the ones listed in the SAVE statement.

Example: [“Example 3: Saving SAS Files from Deletion” on page 502](#)

Syntax

SAVE *SAS-file(s)* *</ MEMTYPE=member-type>*;

Required Argument

SAS-file(s)

specifies one or more SAS files that you do not want to delete from the SAS library.

Note: If a SAS file has an ALTER= password assigned, it must be specified in order to delete the SAS file.

Optional Argument

MEMTYPE=member-type

restricts processing to one member type. You can use the option either in parentheses after the name of each SAS file or after a forward slash.

Aliases MTYPE=

MT=

- Default If you do not specify the MEMTYPE= option in the PROC DATASETS statement or in the SAVE statement, the default is MEMTYPE=ALL.
- See [“Restricting Member Types for Processing” on page 469](#)
- Example [“Example 3: Saving SAS Files from Deletion” on page 502](#)

Details

If one of the SAS files in *SAS-file* does not exist in the procedure input library, PROC DATASETS stops processing the RUN group containing the SAVE statement and issues an error message. To override this behavior, specify the NOWARN option in the PROC DATASETS statement.

When the SAVE statement deletes SAS data sets, it also deletes any indexes associated with those data sets. (If the SAS data set that is to be deleted has an ALTER= password assigned to it, the ALTER= password must be specified in order to delete the SAS data set.)

CAUTION

SAS immediately deletes libraries and library members when you submit a RUN group. You are not asked to verify the Delete operation before it begins. The SAVE statement deletes many SAS files in one operation. Make sure that you understand how the MEMTYPE= option affects which types of SAS files are saved and which types are deleted.

When you use the SAVE statement with generation groups, the SAVE statement treats the base version and all historical versions as a unit. You cannot save a specific version.

SELECT Statement

Selects SAS files for copying.

Restrictions: The SELECT statement must follow a COPY statement
The SELECT statement cannot appear with an EXCLUDE statement in the same COPY step

Example: [“Example 2: Manipulating SAS Files” on page 497](#)

Syntax

```
SELECT SAS-file(s)
</ <ENCRYPTKEY=key-value> <ALTER=alter-password> <MEMTYPE=member-
type>>;
```


Required Argument

SAS-file(s)

specifies one or more SAS files that you want to copy. All of the SAS files that you name must be in the data library that is referenced by the libref named in the IN= option in the COPY statement. If the SAS files have generation groups, all the generations are copied because the SELECT statement does not allow you to select specific versions.

Optional Arguments

ALTER=alter-password

provides the Alter password for any alter-protected SAS files that you are moving from one data library to another. Because you are moving and thus deleting a SAS file from a SAS library, you need Alter access. You can use the option either in parentheses after the name of each SAS file or after a forward slash.

See [“Using Passwords with the DATASETS Procedure” on page 467](#)

ENCRYPTKEY=key-value

specifies the key value for AES encryption.

See [“Appending AES-Encrypted Data Sets” on page 395](#)

MEMTYPE=member-type

restricts processing to one member type. You can use the option either in parentheses after the name of each SAS file or after a forward slash.

Aliases MTYPE=

MT=

Default If you do not specify the MEMTYPE= option in the PROC DATASETS statement, in the COPY statement, or in the SELECT statement, the default is MEMTYPE=ALL.

See [“Specifying Member Types When Copying or Moving SAS Files” on page 423](#) and [“Restricting Member Types for Processing” on page 469](#)

Example [“Example 2: Manipulating SAS Files” on page 497](#)

Details

Selecting Several Like-Named Files

You can use shortcuts for listing several SAS files in the SELECT statement:

Table 15.13 Using the SELECT Statement

Notation	Meaning
----------	---------

x1–xn	Specifies files X1 through Xn. The numbers must be consecutive.
x:	Specifies all files that begin with the letter X.

SUSPEND Statement

suspends event logging to the audit file, but does not delete the audit file.

Restriction: The SUSPEND statement must appear in an AUDIT RUN group.

Example: [“Example 8: Initiating an Audit File” on page 516](#)

Syntax

SUSPEND;

Details

No other audit-related statement, such as SUSPEND, RESUME, TERMINATE, USER_VAR, or LOG, will be valid for an audit file until the INITIATE statement has been submitted. For more information, see [“INITIATE Statement” on page 443](#).

The SUSPEND statement requires the LOG statement ADMIN_IMAGE=YES option. This option specifies whether the administrative events are logged to the audit file (that is, the SUSPEND and RESUME actions). For more information, see [“LOG Statement” on page 444](#).

TERMINATE Statement

terminates event logging and deletes the audit file.

Restriction: The TERMINATE statement must appear in an AUDIT RUN group.

Example: [“Example 8: Initiating an Audit File” on page 516](#)

Syntax

TERMINATE;

Details

No other audit-related statement, such as `TERMINATE`, `SUSPEND`, `RESUME`, `USER_VAR`, or `LOG`, will be valid for an audit file until the `INITIATE` statement has been submitted. For more information, see [“INITIATE Statement” on page 443](#).

USER_VAR Statement

defines optional variables to be logged in the audit file with each update to an observation. When you use `USER_VAR`, it must follow an `INITIATE` statement.

Restrictions: The `USER_VAR` statement must appear in an `AUDIT RUN` group.
The `USER_VAR` statement is optional. If specified, the `USER_VAR` statement must immediately follow the `INITIATE` statement for the applicable audit file.

Example: [“Example 8: Initiating an Audit File” on page 516](#)

Syntax

```
USER_VAR variable-name-1 <$> <length> <LABEL='variable-label' >
<variable-name-2 <$> <length> <LABEL='variable-label' > ...>;
```

Required Argument

variable-name

is a name for the variable.

Optional Arguments

\$

indicates that the variable is a character variable.

length

specifies the length of the variable.

Default 8

LABEL='variable-label'

specifies a label for the variable.

XATTR ADD Statement

Adds an extended attribute to a variable or a data set.

Restrictions: Use in the Compute Server, but not in CAS.

The XATTR ADD statement must appear in a MODIFY RUN group
Generation data sets do not support extended attributes.

Notes:

An extended attribute can have numeric or character values.

A blank space in a character value indicates a missing value. Missing numeric values are also allowed.

An extended attribute name must conform to SAS naming rules. A SAS name can be up to 32 characters long.

Example:

[“Example 9: Extended Attributes” on page 522](#)

Syntax

XATTR ADD DS *attribute-name-1=attribute-value-1*
<*attribute-name-2=attribute-value-2 ...*>;

or

XATTR ADD VAR *variable-name-1 (attribute-name-1=attribute-value-1*
<*attribute-name-2=attribute-value-2 ...*>
<*variable-name-2 (attribute-name-1=attribute-value-1*
<*attribute-name-2=attribute-value-2 ...*>) >;

Required Arguments

Note that for character values, *attribute-value* must be in quotation marks, such as "*attribute-value*".

XATTR ADD DS *attribute-name-1=attribute-value-1* <*attribute-name-2=attribute-value-2 ...*>

adds an extended attribute to a data set. If the extended attribute already exists, an error will be returned.

XATTR ADD VAR *variable-name-1 (attribute-name-1=attribute-value-1* <*attribute-name-2=attribute-value-2 ...*>) <*variable-name-2 (attribute-name-1=attribute-value-1* <*attribute-name-2=attribute-value-2 ...*>) >

adds an extended attribute to a variable. If the extended attribute already exists, an error will be returned.

Details

Extended attributes are organized into (name, value) pairs. If you try to add a new attribute and the attribute already exists, an error is written to the SAS log.

XATTR DELETE Statement

Deletes all extended attributes from a SAS file.

- Restrictions: Use in the Compute Server, but not in CAS.
The XATTR DELETE statement must appear in a MODIFY RUN group
- Example: [“Example 9: Extended Attributes” on page 522](#)

Syntax

XATTR DELETE;

Required Argument

XATTR DELETE

deletes all extended attributes from a data set.

Details

Use the XATTR DELETE statement to delete all of the extended attributes from a data set. None of the extended attributes will exist after using this command. The following example deletes all extended attributes from a data set:

```
proc datasets lib=library_name nolist;
    modify dataset_name;
        xattr delete;
    run;
quit;
```

XATTR OPTIONS Statement

Specifies options as needed for extended attributes. Currently, only SEGLLEN= is a valid option.

- Restrictions: Use in the Compute Server, but not in CAS.
The XATTR OPTIONS statement must appear in a MODIFY RUN group
- Example: [“Example 9: Extended Attributes” on page 522](#)

Syntax

XATTR OPTIONS <*SEGLLEN=number-of-bytes*>;

Required Argument

SEGLLEN=number-of-bytes

Indicates the length of the storage element that will hold the character extended attribute value. The value can be 1 to 32,760 bytes.

XATTR REMOVE Statement

Removes an extended attribute from a variable or a data set.

Restrictions: Use in the Compute Server, but not in CAS.
The XATTR REMOVE statement must appear in a MODIFY RUN group

Example: [“Example 9: Extended Attributes” on page 522](#)

Syntax

XATTR REMOVE DS *attribute-name(s)* ;

or

XATTR REMOVE VAR *variable-name-1 (attribute-name(s))*
<*variable-name-2 (attribute-name(s) ...)*>;

Required Arguments

XATTR REMOVE DS *attribute-name(s)*

removes an extended attribute from a data set.

XATTR REMOVE VAR *variable-name-1 (attribute-name(s))* <*variable-name-2 (attribute-name(s))*>

removes an extended attribute from a variable.

Details

If you no longer need an extended attribute that you created, use the XATTR REMOVE statement to remove it from a variable or a data set. The XATTR REMOVE statement deletes only the extended attribute that you specify.

XATTR SET Statement

Updates or adds extended attributes to variables or data sets.

Restrictions: Use in the Compute Server, but not in CAS.
The XATTR SET statement must appear in a MODIFY RUN group
Generation data sets do not support extended attributes.

- Notes: An extended attribute can have numeric or character values.
 A blank space in a character value indicates a missing value. Missing numeric values are also allowed.
 An extended attribute name must conform to SAS naming rules. A SAS name can be up to 32 characters long.
- Example: [“Example 9: Extended Attributes” on page 522](#)

Syntax

XATTR SET DS *attribute-name-1=attribute-value-1*
<attribute-name-2="attribute-value-2" ...>;

or

XATTR SET VAR *variable-name-1 (attribute-name-1=attribute-value-1*
<attribute-name-2=attribute-value-2 ... >)
<variable-name-2 (attribute-name-1=attribute-value-1
<attribute-name-2=attribute-value-2> ...)>;

Required Arguments

Note that for character values, *attribute-value* must be in quotation marks, such as "*attribute-value*".

XATTR SET DS *attribute-name-1=attribute-value-1 <attribute-name-2=attribute-value-2 ...>*

updates or adds an extended attribute to a data set. If the data set extended attribute does not exist, it will be added. If it does exist, it will be updated with the value specified.

XATTR SET VAR *variable-name-1 (attribute-name-1=attribute-value-1 <attribute-name-2=attribute-value-2 ...>) <variable-name-2 (attribute-name-1=attribute-value-1 <attribute-name-2=attribute-value-2 ...>)>*

updates or adds an extended attribute to a variable. If the variable and extended attribute combination does not exist, the extended attribute will be added. If it does exist, the extended attribute will be updated with the one specified.

Details

Use the XATTR SET statement if you are not sure if an extended attribute exists. If an extended attribute does exist, it will be updated. If the extended attribute does not exist, it added. The XATTR SET statement defines the variable or data set extended attribute even if it does not exist yet. When using XATTR ADD, an error occurs if there is an existing extended attribute using that value. You also get an error if you try to use XATTR UPDATE on an extended attribute that does not exist yet. Using XATTR SET defines the variable or data set extended attributes. If the extended attribute did not exist, it does now. If the extended attribute did exist, then it has a new value.

XATTR UPDATE Statement

Updates an extended attribute to a variable or a data set.

Restrictions:	Use in the Compute Server, but not in CAS. The XATTR UPDATE statement must appear in a MODIFY RUN group
Notes:	A blank space in a character value indicates a missing value. Missing numeric values are also allowed. An extended attribute name must conform to SAS naming rules. A SAS name can be up to 32 characters long.
Example:	“Example 9: Extended Attributes” on page 522

Syntax

XATTR UPDATE DS *attribute-name-1=attribute-value-1*
<attribute-name-2=attribute-value-2 ...>;

or

XATTR UPDATE VAR *variable-name-1 (attribute-name-1=attribute-value-1*
<attribute-name-2=attribute-value-2 ...>)
<variable-name-2 (attribute-name-1=attribute-value-1
<attribute-name-2=attribute-value-2 ...>)>;

Required Arguments

Note that for character values, *attribute-value* must be in quotation marks, such as "*attribute-value*".

XATTR UPDATE DS *attribute-name-1=attribute-value-1 <attribute-name-2=attribute-value-2 ...>*

updates an extended attribute in a data set. If the extended attribute does not exist, an error is written to the SAS log.

XATTR UPDATE VAR *variable-name-1 (attribute-name-1=attribute-value-1 <attribute-name-2=attribute-value-2 ...>) <variable-name-2 (attribute-name-1=attribute-value-1 <attribute-name-2=attribute-value-2 ...>)>*

updates an extended attribute for a variable. If the variable and extended attribute combination is not found, an error message is written to the SAS log.

Details

To make changes to an existing extended attribute, use the XATTR UPDATE statement. If you try to update an extended attribute that does not exist, an error is written to the SAS log.

Usage: DATASETS Procedure

Using Passwords with the DATASETS Procedure

Several statements in the DATASETS procedure support options that manipulate passwords on SAS files. These options, ALTER=, PW=, READ=, and WRITE=, are also data set options.¹ If you do not know how passwords affect SAS files, see [“Passwords” in SAS Programmer’s Guide: Essentials](#).

When you are working with password-protected SAS files in the AGE, CHANGE, DELETE, EXCHANGE, REPAIR, or SELECT statement, you can specify ALTER= and PW= password options in the PROC DATASETS statement or in the subordinate statement.

Note: The ALTER= option works slightly different for the COPY (when moving a file) and MODIFY statements. For more information, see [COPY statement on page 413](#) and the [MODIFY statement on page 446](#).

SAS searches for passwords in the following order:

- 1 in parentheses after the name of the SAS file in a subordinate statement. When used in parentheses, the option refers only to the name immediately preceding the option. If you are working with more than one SAS file in a data library and each SAS file has a different password, you must specify password options in parentheses after individual names.

In the following statement, the ALTER= option provides the password `red` for the SAS file `Bones` only:

```
delete xplant bones(alter=red);
```

- 2 after a forward slash (/) in a subordinate statement. When you use a password option following a slash, the option refers to all SAS files named in the statement unless the same option appears in parentheses after the name of a SAS file. This method is convenient when you are working with more than one SAS file and they all have the same password.

In the following statement, the ALTER= option in parentheses provides the password `red` for the SAS file `Chest`, and the ALTER= option after the slash provides the password `blue` for the SAS file `Virus`:

```
delete chest(alter=red) virus / alter=blue;
```

1. In the APPEND and CONTENTS statements, you use these options just as you use any SAS data set option, in parentheses after the SAS data set name.

- 3 in the PROC DATASETS statement. Specifying the password in the PROC DATASETS statement can be useful if all the SAS files that you are working with in the library have the same password. Do not specify the option in parentheses.

In the following PROC DATASETS step, the PW= option provides the password red for the SAS files Insulin and Abneg:

```
proc datasets pw=red;
    delete insulin;
    contents data=abneg;
run;
```

Note: For the password for a SAS file in a SELECT statement, SAS looks in the COPY statement before it looks in the PROC DATASETS statement.

Restricting Processing for Generation Data Sets

Several statements in the DATASETS procedure support the GENNUM= option to restrict processing for generation data sets. GENNUM= is also a data set option.¹ If you do not know how to request and use generation data sets, see “Generation Data Sets” in For information to repair generation data sets, see [“Generation Data Sets” in SAS V9 LIBNAME Engine: Reference](#).

When you are working with a generation group for the AUDIT, CHANGE, DELETE, MODIFY, and REPAIR statements, you can restrict processing in the PROC DATASETS statement or in the subordinate statement to a specific version.

Note: The GENNUM= option works slightly different for the MODIFY statement. See [MODIFY statement on page 446](#).

Note: You cannot restrict processing to a specific version for the AGE, COPY, EXCHANGE, and SAVE statements. These statements apply to the entire generation group.

SAS searches for a generation specification in the following order:

- 1 in parentheses after the name of the SAS data set in a subordinate statement. When used in parentheses, the option refers only to the name immediately preceding the option. If you are working with more than one SAS data set in a data library and you want a different generation version for each SAS data set, then you must specify GENNUM= in parentheses after individual names.

In the following statement, the GENNUM= option specifies the version of a generation group for the SAS data set Bones only:

```
delete xplant bones (gennum=2);
```

1. For the APPEND and CONTENTS statements, use GENNUM= just as you use any SAS data set option, in parentheses after the SAS data set name.

- 2 after a forward slash (/) in a subordinate statement. When you use the GENNUM= option following a slash, the option refers to all SAS data sets named in the statement unless the same option appears in parentheses after the name of a SAS data set. This method is convenient when you are working with more than one file and you want the same version for all files.

In the following statement, the GENNUM= option in parentheses specifies the generation version for SAS data set Chest, and the GENNUM= option after the slash specifies the generation version for SAS data set Virus:

```
delete chest (gennum=2) virus / gennum=1;
```

- 3 in the PROC DATASETS statement. Specifying the generation version in the PROC DATASETS statement can be useful if you want the same version for all of the SAS data sets you are working with in the library. Do not specify the option in parentheses.

In the following PROC DATASETS step, the GENNUM= option specifies the generation version for the SAS files Insulin and Abneg:

```
proc datasets gennum=2;
  delete insulin;
  contents data=abneg;
run;
```

Note: For the generation version for a SAS file in a SELECT statement, SAS looks in the COPY statement before it looks in the PROC DATASETS statement.

Restricting Member Types for Processing

In the PROC DATASETS Statement

If you reference more than one member type in subordinate statements and you have a specified member type in the PROC DATASETS statement, then include all of the member types in the PROC DATASETS statement. Only the member type or types in the original PROC DATASETS statement is in effect. The following example lists multiple member types:

```
proc datasets lib=library memtype=(data view);
```

In Subordinate Statements

Use the MEMTYPE= option in the following subordinate statements to limit the member types that are available for processing:

```
AGE    CHANGE    DELETE    EXCHANGE
```

EXCLUDE REPAIR SAVE SELECT

Note: The MEMTYPE= option works slightly differently for the CONTENTS, COPY, and MODIFY statements. For more information, see [CONTENTS statement on page 405](#), [COPY Statement on page 413](#), and [MODIFY statement on page 446](#).

The procedure searches for MEMTYPE= in the following order:

- 1 in parentheses immediately after the name of a SAS file. When used in parentheses, the MEMTYPE= option refers only to the SAS file immediately preceding the option. For example, the following statement deletes House.data, Lot.catalog, and Sales.data because the default member type for the DELETE statement is DATA. (For more information, see [Table 15.48 on page 471](#) for the default types for each statement.)

```
delete house lot(memtype=catalog) sales;
```

- 2 after a slash (/) at the end of the statement. When used following a slash, the MEMTYPE= option refers to all SAS files named in the statement unless the option appears in parentheses after the name of a SAS file. For example, the following statement deletes Lotpix.catalog, Regions.data, and Appl.catalog:

```
delete lotpix regions(memtype=data) appl / memtype=catalog;
```

- 3 in the PROC DATASETS statement. For example, this DATASETS procedure deletes Appl.catalog:

```
proc datasets memtype=catalog;
    delete appl;
run;
```

Note: When you use the EXCLUDE and SELECT statements, the procedure looks in the COPY statement for the MEMTYPE= option before it looks in the PROC DATASETS statement. For more information, see [“Specifying Member Types When Copying or Moving SAS Files” on page 423](#).

- 4 for the default value. If you do not specify a MEMTYPE= option in the subordinate statement or in the PROC DATASETS statement, the default value for the subordinate statement determines the member type available for processing.

Member Types

The following list gives the possible values for the MEMTYPE= option:

ACCESS

access descriptor files (created by SAS/ACCESS software)

ALL

all member types

CATALOG
SAS catalogs

DATA
SAS data files

FDB
financial database

MDDDB
multidimensional database

PROGRAM
stored compiled SAS programs

VIEW
SAS views

The following table shows the member types that you can use in each statement:

Table 15.14 Subordinate Statements and Appropriate Member Types

Statement	Appropriate Member Types	Default Member Type
AGE	ACCESS, CATALOG, DATA, FDB, MDDDB, PROGRAM, VIEW	DATA
CHANGE	ACCESS, ALL, CATALOG, DATA, FDB, MDDDB, PROGRAM, VIEW	ALL
CONTENTS	ALL, DATA, VIEW	DATA ¹
COPY	ACCESS, ALL, CATALOG, DATA, FDB, MDDDB, PROGRAM, VIEW	ALL
DELETE	ACCESS, ALL, CATALOG, DATA, FDB, MDDDB, PROGRAM, VIEW	DATA
EXCHANGE	ACCESS, ALL, CATALOG, DATA, FDB, MDDDB, PROGRAM, VIEW	ALL
EXCLUDE	ACCESS, ALL, CATALOG, DATA, FDB, MDDDB, PROGRAM, VIEW	ALL
MODIFY	ACCESS, DATA, VIEW	DATA
REPAIR	ALL, CATALOG, DATA	ALL ²
SAVE	ACCESS, ALL, CATALOG, DATA, FDB, MDDDB, PROGRAM, VIEW	ALL

Statement	Appropriate Member Types	Default Member Type
SELECT	ACCESS, ALL, CATALOG, DATA, FDB, MDDB, PROGRAM, VIEW	ALL

- 1 When DATA=_ALL_ in the CONTENTS statement, the default is ALL. ALL includes only DATA and VIEW.
- 2 ALL includes only DATA and CATALOG.

Sample PROC DATASETS Output

The DATASETS procedure includes the following:

- copies all data sets from the Control library to the Health library
- lists the contents of the Health library
- deletes the Syndrome data set from the Health library
- changes the name of the Prenat data set to Infant

The SAS log is shown in the following output.

```
LIBNAME control 'SAS-library-1';
LIBNAME health 'SAS-library-2';

proc datasets memtype=data;
    copy in=control out=health;
run;

proc datasets library=health memtype=data details;
    delete syndrome;
    change prenat=infant;
run;
quit;
```

Example Code 15.1 Log from PROC DATASETS

```

744 proc datasets library=health memtype=data details;
      Directory

Libref      HEALTH
Engine      V9
Physical Name /home/userid/Documents and Settings/myfil/My Documents/procdatasets/health
Filename    /home/userid/Documents and Settings/myfile/My Documents/procdatasets/health

#  Name      Member  Obs, Entries      File
      Type    or Indexes  Vars  Label      Size  Last Modified

1  ALL       DATA     23      17
2  BODYFAT   DATA     83      13  California Results
3  CONFOUND  DATA      8       4
4  CORONARY  DATA     39       4
5  DRUG1     DATA      6       2  JAN2005 DATA
6  DRUG2     DATA     13       2  MAY2005 DATA
7  DRUG3     DATA     11       2  JUL2005 DATA
8  DRUG4     DATA      7       2  JAN2002 DATA
9  DRUG5     DATA      1       2  JUL2002 DATA
10 GROUP     DATA    148     11  Test Subjects
    GROUP     INDEX      1
11 GRPOUT   DATA     11      40
12 GRPOUT1  DATA     11      40
13 INFANT    DATA    149       6
14 MLSCCL   DATA     32       4  Multiple Sclerosis Data
15 NAMES    DATA      7       4
16 OXYGEN    DATA     31       7
17 PERSONL   DATA    148     11
18 PHARM     DATA      6       3  Sugar Study
19 POINTS    DATA      6       6
20 RESULTS   DATA     10       5
21 SLEEP     DATA    108       6
22 TEST2     DATA     15       5
23 TRAIN     DATA      7       2
24 VISION    DATA     16       3
25 WEIGHT    DATA      1       2
26 WGHT      DATA     83      13

745 delete syndrome;
746 change prenatal=infant;
747 run;
ERROR: The file HEALTH.PRENAT (memtype=DATA) was not found, but appears on a CHANGE statement.
748 quit;
749
750 proc datasets memtype=data;

```

Results: DATASETS Procedure

Directory Listing to the SAS Log

The PROC DATASETS statement lists the SAS files in the procedure input library unless the NOLIST option is specified. The NOLIST option prevents the creation of the procedure results that go to the log. If you specify the MEMTYPE= option, only specified types are listed. If you specify the DETAILS option, PROC DATASETS prints these additional columns of information: **Obs**, **Entries or Indexes**, **Vars**, and **Label**.

Directory Listing as SAS Output

The CONTENTS statement lists the directory of the procedure input library if you use the DIRECTORY option or specify DATA=_ALL_.

If you want only a directory, use the NODS option and the _ALL_ keyword in the DATA= option. The NODS option suppresses the description of the SAS data sets; only the directory appears in the output.

Note: The CONTENTS statement does not put a directory in an output data set. If you try to create an output data set using the NODS option, you receive an empty output data set. Use the SQL procedure to create a SAS data set that contains information about a SAS library.

Note: If you specify the ODS RTF destination, the PROC DATASETS output goes to both the SAS log and the ODS output area. The NOLIST option suppresses output to both. To see the output only in the SAS log, use the ODS EXCLUDE statement by specifying the member directory as the exclusion.

Procedure Output

The CONTENTS Statement

The only statement in PROC DATASETS that produces procedure output is the CONTENTS statement. This section shows the output from the CONTENTS statement for the Health library and the Group data set, which is shown in the following output.

Only the items in the output that require explanation are discussed.

The Health Library Contents

The following is the list of all the files and catalogs in the Health library.

Directory	
Libref	HEALTH
Engine	V9
Physical Name	c:\procdatasets1\health
Filename	c:\procdatasets1\health
Owner Name	BUILTIN\Administrators
File Size	12KB
File Size (bytes)	12288

#	Name	Member Type	File Size	Last Modified
1	BODYFAT	DATA	8KB	09/03/2014 10:35:01
2	CONFOUND	DATA	8KB	09/03/2014 10:35:01
3	CORONARY	DATA	8KB	09/03/2014 10:35:01
4	FORMATS	CATALOG	17KB	11/16/2011 13:53:09
5	GROUP	DATA	32KB	09/03/2014 10:35:02
6	GRPOUT	DATA	144KB	01/15/2016 15:37:10
7	GRPOUT1	DATA	144KB	08/06/2015 09:54:32
8	INFANT	DATA	17KB	09/12/2007 10:57:52
9	MLSCL	DATA	8KB	09/03/2014 10:35:02
10	NAMES	DATA	8KB	09/03/2014 10:35:02
11	OXYGEN	DATA	16KB	09/03/2014 10:35:02
12	PERSONL	DATA	32KB	09/03/2014 10:35:02
13	PHARM	DATA	8KB	09/03/2014 10:35:02
14	POINTS	DATA	8KB	09/03/2014 10:35:02
15	POSTDRUG	CATALOG	61KB	11/16/2011 13:53:08
16	PRENAT	DATA	24KB	09/03/2014 10:35:02
17	RESULTS	DATA	8KB	09/03/2014 10:35:03
18	SLEEP	DATA	12KB	09/03/2014 10:35:03
19	SPDATA	VIEW	5KB	03/24/2005 13:12:22
20	SYNDROME	DATA	16KB	09/03/2014 10:35:03
21	TENSION	DATA	8KB	09/03/2014 10:35:03
22	TRAIN	DATA	8KB	09/03/2014 10:35:03
23	VISION	DATA	8KB	09/03/2014 10:35:03
24	WEIGHT	DATA	24KB	09/03/2014 10:35:03
25	WGHT	DATA	24KB	09/03/2014 10:35:03

Data Set Attributes

Here are descriptions of selected fields shown in the following output:

Member Type

is the type of library member (DATA or VIEW).

Created

indicates the date and time that the data set was created. The date and time reflect the setting of the TIMEZONE= system option. If the TIMEZONE= system option is not set, then the local time zone in which the Compute Server session is running is used.

Last Modified

indicates the date and time that the data set was last modified. The date and time reflect the setting of the TIMEZONE= system option. If the TIMEZONE= system option is not set, then the local time zone in which the Compute Server session is running is used.

Protection

indicates whether the SAS data set is Read, Write, or Alter password protected.

Data Set Type

names the special data set type (such as CORR, COV, SSPC, EST, or FACTOR), if any.

Observations

is the total number of observations currently in the file. Note that for a very large data set, if the number of observations exceeds the largest integer value that can be represented in a double precision floating point number, the count is shown as missing.

Deleted Observations

is the number of observations marked for deletion. These observations are not included in the total number of observations, shown in the **Observations** field. Note that for a very large data set, if the number of deleted observations exceeds the number that can be stored in a double-precision integer, the count is shown as missing. Also, the count for **Deleted Observations** shows a missing value if you use the COMPRESS=YES option with one or both of the REUSE=YES and POINTOBS=NO options.

Compressed

indicates whether the data set is compressed. If the data set is compressed, the output includes an additional item, **Reuse Space** (with a value of YES or NO). This item indicates whether to reuse space that is made available when observations are deleted.

Sorted

indicates whether the data set is sorted. If you sort the data set with PROC SORT, PROC SQL, or specify sort information with the SORTEDBY= data set option, a value of YES appears here, and there is an additional section to the output. See [“Sort Information” on page 481](#) for details.

Data Representation

is the format in which data is represented on a computer architecture or in an operating environment. For example, on an IBM PC, character data is represented by its ASCII encoding and byte-swapped integers. Native data representation refers to an environment for which the data representation compares with the CPU that is accessing the file.

Encoding

is the encoding value. Encoding is a set of characters (letters, logograms, digits, punctuation, symbols, control characters, and so on) that have been mapped to numeric values (called code points) that can be used by computers. The code points are assigned to the characters in the character set when you apply an encoding method.

Output 15.4 Attributes of the Class Data Set

Printing Class The CONTENTS Procedure	
Data Set Name	SASHELP.CLASS
Member Type	DATA
Engine	V9
Created	10/22/2019 09:38:47
Last Modified	10/22/2019 09:38:47
Protection	
Data Set Type	
Label	Student Data
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64, LINUX_POWER_6
Encoding	us-ascii ASCII (ANSI)

Engine and Operating Environment-Dependent Information

The CONTENTS statement produces operating environment-specific and engine-specific information. This information differs depending on the operating environment.

Output 15.5 Engine/Host Dependent Information for the Group Data Set

Engine/Host Dependent Information	
Data Set Page Size	65536
Number of Data Set Pages	1
First Data Page	1
Max Obs per Page	1632
Obs in First Data Page	19
Number of Data Set Repairs	0
Filename	/opt/sas/viya/home/SASFoundation/sashelp/class.sas7bdat
Release Created	V.0305M0
Host Created	Linux
Inode Number	70297014
Access Permission	rw-r--r--
Owner Name	sas
File Size	128KB
File Size (bytes)	131072

Alphabetic List of Variables and Attributes			
#	Variable	Type	Len
3	Age	Num	8
4	Height	Num	8
1	Name	Char	8
2	Sex	Char	1
5	Weight	Num	8

Alphabetic List of Variables and Attributes

Here are descriptions of selected columns in the following output:

#

is the logical position of each variable in the observation. This number is assigned to the variable when the variable is defined.

Variable

is the name of each variable. By default, variables appear alphabetically.

Note: Variable names are sorted such that X1, X2, and X10 appear in that order and not in the true collating sequence of X1, X10, and X2. Variable names that contain an underscore and digits might appear in a nonstandard sort order. For example, P25 and P75 appear before P2_5.

Type

specifies the type of variable: character or numeric.

Len

specifies the variable's length, which is the number of bytes used to store each of a variable's values in a SAS data set.

Transcode

specifies whether a character variable is transcoded. If the attribute is NO, then transcoding is suppressed. By default, character variables are transcoded when required. For more information about transcoding, see *SAS National Language Support (NLS): Reference Guide*.

Note: If none of the variables in the SAS data set has a format, informat, or label associated with it, or if all of the variables are set to TRANSCODE=YES, then the column for the attribute is NOT displayed.

Output 15.6 Listing of Variables and Attributes of the Group Data Set

Alphabetic List of Variables and Attributes					
#	Variable	Type	Len	Format	Informat
9	BIRTH	Num	8		
4	CITY	Char	15		
3	FNAME	Char	15		
10	HIRED	Num	8	DATE7.	DATE7.
11	HPHONE	Char	12		
1	IDNUM	Char	4		
7	JOBCODE	Char	4		
2	LNAME	Char	15		
8	SALARY	Num	8	COMMA8.	
6	SEX	Char	2		
5	STATE	Char	3		

Alphabetic List of Indexes and Attributes

The section shown in the following output appears only if the data set has indexes associated with it.

#

indicates the number of each index. The indexes are numbered sequentially as they are defined.

Index

displays the name of each index. For simple indexes, the name of the index is the same as a variable in the data set.

Unique Option

indicates whether the index must have unique values. If the column contains YES, the combination of values of the index variables is unique for each observation.

Nomiss Option

indicates whether the index excludes missing values for all index variables. If the column contains YES, the index does not contain observations with missing values for all index variables.

of Unique Values

gives the number of unique values in the index.

Variables

names the variables in a composite index.

Output 15.7 *Listing of Indexes and Attributes of the Group Data Set*

Alphabetic List of Indexes and Attributes					
#	Index	Unique Option	NoMiss Option	# of Unique Values	Variables
1	vital	YES	YES	148	BIRTH SALARY

Sort Information

The section shown in the following output appears only if the **sorted** field has a value of YES.

Sortedby

indicates how the data are currently sorted. This field contains either the variables and options that you use in the BY statement in PROC SORT, the column name in PROC SQL, or the values that you specify in the SORTEDBY= option.

Validated

indicates whether the data was sorted using PROC SORT or SORTEDBY. If PROC SORT or PROC SQL sorted the data set, the value is YES. If you assigned the sort indicator with the SORTEDBY= data set option, the value is NO.

Character Set

is the character set used to sort the data. The value for this field can be ASCII, EBCDIC, or PASCII.

Collating Sequence

is the collating sequence used to sort the data set, which can be a translation table name, an encoding value, or LINGUISTIC if the data set is sorted linguistically. This field does not appear if you do not specify a collating sequence that is different from the character set.

If the data set is sorted linguistically, additional linguistic collating sequence information appears after **Collating Sequence**, such as the locale, collation

style, and so on. For a list of the collation rules that can be specified for linguistic collation.

Sort Option

indicates whether PROC SORT used the NODUPKEY option when sorting the data set. This field does not appear if you did not use this option in a PROC SORT statement (not shown).

Output 15.8 Group Data Set Sort Information

Sort Information	
Sortedby	LNAME
Validated	NO
Character Set	ANSI

PROC DATASETS and the Output Delivery System (ODS)

Most SAS procedures send their messages to the SAS log and their procedure results to the output. PROC DATASETS is unique because it sends procedure results to both the SAS log and the procedure output file. When the interface to ODS was created, it was decided that all procedure results (from both the log and the procedure output file) should be available to ODS. In order to implement this feature and maintain compatibility with earlier releases, the interface to ODS had to be slightly different from the usual interface.

By default, the PROC DATASETS statement itself produces two output objects: Members and Directory. These objects are routed to the SAS log. The CONTENTS statement produces three output objects by default: Attributes, EngineHost, and Variables. (The use of various options adds other output objects.) These objects are routed to the procedure output file. If you open an ODS destination (such as HTML, RTF, or PRINTER), all of these objects are, by default, routed to that destination.

You can use ODS SELECT and ODS EXCLUDE statements to control which objects go to which destination, just as you can for any other procedure. However, because of the unique interface between PROC DATASETS and ODS, when you use the keyword LISTING in an ODS SELECT or ODS EXCLUDE statement, you affect both the log and the listing.

ODS Table Names

PROC DATASETS and PROC CONTENTS assign a name to each table that they create. You can use these names to reference the table when using the Output Delivery System (ODS) to select tables and create output data sets.

PROC CONTENTS generates the same ODS tables as PROC DATASETS with the CONTENTS statement.

Table 15.15 ODS Tables Produced by the DATASETS Procedure without the CONTENTS Statement

ODS Table	Description	Generates Table
Directory	General library information	Unless you specify the NOLIST option
Members	Library member information	Unless you specify the NOLIST option

Table 15.16 ODS Table Names Produced by PROC CONTENTS and PROC DATASETS with the CONTENTS Statement

ODS Table	Description	Generates Table
Attributes	Data set attributes	Unless you specify the SHORT option
Directory	General library information	If you specify DATA=<libref.>_ALL_ or the DIRECTORY option ¹
EngineHost	Engine and operating environment information	Unless you specify the SHORT option
IntegrityConstraints	A detailed listing of integrity constraints	If the data set has integrity constraints and you do not specify the SHORT option
IntegrityConstraintsShort	A concise listing of integrity constraints	If the data set has integrity constraints and you specify the SHORT option
Indexes	A detailed listing of indexes	If the data set is indexed and you do not specify the SHORT option
IndexesShort	A concise listing of indexes	If the data set is indexed and you specify the SHORT option

ODS Table	Description	Generates Table
Members	Library member information	If you specify DATA=<libref.>_ALL_ or the DIRECTORY option ¹
Position	A detailed listing of variables by logical position in the data set	If you specify the VARNUM option and you do not specify the SHORT option
PositionShort	A concise listing of variables by logical position in the data set	If you specify the VARNUM option and the SHORT option
PositionVarchar	Position including varchar type	If a varchar is in the data set, you specify VARNUM and you do not specify the SHORT option
Sortedby	Detailed sort information	If the data set is sorted and you do not specify the SHORT option
SortedbyShort	Concise Sort information	If the data set is sorted and you specify the SHORT option
Variables	A detailed listing of variables in alphabetical order	Unless you specify the SHORT option
VariablesShort	A concise listing of variables in alphabetical order	If you specify the SHORT option
VariablesVarchar	Variables including varchar type	If a varchar is in the data set, unless the SHORT option is specified

¹ For PROC DATASETS, if both the NOLIST option and either the DIRECTORY option or DATA=<libref.>_ALL_ are specified, then the NOLIST option is ignored.

Output Data Sets

The CONTENTS Statement

The CONTENTS statement is the only statement in the DATASETS procedure that generates output data sets.

The OUT= Data Set

The OUT= option in the CONTENTS statement creates an output data set. Each variable in each DATA= data set has one observation in the OUT= data set. Here are the variables in the output data set:

CHARSET

the character set used to sort the data set. The value is ASCII, EBCDIC, or PASCII. A blank appears if the data set does not have a sort indicator stored with it.

COLLATE

the collating sequence used to sort the data set. A blank appears if the sort indicator for the input data set does not include a collating sequence.

COMPRESS

indicates whether the data set is compressed.

CRDATE

date the data set was created.

DELOBS

number of observations marked for deletion in the data set. (Observations can be marked for deletion but not actually deleted when you use the FSEDIT procedure of SAS/FSP software.)

ENCRYPT

indicates whether the data set is encrypted.

ENGINE

name of the method used to read from and write to the data set.

FLAGS

indicates whether the variables in an SQL view are protected (**P**) or contribute (**C**) to a derived variable.

P

indicates the variable is protected. The value of the variable can be displayed but not updated.

C

indicates whether the variable contributes to a derived variable.

The value of FLAG is blank if **P** or **C** does not apply to an SQL view or if it is a data set view.

FORMAT

variable format. The value of FORMAT is a blank if you do not associate a format with the variable.

FORMATD

number of decimals that you specify when you associate the format with the variable. The value of FORMATD is 0 if you do not specify decimals in the format.

FORMATL

format length. If you specify a length for the format when you associate the format with a variable, the length that you specify is the value of **FORMATL**. If you do not specify a length for the format when you associate the format with a variable, the value of **FORMATL** is the default length of the format if you use the **FMTLEN** option and 0 if you do not use the **FMTLEN** option.

GENMAX

maximum number of versions for the generation group.

GENNEXT

the next generation number for a generation group.

GENNUM

the version number.

IDXCOUNT

number of indexes for the data set.

IDXUSAGE

use of the variable in indexes. Possible values are

NONE

the variable is not part of an index.

SIMPLE

the variable has a simple index. No other variables are included in the index.

COMPOSITE

the variable is part of a composite index.

BOTH

the variable has a simple index and is part of a composite index.

INFORMAT

variable informat. The value is a blank if you do not associate an informat with the variable.

INFORMD

number of decimals that you specify when you associate the informat with the variable. The value is 0 if you do not specify decimals when you associate the informat with the variable.

INFORML

informat length. If you specify a length for the informat when you associate the informat with a variable, the length that you specify is the value of **INFORML**. If you do not specify a length for the informat when you associate the informat with a variable, the value of **INFORML** is the default length of the informat if you use the **FMTLEN** option and 0 if you do not use the **FMTLEN** option.

JUST

justification (0=left, 1=right).

LABEL

variable label (blank if none given).

LENGTH

variable length.

LIBNAME

libref used for the data library.

MEMLABEL

label for this SAS data set (blank if no label).

MEMNAME

SAS data set that contains the variable.

MEMTYPE

library member type (DATA or VIEW).

MODATE

date the data set was last modified.

NAME

variable name.

NOBS

number of observations in the data set.

NODUPKEY

indicates whether the NODUPKEY option was used in a PROC SORT statement to sort the input data set.

NPOS

physical position of the first character of the variable in the data set.

POINTOBS

indicates whether the data set can be addressed by observation.

PROTECT

the first letter of the level of protection. The value for PROTECT is one or more of the following:

A

indicates the data set is alter-protected.

R

indicates the data set is read-protected.

W

indicates the data set is write-protected.

REUSE

indicates whether the space made available when observations are deleted from a compressed data set should be reused. If the data set is not compressed, the REUSE variable has a value of NO.

SORTED

the value depends on the sorting characteristics of the input data set. Here are the possible values:

. (period)

for not sorted.

0

for sorted but not validated.

1

for sorted and validated.

SORTEDBY

the value depends on that variable's role in the sort. Here are the possible values:

. (period)
if the variable was not used to sort the input data set.

n
where n is an integer that denotes the position of that variable in the sort. A negative value of n indicates that the data set is sorted by the descending order of that variable.

TRANSCOD
indicates whether the variable is transcoded.

TYPE
type of the variable (1=numeric, 2=character).

TYPEMEM
special data set type (blank if no TYPE= value is specified).

VARNUM
variable number in the data set. Variables are numbered in the order in which they appear.

The output data set is sorted by the variables LIBNAME and MEMNAME.

Note: The variable names are sorted so that the values X1, X2, and X10 are listed in that order, not in the true collating sequence of X1, X10, X2. Therefore, if you want to use a BY statement on MEMNAME in subsequent steps, run a PROC SORT step on the output data set first. You can also use the NOTSORTED option in the BY statement.

Here is an example of an output data set created from the Group data set, which is shown in [“Example 5: Describing a SAS Data Set” on page 508](#) and in [“Procedure Output” on page 475](#).

Due to the size of the Health.Grpout, the following output is in five sections.

Output 15.9 An Example of an Output Data Set – Section 1

Obs	LIBNAME	MEMNAME	MEMLABEL	TYPOMEM	NAME	TYPE	LENGTH	VARNUM
1	HEALTH	GROUP			BIRTH	1	8	9
2	HEALTH	GROUP			CITY	2	15	4
3	HEALTH	GROUP			FNAME	2	15	3
4	HEALTH	GROUP			HIRED	1	8	10
5	HEALTH	GROUP			HPHONE	2	12	11
6	HEALTH	GROUP			IDNUM	2	4	1
7	HEALTH	GROUP			JOBCODE	2	4	7
8	HEALTH	GROUP			LNAME	2	15	2
9	HEALTH	GROUP			SALARY	1	8	8
10	HEALTH	GROUP			SEX	2	2	6
11	HEALTH	GROUP			STATE	2	3	5

Output 15.10 An Example of an Output Data Set – Section 2

LABEL	FORMAT	FORMATL	FORMATD	INFORMAT	INFORML	INFORMD
		0	0		0	0
		0	0		0	0
		0	0		0	0
	DATE	7	0	DATE	7	0
		0	0		0	0
		0	0		0	0
		0	0		0	0
		0	0		0	0
	COMMA	8	0		0	0
		0	0		0	0
		0	0		0	0

Output 15.11 *An Example of an Output Data Set – Section 3*

JUST	NPOS	NOBS	ENGINE	CRDATE	MODATE	DELOBS
1	8	148	V9	03SEP14:10:35:02	03SEP14:10:35:02	0
0	58	148	V9	03SEP14:10:35:02	03SEP14:10:35:02	0
0	43	148	V9	03SEP14:10:35:02	03SEP14:10:35:02	0
1	16	148	V9	03SEP14:10:35:02	03SEP14:10:35:02	0
0	82	148	V9	03SEP14:10:35:02	03SEP14:10:35:02	0
0	24	148	V9	03SEP14:10:35:02	03SEP14:10:35:02	0
0	78	148	V9	03SEP14:10:35:02	03SEP14:10:35:02	0
0	28	148	V9	03SEP14:10:35:02	03SEP14:10:35:02	0
1	0	148	V9	03SEP14:10:35:02	03SEP14:10:35:02	0
0	76	148	V9	03SEP14:10:35:02	03SEP14:10:35:02	0
0	73	148	V9	03SEP14:10:35:02	03SEP14:10:35:02	0

Output 15.12 *An Example of an Output Data Set – Section 4*

[illegible]

Output 15.13 An Example of an Output Data Set — Section 5

CHARSET	COLLATE	NODUPKEY	NODUPREC	ENCRYPT	POINTOBS	GENMAX	GENNUM	GENNEXT	TRANSCOD
		NO	NO	NO	YES	0	.	.	YES
		NO	NO	NO	YES	0	.	.	YES
		NO	NO	NO	YES	0	.	.	YES
		NO	NO	NO	YES	0	.	.	YES
		NO	NO	NO	YES	0	.	.	YES
		NO	NO	NO	YES	0	.	.	YES
		NO	NO	NO	YES	0	.	.	YES
		NO	NO	NO	YES	0	.	.	YES
		NO	NO	NO	YES	0	.	.	YES
		NO	NO	NO	YES	0	.	.	YES
		NO	NO	NO	YES	0	.	.	YES

The OUT2= Data Set

The OUT2= option in the CONTENTS statement creates an output data set that contains information about indexes and integrity constraints. Here are the variables in the output data set:

IC_OWN

contains YES if the index is owned by the integrity constraint.

INACTIVE

contains YES if the integrity constraint is inactive.

LIBNAME

libref used for the data library.

MEMNAME

SAS data set that contains the variable.

MG

the value of MESSAGE=, if it is used, in the IC CREATE statement.

MSGTYPE

the value is blank unless an integrity constraint is violated and you specified a message.

NAME

the name of the index or integrity constraint.

NOMISS

contains YES if the NOMISS option is defined for the index.

NUMVALS

the number of distinct values in the index (displayed for centiles).

NUMVARS

the number of variables involved in the index or integrity constraint.

ONDELETE

for a foreign key integrity constraint, contains RESTRICT or SET NULL if applicable (the ON DELETE option in the IC CREATE statement).

ONUPDATE

for a foreign key integrity constraint, contains RESTRICT or SET NULL if applicable (the ON UPDATE option in the IC CREATE statement).

RECREATE

the SAS statement necessary to re-create the index or integrity constraint.

REFERENCE

for a foreign key integrity constraint, contains the name of the referenced data set.

TYPE

the type. For an index, the value is "Index" while for an integrity constraint, the value is the type of integrity constraint (Not Null, Check, Primary Key, and so on).

UNIQUE

contains YES if the UNIQUE option is defined for the index.

UPERC

the percentage of the index that has been updated since the last refresh (displayed for centiles).

UPERCMX

the percentage of the index update that triggers a refresh (displayed for centiles).

WHERE

for a check integrity constraint, contains the WHERE statement.

Examples: DATASETS Procedure

Example 1: Removing All Labels and Formats in a Data Set

Features:

PROC DATASETS statement options

LIB=

MEMTYPE=

CONTENTS statement

MODIFY statement

CONTENTS procedure

FORMAT procedure

OPTIONS statement

TITLE statement

Details

This example demonstrates the following tasks:

- sets system options
- creates a user defined FORMAT
- creates a data set
- deletes labels and format from the data set
- uses PROC CONTENTS to show data set with and without labels and format

Program

```
options ls=79 nodate center;
title ;

libname mylib '/home/userid/mylib';

proc format;
  value clsfmt 1='Freshman' 2='Sophomore' 3='Junior' 4='Senior';
run;

data mylib.class;
  format z clsfmt.;
  label x='ID NUMBER'
        y='AGE'
        z='CLASS STATUS';
  input x y z;
datalines;
1 20 4
2 18 1
;

proc contents data=mylib.class;
run;

proc datasets lib=mylib memtype=data;
  modify class;
  attrib _all_ label=' ';
  attrib _all_ format=;
  contents data=mylib.class;
run;
quit;
```

Program Description

Set the system options and the LIBNAME statement. In this example, the LIBNAME is MyLib. The CENTER option specifies to align SAS procedure output in the center. The NODATE option specifies that the date and the time are not printed. The LS= option specifies the line size for the SAS log and for the output. The TITLE statement followed by a blank space removes any existing title in your Compute Server session.

```
options ls=79 nodate center;
title ;

libname mylib '/home/userid/mylib';
```

Create a user-defined format with a value of CLSFMT.

```
proc format;
  value clsfmt 1='Freshman' 2='Sophomore' 3='Junior' 4='Senior';
run;
```

Create a data set named Class. Use the CLSFMT format on variable Z. Create labels for variables, X, Y, and Z.

```
data mylib.class;
  format z clsfmt.;
  label x='ID NUMBER'
        y='AGE'
        z='CLASS STATUS';
  input x y z;
datalines;
1 20 4
2 18 1
;
```

Use PROC CONTENTS to view the contents of the data set before removing the labels and format.

```
proc contents data=mylib.class;
run;
```

Within PROC DATASETS, remove all the labels and formats using the MODIFY statement and the ATTRIB option. Use the CONTENTS statement within PROC DATASETS to view the contents of the data set without the labels and format.

```
proc datasets lib=mylib memtype=data;
  modify class;
  attrib _all_ label=' ';
  attrib _all_ format=;
  contents data=mylib.class;
run;
quit;
```

Output Examples

Output 15.14 CONTENTS Procedure for Class Data Set with Labels and Format

The CONTENTS Procedure

Data Set Name	MYLIB.CLASS	Observations	2
Member Type	DATA	Variables	3
Engine	V9	Indexes	0
Created	08/16/2016 14:48:34	Observation Length	24
Last Modified	08/16/2016 14:48:34	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	WINDOWS_64		
Encoding	wlatin1 Western (Windows)		

Engine/Host Dependent Information

Data Set Page Size	65536
Number of Data Set Pages	1
First Data Page	1
Max Obs per Page	2715
Obs in First Data Page	2
Number of Data Set Repairs	0
ExtendObsCounter	YES
Filename	c:\mylib\class.sas7bdat
Release Created	9.0401M4
Host Created	X64_7PRO
Owner Name	BUILTIN\Administrators
File Size	128KB
File Size (bytes)	131072

Alphabetic List of Variables and Attributes					
#	Variable	Type	Len	Format	Label
2	x	Num	8		ID NUMBER
3	y	Num	8		AGE
1	z	Num	8	CLSFMT.	CLASS STATUS

Output 15.15 CONTENTS Statement for Class Data Set without Labels and Format

The DATASETS Procedure

Data Set Name	MYLIB.CLASS	Observations	2
Member Type	DATA	Variables	3
Engine	V9	Indexes	0
Created	08/16/2016 14:48:34	Observation Length	24
Last Modified	08/16/2016 14:48:34	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	WINDOWS_64		
Encoding	wlatin1 Western (Windows)		

Engine/Host Dependent Information	
Data Set Page Size	65536
Number of Data Set Pages	1
First Data Page	1
Max Obs per Page	2715
Obs in First Data Page	2
Number of Data Set Repairs	0
ExtendObsCounter	YES
Filename	c:\mylib\class.sas7bdat
Release Created	9.0401M4
Host Created	X64_7PRO
Owner Name	BUILTIN\Administrators
File Size	128KB
File Size (bytes)	131072

Alphabetic List of Variables and Attributes			
#	Variable	Type	Len
2	x	Num	8
3	y	Num	8
1	z	Num	8

Example 2: Manipulating SAS Files

Features:

- PROC DATASETS statement options
 - DETAILS
 - LIBRARY=
 - COPY statement
 - CHANGE statement
 - DELETE statement
 - EXCHANGE statement
- COPY procedure
- EXCLUDE statement
- OPTIONS statement

Details

This example demonstrates the following tasks:

- changes the names of SAS files
- copies SAS files between SAS libraries
- deletes SAS files
- selects SAS files to copy
- exchanges the names of SAS files
- excludes SAS files from a copy operation

Program

```
options pagesize=60 linesize=80 nodate pageno=1 source;

LIBNAME dest1 'SAS-library-1';
LIBNAME dest2 'SAS-library-2';
LIBNAME health 'SAS-library-3';

proc datasets library=health details;
```

```

delete tension a2(mt=catalog);
change a1=postdrug;
exchange weight=bodyfat;

copy out=dest1 move memtype=view;

select spdata;

select etest1-etest5 / memtype=catalog;

copy out=dest2;
exclude d: mlscl oxygen test2 vision weight;
quit;

```

Program Description

Set the system options. The SOURCE system option writes the programming statements to the SAS log. PAGESIZE= option specifies the number of lines that compose a page of the SAS log and SAS output. LINESIZE= option specifies the line size for the SAS log and for SAS procedure output. NODATE option specifies that the date and the time are not printed. PAGENO= option specifies a beginning page number for the next page of output.

```

options pagesize=60 linesize=80 nodate pageno=1 source;

LIBNAME dest1 'SAS-library-1';
LIBNAME dest2 'SAS-library-2';
LIBNAME health 'SAS-library-3';

```

Specify the procedure input library, and add more details to the directory.

DETAILS prints these additional columns in the directory: **Obs**, **Entries** or **Indexes**, **Vars**, and **Label**. All member types are available for processing because the MEMTYPE= option does not appear in the PROC DATASETS statement.

```
proc datasets library=health details;
```

Delete two files in the library, and modify the names of a SAS data set and a catalog. The DELETE statement deletes the Tension data set and the A2 catalog. MT=CATALOG applies only to A2 and is necessary because the default member type for the DELETE statement is DATA. The CHANGE statement changes the name of the A1 catalog to Postdrug. The EXCHANGE statement exchanges the names of the Weight and Bodyfat data sets. MEMTYPE= is not necessary in the CHANGE or EXCHANGE statement because the default is MEMTYPE=ALL for each statement.

```

delete tension a2(mt=catalog);
change a1=postdrug;
exchange weight=bodyfat;

```

Restrict processing to one member type and delete and move data views.

MEMTYPE=VIEW restricts processing to SAS views. MOVE specifies that all SAS views named in the SELECT statements in this step be deleted from the Health data library and moved to the Dest1 data library.

```
copy out=dest1 move memtype=view;
```

Move the SAS view Spdata from the Health data library to the Dest1 data library.

```
select spdata;
```


Move the catalogs to another data library. The SELECT statement specifies that the catalogs Etest1 through Etest5 be moved from the Health data library to the Dest1 data library. MEMTYPE=CATALOG overrides the MEMTYPE=VIEW option in the COPY statement.

```
select etest1-etest5 / memtype=catalog;
```

Exclude all files with specified criteria from processing. The EXCLUDE statement excludes from the COPY operation all SAS files that begin with the letter D and the other SAS files listed. All remaining SAS files in the Health data library are copied to the Dest2 data library.

```
copy out=dest2;  
  exclude d: mlscl oxygen test2 vision weight;  
quit;
```

Log Examples

Example Code 15.2 SAS Log for Dest1

```

117  options pagesize=60 linesize=80 nodate pageno=1 source;
118  LIBNAME dest1 'SAS-library-1';
NOTE: Libref DEST1 was successfully assigned as follows:
      Engine:          V9
      Physical Name:   SAS-library-1\dest1
119  LIBNAME dest2 'SAS-library-2';
NOTE: Libref DEST2 was successfully assigned as follows:
      Engine:          V9
      Physical Name:   SAS-library-2\dest2
120  LIBNAME health 'SAS-library-3';
NOTE: Libref HEALTH was successfully assigned as follows:
      Engine:          V9
      Physical Name:   SAS-library-3\health

121  proc datasets library=health details;
                                Directory

Libref      HEALTH
Engine      V9
Physical Name \myfiles\health
Filename    \myfiles\health

#  Name      Member  Obs, Entries
      Type      or Indexes  Vars  Label

 1  A1        CATALOG    23
 2  ALL        DATA      23      17
 3  BODYFAT    DATA      1       2
 4  CONFOUND   DATA      8       4
 5  CORONARY   DATA     39      4
 6  DRUG1      DATA      6       2  JAN2005 DATA
 7  DRUG2      DATA     13      2  MAY2005 DATA
 8  DRUG3      DATA     11      2  JUL2005 DATA
 9  DRUG4      DATA      7       2  JAN2002 DATA
10  DRUG5      DATA      1       2  JUL2002 DATA
11  ETEST1     CATALOG    1
12  ETEST2     CATALOG    1
13  ETEST3     CATALOG    1
14  ETEST4     CATALOG    1
15  ETEST5     CATALOG    1
16  ETESTS     CATALOG    1
17  FORMATS    CATALOG    6
18  GROUP      DATA     148     11
19  GRPOUT     DATA     11      40
20  INFANT     DATA     149      6
21  MLSCL      DATA     32      4  Multiple Sclerosis Data
22  NAMES      DATA      7       4
23  OXYGEN     DATA     31      7
24  PERSONL    DATA     148     11
25  PHARM      DATA      6       3  Sugar Study
26  POINTS     DATA      6       6
27  RESULTS    DATA     10      5
28  SLEEP      DATA    108      6
29  SPDATA     VIEW        .       2
30  TEST2      DATA     15      5
31  TRAIN      DATA      7       2
32  VISION     DATA     16      3
33  WEIGHT     DATA     83     13  California Results
34  WGHT       DATA     83     13

```

#	File	Size	Last Modified
1	62464	07Mar05:14:36:20	
2	13312	12Sep07:13:57:48	
3	5120	12Sep07:13:57:48	
4	5120	12Sep07:13:57:48	
5	5120	12Sep07:13:57:48	
6	5120	12Sep07:13:57:49	
7	5120	12Sep07:13:57:49	
8	5120	12Sep07:13:57:49	
9	5120	12Sep07:13:57:49	
10	5120	12Sep07:13:57:49	
11	17408	04Jan02:14:20:16	
12	17408	04Jan02:14:20:16	
13	17408	04Jan02:14:20:16	
14	17408	04Jan02:14:20:16	
15	17408	04Jan02:14:20:16	
16	17408	24Mar05:16:12:20	
17	17408	24Mar05:16:12:20	
18	25600	12Sep07:13:57:50	
19	17408	24Mar05:15:33:31	
20	17408	12Sep07:13:57:51	
21	5120	12Sep07:13:57:50	
22	5120	12Sep07:13:57:50	
23	9216	12Sep07:13:57:50	
24	25600	12Sep07:13:57:51	
25	5120	12Sep07:13:57:51	
26	5120	12Sep07:13:57:51	
27	5120	12Sep07:13:57:52	
28	9216	12Sep07:13:57:52	
29	5120	24Mar05:16:12:21	
30	5120	12Sep07:13:57:52	
31	5120	12Sep07:13:57:53	
32	5120	12Sep07:13:57:53	
33	13312	12Sep07:13:57:53	
34	13312	12Sep07:13:57:53122	delete tension

```

a2(mt=catalog);
123  change a1=postdrug;
124  exchange weight=bodyfat;
NOTE: Changing the name HEALTH.A1 to HEALTH.POSTDRUG (memtype=CATALOG).
NOTE: Exchanging the names HEALTH.WEIGHT and HEALTH.BODYFAT (memtype=DATA).
125  copy out=dest1 move memtype=view;
126  select spdata;
127
128  select etest1-etest5 / memtype=catalog;
NOTE: Moving HEALTH.SPDATA to DEST1.SPDATA (memtype=VIEW).
NOTE: Moving HEALTH.ETEST1 to DEST1.ETEST1 (memtype=CATALOG).
NOTE: Moving HEALTH.ETEST2 to DEST1.ETEST2 (memtype=CATALOG).
NOTE: Moving HEALTH.ETEST3 to DEST1.ETEST3 (memtype=CATALOG).
NOTE: Moving HEALTH.ETEST4 to DEST1.ETEST4 (memtype=CATALOG).
NOTE: Moving HEALTH.ETEST5 to DEST1.ETEST5 (memtype=CATALOG).

```

```

129 copy out=dest2;
130 exclude d: mlscl oxygen test2 vision weight;
131 quit;

```

NOTE: Copying HEALTH.ALL to DEST2.ALL (memtype=DATA).

NOTE: There were 23 observations read from the data set HEALTH.ALL.

NOTE: The data set DEST2.ALL has 23 observations and 17 variables.

NOTE: Copying HEALTH.BODYFAT to DEST2.BODYFAT (memtype=DATA).

NOTE: There were 83 observations read from the data set HEALTH.BODYFAT.

NOTE: The data set DEST2.BODYFAT has 83 observations and 13 variables.

NOTE: Copying HEALTH.CONFOUND to DEST2.CONFOUND (memtype=DATA).

NOTE: There were 8 observations read from the data set HEALTH.CONFOUND.

NOTE: The data set DEST2.CONFOUND has 8 observations and 4 variables.

NOTE: Copying HEALTH.CORONARY to DEST2.CORONARY (memtype=DATA).

NOTE: There were 39 observations read from the data set HEALTH.CORONARY.

NOTE: The data set DEST2.CORONARY has 39 observations and 4 variables.

NOTE: Copying HEALTH.ETESTS to DEST2.ETESTS (memtype=CATALOG).

NOTE: Copying HEALTH.FORMATS to DEST2.FORMATS (memtype=CATALOG).

NOTE: Copying HEALTH.GROUP to DEST2.GROUP (memtype=DATA).

NOTE: There were 148 observations read from the data set HEALTH.GROUP.

NOTE: The data set DEST2.GROUP has 148 observations and 11 variables.

NOTE: Copying HEALTH.GRPOUT to DEST2.GRPOUT (memtype=DATA).

NOTE: There were 11 observations read from the data set HEALTH.GRPOUT.

NOTE: The data set DEST2.GRPOUT has 11 observations and 40 variables.

NOTE: Copying HEALTH.INFANT to DEST2.INFANT (memtype=DATA).

NOTE: There were 149 observations read from the data set HEALTH.INFANT.

NOTE: The data set DEST2.INFANT has 149 observations and 6 variables.

NOTE: Copying HEALTH.NAMES to DEST2.NAMES (memtype=DATA).

NOTE: There were 7 observations read from the data set HEALTH.NAMES.

NOTE: The data set DEST2.NAMES has 7 observations and 4 variables.

NOTE: Copying HEALTH.PERSONL to DEST2.PERSONL (memtype=DATA).

NOTE: There were 148 observations read from the data set HEALTH.PERSONL.

NOTE: The data set DEST2.PERSONL has 148 observations and 11 variables.

NOTE: Copying HEALTH.PHARM to DEST2.PHARM (memtype=DATA).

NOTE: There were 6 observations read from the data set HEALTH.PHARM.

NOTE: The data set DEST2.PHARM has 6 observations and 3 variables.

NOTE: Copying HEALTH.POINTS to DEST2.POINTS (memtype=DATA).

NOTE: There were 6 observations read from the data set HEALTH.POINTS.

NOTE: The data set DEST2.POINTS has 6 observations and 6 variables.

NOTE: Copying HEALTH.POSTDRUG to DEST2.POSTDRUG (memtype=CATALOG).

NOTE: Copying HEALTH.RESULTS to DEST2.RESULTS (memtype=DATA).

NOTE: There were 10 observations read from the data set HEALTH.RESULTS.

NOTE: The data set DEST2.RESULTS has 10 observations and 5 variables.

NOTE: Copying HEALTH.SLEEP to DEST2.SLEEP (memtype=DATA).

NOTE: There were 108 observations read from the data set HEALTH.SLEEP.

NOTE: The data set DEST2.SLEEP has 108 observations and 6 variables.

NOTE: Copying HEALTH.TRAIN to DEST2.TRAIN (memtype=DATA).

NOTE: There were 7 observations read from the data set HEALTH.TRAIN.

NOTE: The data set DEST2.TRAIN has 7 observations and 2 variables.

NOTE: Copying HEALTH.WGHT to DEST2.WGHT (memtype=DATA).

NOTE: There were 83 observations read from the data set HEALTH.WGHT.

NOTE: The data set DEST2.WGHT has 83 observations and 13 variables.

NOTE: PROCEDURE DATASETS used (Total process time):

real time	44.04 seconds
cpu time	0.60 seconds

Example 3: Saving SAS Files from Deletion

Features: PROC DATASETS statement options
LIB=

SAVE statement
OPTIONS statement

Details

This example demonstrates how to use the SAVE statement to save some SAS files from deletion and to delete other SAS files.

Program

```
options pagesize=40 linesize=80 nodate pageno=1 source;  
LIBNAME elder 'SAS-library';  
proc datasets lib=elder;  
    save chronic aging clinics / memtype=data;  
run;
```

Program Description

Set the system options and the LIBNAME statement. The SOURCE system option writes the programming statements to the SAS log. LINESIZE= option specifies the line size for the SAS log and for SAS procedure output. NODATE option specifies that the date and the time are not printed.

```
options pagesize=40 linesize=80 nodate pageno=1 source;  
LIBNAME elder 'SAS-library';
```

Specify the procedure input library to process.

```
proc datasets lib=elder;
```

Save the data sets Chronic, Aging, and Clinics, and delete all other SAS files (of all types) in the Elder library. MEMTYPE=DATA is necessary because the Elder library has a catalog named Clinics and a data set named Clinics.

```
    save chronic aging clinics / memtype=data;  
run;
```

Log Examples

Example Code 15.3 SAS Log for Elder Library

```

6  options pagesize=40 linesize=80 nodate pageno=1 source;
7  LIBNAME elder '/home/userid/procdatasets/elder';
NOTE: Libref ELDER was successfully assigned as follows:
      Engine:          V9
      Physical Name: /home/userid/procdatasets/elder
8  proc datasets lib=elder;
9      save chronic aging clinics / memtype=data;
10 run;

NOTE: Saving ELDER.CHRONIC (memtype=DATA).
NOTE: Saving ELDER.AGING (memtype=DATA).
NOTE: Saving ELDER.CLINICS (memtype=DATA).
11 quit;

NOTE: PROCEDURE DATASETS used (Total process time):
      real time          0.06 seconds
      cpu time           0.01 seconds

```

Output 15.16 Elder Library before and After Using the SAVE Statement

Directory	
Libref	ELDER
Engine	V9
Physical Name	c:\procdatasets\elder
Filename	c:\procdatasets\elder
Owner Name	BUILTIN\Administrators
File Size	4KB
File Size (bytes)	4096

#	Name	Member Type	File Size	Last Modified
1	AGING	DATA	8KB	04/17/2014 12:59:39
2	CHRONIC	DATA	8KB	04/17/2014 12:59:39
3	CLINICS	DATA	8KB	04/17/2014 12:59:39

Directory	
Libref	ELDER
Engine	V9
Physical Name	c:\procdatasets\elder
Filename	c:\procdatasets\elder
Owner Name	BUILTIN\Administrators
File Size	4KB
File Size (bytes)	4096

#	Name	Member Type	File Size	Last Modified
1	AGING	DATA	8KB	04/17/2014 12:59:39
2	CHRONIC	DATA	8KB	04/17/2014 12:59:39
3	CLINICS	DATA	8KB	04/17/2014 12:59:39

Example 4: Modifying SAS Data Sets

Features:

- PROC DATASETS statement options
- LIB=
- NOLIST
- MODIFY statement
- OPTIONS statement

Details

This example modifies two SAS data sets using the MODIFY statement and statements subordinate to it. [“Example 5: Describing a SAS Data Set” on page 508](#) shows the modifications to the Group data set.

This example demonstrates the following tasks:

- modifying SAS files
- labeling a SAS data set
- adding a Read password to a SAS data set
- indicating how a SAS data set is currently sorted
- creating an index for a SAS data set
- assigning informats and formats to variables in a SAS data set
- renaming variables in a SAS data set

- labeling variables in a SAS data set

Program

```
options pagesize=40 linesize=80 nodate pageno=1 source;
LIBNAME health 'SAS-library';
proc datasets library=health nolist;
    modify group (label='Test Subjects' read=green sortedby=lname);
        index create vital=(birth salary) / nomiss unique;
        informat birth date7.;
        format birth date7.;
        label salary='current salary excluding bonus';
    modify oxygen;
        rename oxygen=intake;
        label intake='Intake Measurement';
quit;
```

Program Description

Set the system options and LIBNAME statement. The SOURCE system option writes the programming statements to the SAS log. PAGESIZE= option specifies the number of lines that compose a page of the SAS log and SAS output. LINESIZE= option specifies the line size for the SAS log and for SAS procedure output. NODATE option specifies that the date and the time are not printed. PAGENO= option specifies a beginning page number for the next page of output.

```
options pagesize=40 linesize=80 nodate pageno=1 source;
LIBNAME health 'SAS-library';
```

Specify Health as the procedure input library to process. NOLIST suppresses the directory listing for the Health data library.

```
proc datasets library=health nolist;
```

Add a label to a data set, assign a Read password, and specify how to sort the data. LABEL= adds a data set label to the data set Group. READ= assigns green as the Read password. The password appears as Xs in the SAS log. SAS issues a warning message if you specify a level of password protection on a SAS file that does not include alter protection. SORTEDBY= specifies how the data is sorted.

```
    modify group (label='Test Subjects' read=green sortedby=lname);
```

Create the composite index VITAL on the variables BIRTH and SALARY for the Group data set. NOMISS excludes all observations that have missing values for BIRTH and SALARY from the index. UNIQUE specifies that the index is created only if each observation has a unique combination of values for BIRTH and SALARY.


```
index create vital=(birth salary) / nomiss unique;
```

Assign an informat and format, respectively, to the BIRTH variable.

```
informat birth date7.;
format birth date7.;
```

Assign a label to the variable SALARY.

```
label salary='current salary excluding bonus';
```

Rename a variable, and assign a label. Modify the data set Oxygen by renaming the variable OXYGEN to INTAKE and assigning a label to the variable INTAKE.

```
modify oxygen;
rename oxygen=intake;
label intake='Intake Measurement';
quit;
```

Log Examples

Example Code 15.4 SAS Log for Health Library

```
169 options pagesize=40 linesize=80 nodate pageno=1 source;
170 LIBNAME health 'SAS-library';
NOTE: Libref HEALTH was successfully assigned as follows:
      Engine:          V9
      Physical Name:   SAS-library\health

NOTE: PROCEDURE DATASETS used (Total process time):
      real time          8:06.11
      cpu time           0.54 seconds

171 proc datasets library=health nolist;
172   modify group (label='Test Subjects' read=XXXXX sortedby=lname);
WARNING: The file HEALTH.GROUP.DATA is not ALTER protected.  It could be
        deleted or replaced without knowing the password.
173   index create vital=(birth salary) / nomiss unique;
NOTE: Composite index vital has been defined.
NOTE: MODIFY was successful for HEALTH.GROUP.DATA.
174   informat birth date7.;
175   format birth date7.;
176   label salary='current salary excluding bonus';
177   modify oxygen;
178   rename oxygen=intake;
NOTE: Renaming variable oxygen to intake.
179   label intake='Intake Measurement';
180   quit;

NOTE: MODIFY was successful for HEALTH.OXYGEN.DATA.
NOTE: PROCEDURE DATASETS used (Total process time):
      real time          15.09 seconds
      cpu time           0.06 seconds
```

Example 5: Describing a SAS Data Set

Features: PROC DATASETS statement options
LIB=
NOLIST
CONTENTS statement
OPTIONS statement

Details

This example demonstrates the output from the CONTENTS statement for the Group data set. The output shows the modifications made to the Group data set in [“Example 4: Modifying SAS Data Sets” on page 505](#).

Program

```
options pagesize=40 linesize=80 nodate pageno=1;

LIBNAME health 'SAS-library';

proc datasets library=health nolist;

    contents data=group (read=green) out=grpout;
    title  'The Contents of the GROUP Data Set';
run;
quit;
```

Program Description

Set the system options and LIBNAME statement. PAGESIZE= option specifies the number of lines that compose a page of the SAS log and SAS output. LINESIZE= option specifies the line size for the SAS log and for SAS procedure output. NODATE option specifies that the date and the time are not printed. PAGENO= option specifies a beginning page number for the next page of output.

```
options pagesize=40 linesize=80 nodate pageno=1;

LIBNAME health 'SAS-library';
```

Specify Health as the procedure input library, and suppress the directory listing with the NOLIST option.

```
proc datasets library=health nolist;
```

Create the output data set Grpout from the data set Group. Specify Group as the data set to describe, give Read access to the Group data set, and create the output data set Grpout, which appears in the OUT= data set.

```
contents data=group (read=green) out=grpout;
title 'The Contents of the GROUP Data Set';
run;
quit;
```

Output Examples

Output 15.17 Contents of Group Data Set

The Contents of the GROUP Data Set			
The DATASETS Procedure			
Data Set Name	HEALTH.GROUP	Observations	148
Member Type	DATA	Variables	11
Engine	V9	Indexes	0
Created	09/03/2014 10:35:02	Observation Length	96
Last Modified	09/03/2014 10:35:02	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	WINDOWS_32		
Encoding	wlatin1 Western (Windows)		

Output 15.18 Engine Host Dependent Information

Engine/Host Dependent Information	
Data Set Page Size	8192
Number of Data Set Pages	3
First Data Page	1
Max Obs per Page	84
Obs in First Data Page	63
Number of Data Set Repairs	0
ExtendObsCounter	YES
Filename	c:\procdatasets\health\group.sas7bdat
Release Created	9.0401M3
Host Created	W32_7PRO
Owner Name	BUILTIN\Administrators
File Size	32KB
File Size (bytes)	32768

Output 15.19 Alphabetic List of Variables and Attributes

Alphabetic List of Variables and Attributes					
#	Variable	Type	Len	Format	Informat
9	BIRTH	Num	8		
4	CITY	Char	15		
3	FNAME	Char	15		
10	HIRED	Num	8	DATE7.	DATE7.
11	HPHONE	Char	12		
1	IDNUM	Char	4		
7	JOBCODE	Char	4		
2	LNAME	Char	15		
8	SALARY	Num	8	COMMA8.	
6	SEX	Char	2		
5	STATE	Char	3		

Output 15.20 *Alphabetic List of Extended Attributes for the Data Set and Variables*

Alphabetic List of Data Set Extended Attributes		
Extended Attribute	Numeric Value	Character Value
level	1	
region	.	NorthEast

Alphabetic List of Extended Attributes on Variables			
Extended Attribute	Attribute Variable	Numeric Value	Character Value
classification	jobcode	.	local
length	idnum	4	

Example 6: Concatenating Two SAS Data Sets

Features: PROC DATASETS statement options
 LIBRARY=
 NOLIST
 APPEND statement
 PRINT procedure
 OPTIONS statement

Data set: [EXP Library](#)

Details

This example demonstrates the following tasks:

- suppresses the printing of a library
- appends two data sets
- prints the data sets before appending and prints the new data set after appending

To create the Exp.Results and Exp.Sur data sets and print them out before using this example to concatenate them, see [“EXP Library” on page 2513](#).

Program

```
options pagesize=40 linesize=64 nodate pageno=1;
```

```
LIBNAME exp 'SAS-library';
proc datasets library=exp nolist;
    append base=exp.results data=exp.sur force;
run;
proc print data=exp.results noobs;
    title 'The RESULTS Data Set';
run;
```

Program Description

This example appends one data set to the end of another data set.

The data set Exp.Sur contains the variable Wt6Mos, but the Exp.Results data set does not.

Set the system options. The NODATE option suppresses the display of the date and time in the output. The PAGENO= option specifies the starting page number. The LINESIZE= option specifies the output line length, and the PAGESIZE= option specifies the number of lines on an output page.

```
options pagesize=40 linesize=64 nodate pageno=1;
```

The LIBNAME statement assigns the library.

```
LIBNAME exp 'SAS-library';
```

Suppress the printing of the Exp library. LIBRARY= specifies Exp as the procedure input library. NOLIST suppresses the directory listing for the Exp library.

```
proc datasets library=exp nolist;
```

Append the data set Exp.Sur to the Exp.Results data set. The APPEND statement appends the data set Exp.Sur to the data set Exp.Results. FORCE causes the APPEND statement to carry out the Append operation even though Exp.Sur has a variable that Exp.Results does not. APPEND does not add the Wt6Mos variable to Exp.Results.

```
    append base=exp.results data=exp.sur force;
run;
```

Print the data set.

```
proc print data=exp.results noobs;
    title 'The RESULTS Data Set';
run;
```

Output: Concatenating Two Data Sets

Output 15.21 *The Results Data Set*

The RESULTS Data Set				
ID	TREAT	INITWT	WT3MOS	AGE
1	Other	166.28	146.98	35
2	Other	214.42	210.22	54
3	Other	172.46	159.42	33
5	Other	175.41	160.66	37
6	Other	173.13	169.40	20
7	Other	181.25	170.94	30
10	Other	239.83	214.48	48
11	Other	175.32	162.66	51
12	Other	227.01	211.06	29
13	Other	274.82	251.82	31

Output 15.22 *The Sur Data Set*

The EXP.SUR Data Set					
ID	treat	initwt	wt3mos	wt6mos	age
14	surgery	203.60	169.78	143.88	38
17	surgery	171.52	150.33	123.18	42
18	surgery	207.46	155.22	.	41

Output 15.23 Concatenating the Results and the Sur Data Sets

The RESULTS Data Set				
ID	TREAT	INITWT	WT3MOS	AGE
1	Other	166.28	146.98	35
2	Other	214.42	210.22	54
3	Other	172.46	159.42	33
5	Other	175.41	160.66	37
6	Other	173.13	169.40	20
7	Other	181.25	170.94	30
10	Other	239.83	214.48	48
11	Other	175.32	162.66	51
12	Other	227.01	211.06	29
13	Other	274.82	251.82	31
14	surgery	203.60	169.78	38
17	surgery	171.52	150.33	42
18	surgery	207.46	155.22	41

Example 7: Aging SAS Data Sets

Features: PROC DATASETS statement options
 LIBRARY=
 NOLIST
 AGE statement
 OPTIONS statement

Details

This example demonstrates how the AGE statement ages SAS files.

Program

```
options pagesize=40 linesize=80 nodate pageno=1 source;
```



```
LIBNAME daily 'SAS-library';
proc datasets library=daily nolist;
    age today day1-day7;
run;
```

Program Description

Set the system options. The SOURCE system option writes the programming statements to the SAS log. PAGESIZE= option specifies the number of lines that compose a page of the SAS log and SAS output. LINESIZE= option specifies the line size for the SAS log and for SAS procedure output. NODATE option specifies that the date and the time are not printed. PAGENO= option specifies a beginning page number for the next page of output.

```
options pagesize=40 linesize=80 nodate pageno=1 source;
LIBNAME daily 'SAS-library';
```

Specify Daily as the procedure input library and suppress the directory listing.

```
proc datasets library=daily nolist;
```

Delete and age. Delete the last SAS file in the list, Day7, and then age (or rename) Day6 to Day7, Day5 to Day6, and so on, until it ages Today to Day1.

```
    age today day1-day7;
run;
```

Log Examples

Example Code 15.5 SAS Log

```
6  options pagesize=40 linesize=80
nodate pageno=1 source;
7
8      proc datasets library=daily nolist;
9
10         age today day1-day7;
11     run;
NOTE: Deleting DAILY.DAY7 (memtype=DATA).
NOTE: Ageing the name DAILY.DAY6 to DAILY.DAY7 (memtype=DATA).
NOTE: Ageing the name DAILY.DAY5 to DAILY.DAY6 (memtype=DATA).
NOTE: Ageing the name DAILY.DAY4 to DAILY.DAY5 (memtype=DATA).
NOTE: Ageing the name DAILY.DAY3 to DAILY.DAY4 (memtype=DATA).
NOTE: Ageing the name DAILY.DAY2 to DAILY.DAY3 (memtype=DATA).
NOTE: Ageing the name DAILY.DAY1 to DAILY.DAY2 (memtype=DATA).
NOTE: Ageing the name DAILY.TODAY to DAILY.DAY1 (memtype=DATA).
```

Example 8: Initiating an Audit File

Features:

- PROC DATASETS statement options
 - LIB=
 - AUDIT statement
 - INITIATE statement
 - LOG statement
 - RESUME statement
 - SUSPEND statement
 - TERMINATE statement
 - USER_VAR statement
- SQL procedure

Details

This example demonstrates the following tasks:

- initiates an audit file
- updates the data set
- suspends audit file
- terminates audit file

Program

```
libname mylib "SAS-library";

data mylib.inventory;
input vendor $10. +1 item $4. +1 description $11. +1 units 4.;
datalines;
SmithFarms F001 Apples      10
Tropicana  B002 OrangeJuice 45
UpperCrust C215 WheatBread  25
;
run;

proc datasets lib=mylib;
  audit inventory;
  initiate;
  user_var reason $ 30;
quit;

proc sql;
```

```

        Insert into mylib.inventory values ('Bordens','B132', 'Milk', 100,
                                           'increase on hand');
        Update mylib.inventory      set units=10, reason='recounted inventory'
                                   where item='B002';
quit;

proc datasets lib=mylib;
audit inventory;
log admin_image=no;
suspend;
quit;

proc sql;
select * from mylib.inventory(type=audit);
quit;

proc datasets lib=mylib;
audit inventory;
resume;
quit;

proc datasets lib=mylib;
audit inventory;
terminate;
quit;

```

Program Description

Initiates an audit file using USER_VAR.

```

libname mylib "SAS-library";

data mylib.inventory;
input vendor $10. +1 item $4. +1 description $11. +1 units 4.;
datalines;
SmithFarms F001 Apples      10
Tropicana   B002 OrangeJuice 45
UpperCrust  C215 WheatBread 25
;
run;

proc datasets lib=mylib;
    audit inventory;
    initiate;
    user_var reason $ 30;
quit;

```

Update the data set.

```
proc sql;
  Insert into mylib.inventory values ('Bordens','B132', 'Milk', 100,
                                     'increase on hand');
  Update mylib.inventory      set units=10, reason='recounted inventory'
                             where item='B002';
quit;
```

Discontinue the logging of ADMIN images and suspend audit file.

```
proc datasets lib=mylib;
  audit inventory;
  log admin_image=no;
  suspend;
quit;
```

View the audit file.

```
proc sql;
  select * from mylib.inventory(type=audit);
quit;
```

Resume the audit file.

```
proc datasets lib=mylib;
  audit inventory;
  resume;
quit;
```

Terminate the audit file.

```
proc datasets lib=mylib;
  audit inventory;
  terminate;
quit;
```

Log Examples

Example Code 15.6 Initiating an Audit File

```

1  options nocenter;
2
3  libname mylib "SAS-library";
NOTE: Libref MYLIB was successfully assigned as follows:
      Engine:          V9
      Physical Name:  /home/userod/mylib
4
5  data mylib.inventory;
6  input vendor $10. +1 item $4. +1 description $11. +1 units 4.;
7  datalines;

NOTE: The data set MYLIB.INVENTORY has 3 observations and 4 variables.
NOTE: DATA statement used (Total process time):
      real time          3.45 seconds
      cpu time           0.00 seconds

11 ;
12 run;
13
14 proc datasets lib=mylib;
NOTE: Writing HTML Body file: sashtml.htm
15   audit inventory;
16   initiate;
WARNING: The audited data file MYLIB.INVENTORY.DATA is not password protected.
Apply an Alter password to prevent accidental
      deletion or replacement of it and any associated audit files.
17   user_var reason $ 30;
18   quit;

NOTE: The data set MYLIB.INVENTORY.AUDIT has 0 observations and 11 variables.
NOTE: PROCEDURE DATASETS used (Total process time):
      real time          18.17 seconds
      cpu time           0.79 seconds

19
20 proc sql;
21   Insert into mylib.inventory values ('Bordens','B132', 'Milk', 100,
22                                     'increase on hand');
NOTE: 1 row was inserted into MYLIB.INVENTORY.

23   Update mylib.inventory      set units=10, reason='recounted inventory'
24                               where item='B002';
NOTE: 1 row was updated in MYLIB.INVENTORY.

25   quit;
NOTE: PROCEDURE SQL used (Total process time):
      real time          2.57 seconds
      cpu time           0.03 seconds

26
27 proc datasets lib=mylib;
28   audit inventory;
29   log admin_image=no;
30   suspend;
31   quit;

NOTE: PROCEDURE DATASETS used (Total process time):
      real time          0.01 seconds
      cpu time           0.01 seconds

```

```
32
33  proc sql;
34  select * from mylib.inventory(type=audit);
35  quit;
NOTE: PROCEDURE SQL used (Total process time):
      real time          0.54 seconds
      cpu time           0.01 seconds

36
37  proc datasets lib=mylib;
37 !                               /* resume audit file */
38  audit inventory;
39  resume;
40  quit;

NOTE: PROCEDURE DATASETS used (Total process time):
      real time          0.01 seconds
      cpu time           0.01 seconds

41
42  /* additional step(s) which update the inventory dataset could go here*/
43
44  proc datasets lib=mylib;
44 !                               /* terminate audit file */
45  audit inventory;
46  terminate;
NOTE: Deleting MYLIB.INVENTORY (memtype=AUDIT).
47  quit;

NOTE: PROCEDURE DATASETS used (Total process time):
      real time          0.01 seconds
      cpu time           0.01 seconds
```

Output Examples

Output 15.24 Inventory Contents for MyLib Library

Directory				
Libref		MYLIB		
Engine		V9		
Physical Name		c:\mylib		
Filename		c:\mylib		
Owner Name		BUILTIN\Administrators		
File Size		4KB		
File Size (bytes)		4096		

#	Name	Member Type	File Size	Last Modified
1	CLASS	DATA	128KB	08/16/2016 14:48:34
2	INVENTORY	DATA	128KB	08/16/2016 15:38:44
3	SALARY	DATA	128KB	04/30/2014 13:42:52

Output 15.25 Audit File Listing for MyLib Library

Directory				
Libref		MYLIB		
Engine		V9		
Physical Name		c:\mylib		
Filename		c:\mylib		
Owner Name		BUILTIN\Administrators		
File Size		4KB		
File Size (bytes)		4096		

#	Name	Member Type	File Size	Last Modified
1	CLASS	DATA	128KB	08/16/2016 14:48:34
2	INVENTORY	DATA	128KB	08/16/2016 15:38:44
	INVENTORY	AUDIT	65KB	08/16/2016 15:38:44
3	SALARY	DATA	128KB	04/30/2014 13:42:52

Output 15.26 Inventory Data File Contents for MyLib Library

vendor	item	description	units	reason	_ATDATETIME_	_ATOBSNO_	_ATRETURNCODE_	_ATUSERID_	_ATOPCODE_	_ATMESSAGE_
Bordens	B132	Milk	100	increase on hand	16AUG2016:15:38:44	4	.	suholm	DA	
Tropicana	B002	OrangeJuice	45		16AUG2016:15:38:44	2	.	suholm	DR	
Tropicana	B002	OrangeJuice	10	recounted inventory	16AUG2016:15:38:44	2	.	suholm	DW	

Example 9: Extended Attributes

Features:

- PROC DATASETS statement options
- LIB=
- NOLIST
- CONTENTS statement
- MODIFY statement
- XATTR ADD DS statement
- XATTR ADD VAR statement

Program

```
libname mylib '/home/userid/mylib';

data mylib.sales;
    purchase="car";
    age=10;
    income=200000;
    kids=3;
    cars=4;
run;

proc datasets lib=mylib nolist;
    modify sales;
        xattr add ds role="train" attrib="table";
        xattr add var purchase (role="target" level="nominal")
            age (role="reject")
            income (role="input" level="interval");

    contents data=sales;
    title 'The Contents of the Sales Data Set That Contains Extended
Attributes';

run;
quit;
```

Program Description

Create MyLib.Sales data set.


```
libname mylib '/home/userid/mylib';

data mylib.sales;
  purchase="car";
  age=10;
  income=200000;
  kids=3;
  cars=4;
run;
```

Adding extended attributes to the data set and to the variables.

```
proc datasets lib=mylib nolist;
  modify sales;
    xattr add ds role="train" attrib="table";
    xattr add var purchase (role="target" level="nominal")
               age (role="reject")
               income (role="input" level="interval");

  contents data=sales;
  title 'The Contents of the Sales Data Set That Contains Extended
Attributes';

run;
quit;
```

Log Examples

Example Code 15.7 Extended Attributes

```
355 libname mylib '/home/userid/mylib';
NOTE: Libref MYLIB was successfully assigned as follows:
      Engine:          V9
      Physical Name:   /home/userid/mylib
356
357 data mylib.sales;
358     purchase = "car";
359     age = 10;
360     income = 200000;
361     kids = 3;
362     cars = 4;
363 run;

NOTE: The data set MYLIB.SALES has 1 observations and 5 variables.
NOTE: DATA statement used (Total process time):
      real time          0.00 seconds
      cpu time           0.00 seconds

364
365 proc datasets lib=mylib nolist ;
366     modify sales;
367         xattr add ds role= "train" attrib="table";
368         xattr add var purchase ( role="target" level="nominal" )
369             age ( role="reject" )
370             income ( role="input" level="interval" );
NOTE: MODIFY was successful for MYLIB.SALES.DATA.
371
372     contents data=sales;
373     title 'The Contents of the Sales Data Set That Contains Extended
Attributes';
374
375 run;

376 quit;

NOTE: PROCEDURE DATASETS used (Total process time):
      real time          1.02 seconds
      cpu time           0.10 seconds
```

Output Examples

Output 15.27 Contents of the Sales Data Set with Extended Attributes

The Contents of the Sales Data Set That Contains Extended Attributes

The DATASETS Procedure

Data Set Name	MYLIB.SALES	Observations	1
Member Type	DATA	Variables	5
Engine	V9	Indexes	0
Created	08/16/2016 15:46:14	Observation Length	40
Last Modified	08/16/2016 15:46:19	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label		EXTATTR Segment Length	256
Data Representation	WINDOWS_64		
Encoding	wlatin1 Western (Windows)		

Alphabetic List of Variables and Attributes			
#	Variable	Type	Len
2	age	Num	8
5	cars	Num	8
3	income	Num	8
4	kids	Num	8
1	purchase	Char	3

Alphabetic List of Data Set Extended Attributes		
Extended Attribute	Numeric Value	Character Value
attrib	.	table
role	.	train

Alphabetic List of Extended Attributes on Variables			
Extended Attribute	Attribute Variable	Numeric Value	Character Value
level	income	.	interval
level	purchase	.	nominal
role	age	.	reject
role	income	.	input
role	purchase	.	target

DATEKEYS Procedure

Overview: DATEKEYS Procedure	527
Overview: The DATEKEYS Procedure	528
Concepts: DATEKEYS Procedure	528
Concepts: The DATEKEYS Procedure	528
Syntax: DATEKEYS Procedure	532
PROC DATEKEYS Statement	533
BY Statement	534
DATEKEYCALENDAR Statement	534
DATEKEYDATA Statement	535
DATEKEYDEF Statement	536
DATEKEYDSOPT Statement	540
DATEKEYKEY Statement	540
DATEKEYPERIODS Statement	543
ID Statement	543
VAR Statement	545
Usage: DATEKEYS Procedure	545
Using the DATEKEYS Procedure	545
Using the DATEKEYKEY Statement	553
Details for the ID Statement	556
Data Set Output	557
Listing the Datekeys By Using the LIST Option	560
Creating a Data Set of BY Statement Variables	560
Written Output	561
Examples: DATEKEYS Procedure	561
Example 1: Methods for Constructing a Datekeys Definition Data Set	561
Example 2: Using User-Defined SAS Datekey Keywords Directly in Other SAS Procedures	566
Example 3: Obtaining a Calendar Variable By Using the DATEKEYS Procedure ..	570
Example 4: Filtering Data Sets By Using the DATEKEYDSOPT Statement	576
References	583

Overview: DATEKEYS Procedure

Overview: The DATEKEYS Procedure

The DATEKEYS procedure enables you to process a single date or a set of dates in a time series using a name to reference the dates. For example, the DATEKEYS procedure can be used to identify time periods that are associated with a datekey, read and write datekeys, and provide details about datekey definitions.

The DATEKEYS procedure can be used to identify time periods that are associated with a datekey.

Some common uses of the DATEKEYS procedure are to define datekeys that identify holiday periods, sales events, or changes to operating hours.

Concepts: DATEKEYS Procedure

Concepts: The DATEKEYS Procedure

A SAS datekey describes a date or time interval that is associated with special events such as holidays and sale periods and time computations.

A datekey has a name, a date or set of dates that are associated with the datekey, and a set of qualifiers.

The DATEKEYS procedure provides results in output data sets that can be interpreted in other SAS procedures. It enables you to define datekeys that can be specified as SAS predefined datekeys when used in conjunction with the SAS system option EVENTDS=. The datekeys that you define can be used as date keywords, just as SAS predefined datekeys are used.

The following example creates datekeys, and then writes the datekey definitions to an output data set named MyHolidays. The datekey definitions are automatically available to SAS High-Performance Forecasting procedures by setting the SAS system option EVENTDS=.

```

proc datekeys;
  datekeydef SuperBowl=
    '15JAN1967'd '14JAN1968'd '12JAN1969'd '11JAN1970'd
    '17JAN1971'd '16JAN1972'd '14JAN1973'd '13JAN1974'd '12JAN1975'd
    '18JAN1976'd '09JAN1977'd '15JAN1978'd '21JAN1979'd '20JAN1980'd
    '25JAN1981'd '24JAN1982'd '30JAN1983'd '22JAN1984'd '20JAN1985'd
    '26JAN1986'd '25JAN1987'd '31JAN1988'd '22JAN1989'd '28JAN1990'd
    '27JAN1991'd '26JAN1992'd '31JAN1993'd '30JAN1994'd '29JAN1995'd
    '28JAN1996'd '26JAN1997'd '25JAN1998'd '31JAN1999'd '30JAN2000'd
    '28JAN2001'd '03FEB2002'd '26JAN2003'd '01FEB2004'd '06FEB2005'd
    '05FEB2006'd '04FEB2007'd '03FEB2008'd '01FEB2009'd '07FEB2010'd
    '06FEB2011'd '05FEB2012'd '03FEB2013'd '02FEB2014'd '01FEB2015'd
    ... '07FEB2016'd '05FEB2017'd '04FEB2018'd '03FEB2019'd '02FEB2020'd
  / PULSE=DAY ;

  datekeydef GoodFriday=Easter / shift=-2 pulse=day;
  datekeykey EasterMonday=Easter / shift=1 pulse=day;
  datekeydata out=MyHolidays condense;
run;

options eventds=(MyHolidays);

proc hpfevents data=sashelp.citiday;
  id date interval=day start='27JAN1991'd end='01APR1991'd;
  eventkey SuperBowl;
  eventkey GoodFriday;
  eventkey EasterMonday;
  eventdata out=MyHolidayEvents condense;
  eventdummy out=MyHolidayDates;
run;

```

The following output shows the results:

Figure 16.1 Event Definition Data Set Based on User-Defined Datekeys

The SAS System		
Obs	_NAME_	_KEYNAME_
1	SuperBowl	SUPERBOWL
2	GoodFriday	GOODFRIDAY
3	EasterMonday	EASTERMONDAY

The following statements display an output data set that shows variables for the Super Bowl, Good Friday, and Easter Monday events. The first output shows the results when Month=1. The second output shows the results when Month GE 3.

```

proc print data=MyHolidayDates (where=(month(date)=1));
  var date SuperBowl GoodFriday EasterMonday;
run;

proc print data=MyHolidayDates (where=(month(date) GE 3));
  var date SuperBowl GoodFriday EasterMonday;

```

```
run;
```

Figure 16.2 Output Data Set Based on User-Defined Datekeys: Month=1

The SAS System				
Obs	DATE	SuperBowl	GoodFriday	EasterMonday
1	27JAN1991	1	0	0
2	28JAN1991	0	0	0
3	29JAN1991	0	0	0
4	30JAN1991	0	0	0
5	31JAN1991	0	0	0

Figure 16.3 Output Data Set Based on User-Defined Datekeys: Month GE 3

The SAS System				
Obs	DATE	SuperBowl	GoodFriday	EasterMonday
34	01MAR1991	0	0	0
35	02MAR1991	0	0	0
36	03MAR1991	0	0	0
37	04MAR1991	0	0	0
38	05MAR1991	0	0	0
39	06MAR1991	0	0	0
40	07MAR1991	0	0	0
41	08MAR1991	0	0	0
42	09MAR1991	0	0	0
43	10MAR1991	0	0	0
44	11MAR1991	0	0	0
45	12MAR1991	0	0	0
46	13MAR1991	0	0	0
47	14MAR1991	0	0	0
48	15MAR1991	0	0	0
49	16MAR1991	0	0	0
50	17MAR1991	0	0	0
51	18MAR1991	0	0	0
52	19MAR1991	0	0	0
53	20MAR1991	0	0	0
54	21MAR1991	0	0	0
55	22MAR1991	0	0	0
56	23MAR1991	0	0	0
57	24MAR1991	0	0	0
58	25MAR1991	0	0	0
59	26MAR1991	0	0	0
60	27MAR1991	0	0	0
61	28MAR1991	0	0	0
62	29MAR1991	0	1	0
63	30MAR1991	0	0	0
64	31MAR1991	0	0	0
65	01APR1991	0	0	1

Syntax: DATEKEYS Procedure

```

PROC DATEKEYS<options>;
  BY variable(s);
  DATEKEYCALENDAR OUT=SAS-data-set <SUMMARY=SAS-variable-name>;
  DATEKEYDATA IN=SAS-data-set | OUT=SAS-data-set <options>;
  DATEKEYDEF SAS-variable-name=timing-value </qualifier-options>;
  DATEKEYDSOPT LOCALE=<(ONLY)>'POSIX locale';
  DATEKEYKEY <SAS-variable-name=> datekey-keyword </qualifier-options>;
  DATEKEYPERIODS OUT=SAS-data-set;
  ID SAS-variable-name INTERVAL=interval <options>;
  VAR variable(s);

```

Statement	Task	Example
PROC DATEKEYS	Creates and manages datekeys that are associated with time computations	Ex. 1 , Ex. 2 , Ex. 3 , Ex. 4
BY	Obtains separate calendar variables for groups of observations that are defined by the BY variables	
DATEKEYCALENDAR	Writes variables that indicate the active time periods for datekeys	Ex. 3 , Ex. 4
DATEKEYDATA	Inputs and outputs datekeys	Ex. 1 , Ex. 2 , Ex. 3 , Ex. 4
DATEKEYDEF	Defines a datekey that can be used in other SAS procedures	Ex. 1 , Ex. 2 , Ex. 3 , Ex. 4
DATEKEYDSOPT	Limits input and output processing of data sets to a specified locale	Ex. 4
DATEKEYKEY	Alters a user-defined or predefined SAS datekey, or creates a new datekey from another datekey	Ex. 1 , Ex. 2 , Ex. 3 , Ex. 4
DATEKEYPERIODS	Writes variables that list the active time periods in the input time ID for datekeys	Ex. 1
ID	Names a numeric variable containing SAS date, datetime or time values that identifies observations by time period in input and output data sets	Ex. 1 , Ex. 2 , Ex. 4

Statement	Task	Example
<code>VAR</code>	Copies input variables to the output calendar variables data set	

PROC DATEKEYS Statement

Creates and manages datekeys that are associated with time computations.

Syntax

PROC DATEKEYS *<options>*;

Optional Argument

option

The following options are available:

DATA=SAS-data-set

names the SAS data set that contains the variables that are used in the VAR, ID, and BY statements.

Tip If the DATA= option is not specified, the most recently created SAS data set is used.

LEAD=number-of-periods

specifies the number of periods to extend the calendar variables beyond the input time ID. The LEAD= value is relative to the last observation in the input data set. If BY variables are specified, the LEAD= value is relative to the last observation in each BY group.

Default 0

MAXERROR=number

specifies the maximum number of warning and error messages that are produced during the execution of the procedure.

Default 25

Tip This option is particularly useful in BY-group processing, where it can be used to suppress the recurring messages.

SORTNAMES

specifies that the datekeys and variables in the output data sets be written in alphabetical order. Variables are sorted within their groups. Variables that are listed in the VAR statement are sorted with respect to other variables

that are listed in the VAR statement. Calendar variables are sorted with respect to other calendar variables.

BY Statement

Obtains separate calendar variables for groups of observations that are defined by the BY variables.

Syntax

BY *variable(s)*;

Required Argument

variable

specifies a variable that is used to obtain separate calendar variables for groups of observations that are defined by the BY variables.

Tip When a BY statement appears, the procedure expects the input data set to be sorted in order of the BY variables. If your input data set is not sorted in ascending order, use one of the following alternatives: Sort the data by using the SORT procedure with a similar BY statement, or create an index for the BY variables by using the DATASETS procedure. For more information, see the DATASETS procedure in *Base SAS Procedures Guide*.

DATEKEYCALENDAR Statement

Writes variables that indicate the active time periods for datekeys. The active periods are indicated with a value of 1, and the inactive periods are indicated with a value of 0.

Syntax

DATEKEYCALENDAR OUT=*SAS-data-set* <SUMMARY=*SAS-variable-name*>;

Summary of Optional Arguments

SUMMARY SUM=*SAS-variable-name*

Required Argument

OUT=SAS-data-set

names the output data set to contain the calendar variables for the specified datekeys based on the ID information as specified in the ID statement.

Tip SAS-data-set also includes variables that are specified in the VAR, BY, and ID statements.

Optional Argument

SUMMARY SUM=SAS-variable-name

specifies that the datekey calendar variables be summed and that the result be placed in the specified variable in the DATEKEYCALENDAR OUT= data set.

The SUM= variable can be interpreted as the number of keydates that are active for each time interval. If the SUM= option is not specified, no such variable is included in the OUT= data set.

DATEKEYDATA Statement

Inputs datekeys from a datekeys data set or writes datekeys to a datekeys data set. You can specify multiple DATEKEYDATA statements.

Syntax

```
DATEKEYDATA IN=SAS-data-set | OUT=SAS-data-set <options>;
```

Summary of Optional Arguments

option

Required Arguments

IN=SAS-data-set

names an input data set that contains datekey definitions.

OUT=SAS-data-set

names the output data set to contain the datekey definitions that are specified in the DATEKEYDATA IN= data sets and in the DATEKEYDEF and DATEKEYKEY statements.

Tip If the LIST option is not specified, the OUT= data set can then be used in other SAS procedures and system options to define datekeys.

Optional Argument

option

The following options are available:

CONDENSE

specifies that the DATEKEYDATA OUT= data set be condensed. Any variables that contain only default values are omitted from the data set.

The DATEKEYDATA IN= option reads both condensed data sets and data sets that have not been condensed. For more information, see [“Identifying Variables in the DATEKEYDATA OUT= Data Set”](#) on page 557.

LIST

specifies that the DATEKEYDATA OUT= data set contain only a list of the available datekeys. When you specify the LIST option, the output data set does not contain the parameters that are required for datekey definition.

NODEFAULTS

specifies that the DATEKEYDATA OUT= data set not contain any SAS predefined datekeys.

DATEKEYDEF Statement

Defines a datekey that can be interpreted in other SAS procedures. You can specify multiple DATEKEYDEF statements.

Note: These datekeys can be used to create events that can be included in forecasting models.

Syntax

```
DATEKEYDEF SAS-variable-name=timing-value </qualifier-options>;
```

Summary of Optional Arguments

qualifier-options

Required Arguments

SAS-variable-name

specifies a name in the DATEKEYDEF statement.

timing-value list

specifies one or more datekeys, dates, datetime values, or observation numbers. You can also specify a *value-list*.

The way you specify *value-list* depends on the type of variable:

integer value-list

For integer variables, *integer value-list* is either an explicit list of one or more integers or a starting value and an ending value with an interval increment, or a combination of both forms:

- *n* <...*n*>

Here is an example:

```
10,11,12
```

- *n* TO *n* <BY *increment*>

Here is an example:

```
10 to 12 by 1
```

- *n* <...*n*> TO *n* TO *n* <BY *increment*> <*n*<...*n*> >

Here is an example:

```
11 to 5, 5 to 10 by 1;
```

SAS keyword value-list

For character variables, *SAS keyword value-list* is a list of one or more unique character values that are separated by blanks:

```
"value-1" <"value-2"... "value-n">
```

Here is an example:

```
"INDEPENDENCE"
```

or

```
"INDEPENDENCE" "EASTER" "NEWYEARS"
```

SAS date value-list and SAS datetime value-list

For date and time values, *value-list* can have the following forms:

- "SAS-value"*i* <..."SAS-value"*i*>

Here are some examples:

```
'01Jan2000'd, '01Feb2000'd, '01Mar2000'd
```

or

```
'01Mar1990:15:03:00'dt
```

or

```
'01Jan:15:03:00'dt, '1Feb:15:03:00'dt, '01Mar:15:03:00'dt
```

- "SAS-value"*i* TO "SAS-value"*i* < BY *interval*>

Note: "SAS-value"*i* TO "SAS-value"*i* must be integers, dates, datetime, or time values. Do not mix types.

Here are some examples:

```
'01Jan2000'd TO '01Mar2000'd BY month
```

or

```
'01Jan1990:15:03:00'dt TO '01Mar1990:15:03:00'dt
```

Optional Argument

qualifier-options

The following qualifier options are available:

AFTER=(*<DURATION=value>*)

specifies options that control the datekey definition after the timing value.

The DURATION= suboption is used within the parentheses in the AFTER= () option. DURATION specifies the datekey duration after the timing value.

BEFORE=(*<DURATION=value>*)

specifies options that control the datekey definition before the timing value.

The DURATION= suboption is used within the parentheses in the BEFORE= () option. DURATION specifies the datekey duration before the timing value.

LABEL='SAS-label'

specifies a label that is associated with the datekey. 'SAS-label' is a text string that is enclosed in quotation marks and can be up to 256 characters.

The default label is 'SAS-variable-name' where SAS-variable-name is the name that is specified in the DATEKEYDEF statement. The label is stored in the DATEKEYDATA OUT= data set.

LOCALE='POSIX locale'

specifies a locale that is associated with the datekey. The locale should be a POSIX locale value. There is no default for the locale value.

PERIOD=interval

specifies the interval for the frequency of the datekey. For example, PERIOD=YEAR produces a datekey that is periodic in a yearly pattern.

If the PERIOD= option is omitted, the datekey is not periodic. The PERIOD= option does not apply to observation numbers, which are not periodic, or to date keywords, which have their own periodicity. For intervals that you can specify, see Chapter 4, "Date Intervals, Formats, and Functions" in *SAS/ETS User's Guide*.

PULSE=interval

specifies the interval to be used with the DURATION= option to determine the width of the datekey.

If the datekey is evaluated with respect to a time ID variable, then the default pulse is one observation. When no DURATION= values are specified and the PULSE= option is specified, the DURATION= values are set to zero. For intervals that you can specify, see Chapter 4, "Date Intervals, Formats, and Functions" in *SAS/ETS User's Guide*.

RULE=value

specifies the action to take when the defined datekey has multiple timing values that include at least one datekey.

When the datekey timing values consist only of SAS date, SAS datetime values, and observation numbers, the RULE= option does not apply. The RULE= option also does not apply when the timing value list consists of a single datekey. The RULE= option accepts the values AND and OR. The default is RULE=OR.

The following examples demonstrate the RULE= option:


```

datekeykey JANUARY / pulse=month;
datekeydef RainyDays='11JUL2013'd '13JUL2013'd '21JUL2013'd;
datekeydef HotDays= '11JUL2013'd '16JUL2013'd '17JUL2013'd
                    '18JUL2013'd '19JUL2013'd;
datekeydef FridaysInJanuary=JANUARY FRIDAY / rule=and;
datekeydef JanuaryPlusFridays=JANUARY FRIDAY / rule=or;
datekeydef HotandRainyDays=RainyDays HotDays / rule=and;

```

The RULE= option does not apply to the first statement because the timing value list consists of a single datekey. The RULE= option does not apply to the second and third statements because the timing value list consists only of SAS date values. The operation between two SAS date, datetime, or observation values is always OR. In the fourth statement, the RULE=AND option identifies only dates where the month is January and the day of the week is Friday. In the fifth statement, the RULE=OR option identifies all dates in January and all dates where the day of the week is Friday. In the sixth statement, the RULE=AND option identifies days that are both rainy and hot. The RULE=AND applies to the two datekeys, RainyDays and HotDays.

Usually, the result of an AND operation between two discrete time periods is an empty value. Therefore, the OR operation is always used between discrete time periods. An example is '13JUL2013'd, '01Mar1990:15:03:00'dt, 3.

The following table explains how the RULE= option is interpreted for each observation:

Table 16.1 Definition of RULE= Option Values

RULE= option	Name	Definition
AND	and	identifies time periods that are identified by all datekeys and the group of discrete timing values
OR	or	identifies time periods that are identified by any datekeys or the group of discrete timing values

SHIFT=number

specifies the number of pulses to shift the timing value δ . The default is not to shift the timing value ($\delta=0$). When the SHIFT= option is used, all timing values in the list (including those that are generated by date keywords) are shifted. Therefore, SHIFT= can be used with EASTER to specify ecclesiastical holidays that are based on Easter. For example, the following statement specifies Good Friday, which is defined as two days before Easter (Montes 2001).

```
datekeydef GoodFriday=EASTER / shift=-2 pulse=day;
```

DATEKEYDSOPT Statement

Limits the input and output processing of data sets to a specified locale.

Syntax

```
DATEKEYDSOPT LOCALE= 'POSIX locale' <(ONLY)>;
```

Summary of Optional Arguments

(ONLY)

Required Arguments

LOCALE=

specifies a locale that is used to filter input and output data sets.

'POSIX locale'

specifies a POSIX locale value. There is no default for the locale value.

Optional Argument

(ONLY)

specifies to process only the specified locale, both for input and output data sets. If (ONLY) is not specified, the specified locale and defaults (no specified locale) are processed, both for input and output data sets.

DATEKEYKEY Statement

Alters a user-defined or predefined SAS datekey, or creates a new datekey from another datekey. You can specify multiple DATEKEYKEY statements.

See: [“Using the DATEKEYKEY Statement” on page 553](#)

Syntax

```
DATEKEYKEY <SAS-variable-name=> datekey-keyword </qualifier-options>;
```

Summary of Optional Arguments

SAS-variable-name

Required Argument

datekey-keyword

specifies the default SAS variable name for a user-defined or predefined datekey.

Optional Argument

SAS-variable-name

specifies the name of the new datekey keyword.

TIP If you specify the DATEKEYCALENDAR option, a variable by this name is created and indicates the active and inactive periods of the keyword. If this option is not specified, then *datekey-keyword* is the name of the altered datekey.

Other Optional Arguments

qualifier-option

The following options are available:

AFTER=(*<DURATION=value>*)

specifies options that control the datekey definition after the timing value. The DURATION= suboption is used within the parentheses in the AFTER () option.

DURATION specifies the datekey duration after the timing value when used in the AFTER= option.

BEFORE=(*<DURATION=value>*)

specifies options that control the datekey definition before the timing value. The DURATION= suboption is used within the parentheses in the BEFORE () option.

DURATION specifies the datekey duration before the timing value when used in the BEFORE= option.

LABEL='SAS-label'

specifies a label that is associated with the datekey. 'SAS-label' is a text string that is enclosed in quotation marks and can be up to 256 characters. The default label is 'SAS-variable-name', where *SAS-variable-name* is the name that is specified in the DATEKEYKEY statement. If *SAS-variable-name* is not specified in the DATEKEYKEY statement, then the label is the default label for the SAS predefined datekey. The label is stored in the DATEKEYDATA OUT= data set.

LOCALE='POSIX locale'

specifies a locale that is associated with the datekey. The locale should be a POSIX locale value. There is no default for the locale value.

PERIOD=interval

specifies the interval for the frequency of the datekey. For example, PERIOD=YEAR produces a datekey that is periodic in a yearly pattern. If the PERIOD= option is omitted, the datekey is not periodic. The PERIOD= option does not apply to observation numbers, which are not periodic, or to date keywords, which have their own periodicity. For intervals that can be specified, see Chapter 4, "Date Intervals, Formats, and Functions" in *SAS/ETS User's Guide*.

PULSE=interval

specifies the interval to be used with the DURATION= option to determine the width of the datekey. If the datekey is evaluated with respect to a time ID variable, then the default pulse is one observation. When no DURATION= values are specified and the PULSE= option is specified, the DURATION= values are set to zero. For intervals that can be specified, see Chapter 4, "Date Intervals, Formats, and Functions" in *SAS/ETS User's Guide*.

RULE=value

specifies the action to take when the defined datekey has multiple timing values that include at least one datekey. When the datekey timing values consist only of SAS date, SAS datetime, and observation numbers, the RULE= option does not apply. The RULE= option also does not apply when the timing value list consists of a single datekey. The RULE= option accepts the values AND and OR. The default is RULE=OR. The following examples demonstrate the RULE= option:

```
datekeykey JANUARY / pulse=month;
datekeydef RainyDays='11JUL2013'd '13JUL2013'd '21JUL2013'd;
datekeydef HotDays= '11JUL2013'd '16JUL2013'd '17JUL2013'd
                   '18JUL2013'd '19JUL2013'd;
datekeydef FridaysInJanuary=JANUARY FRIDAY / rule=and;
datekeydef JanuaryPlusFridays=JANUARY FRIDAY / rule=or;
datekeydef HotandRainyDays=RainyDays HotDays / rule=and;
```

The RULE= option does not apply to the first statement because the timing value list consists of a single datekey. The RULE= option does not apply to the second and third statements because the timing value list consists only of SAS date values. The operation between two SAS date, datetime, or observation values is always OR. In the fourth statement, the RULE=AND option identifies dates that are both in January and the day of the week is Friday. In the fifth statement, the RULE=OR option identifies all dates in January and all dates where the day of the week is Friday. In the sixth statement, the RULE=AND option identifies days that are both rainy and hot. The RULE=AND applies to the two datekeys, RainyDays and HotDays.

Usually, the result of an AND operation between two discrete time periods is an empty value. Therefore, the OR operation is always used between discrete time periods. An example is '13JUL2013'D, '01Mar1990:15:03:00'DT, 3.

[Table 16.51 on page 539](#) explains how the RULE= option is interpreted for each observation.

SHIFT=number

specifies the number of pulses to shift the timing value δ . The default is not to shift the timing value ($\delta=0$). When the SHIFT= option is used, all timing values in the list (including those generated by date keywords) are shifted.

DATEKEYPERIODS Statement

Writes variables that list the active time periods in the input time ID for datekeys. The active period dates, datetime values, or observation numbers, are listed with the associated datekey.

Syntax

DATEKEYPERIODS OUT=*SAS-data-set*;

Required Argument

OUT=SAS-data-set

names the output data set to contain the active dates, datetime values, or observations for the specified datekeys based on the ID information that is specified in the ID statement. The OUT= data set also includes variables that are specified in the BY and ID statements.

ID Statement

Specifies a numeric variable that identifies observations in the input and output data sets.

See: [“Details for the ID Statement” on page 556](#)

Syntax

ID *SAS-variable-name* INTERVAL=*interval* <*options*>;

Summary of Optional Arguments

option

Required Arguments

SAS-variable-name

specifies a numeric variable that identifies observations in the input and output data sets. *SAS-variable-name* can be a SAS date, time, datetime value, or an observation number.

INTERVAL=interval

specifies the frequency of the input time ID. For example, if the time ID in the input data set consists of quarterly observations, then use INTERVAL=QTR. For intervals that can be specified, see Chapter 4, “Date Intervals, Formats, and Functions” in *SAS/ETS User’s Guide*.

Optional Argument

option

The following options are available:

ALIGN=option

controls the alignment of SAS dates that are used to identify output observations. The ALIGN= option accepts the following values: BEGINNING | BEG | B, MIDDLE | MID | M, and ENDING | END | E.

Default BEGINNING

END=value

specifies a SAS date, time, or datetime value that represents the end of the data. If the value of the last observation of the time ID variable is less than *value* the series is extended with missing values. If the value of the last observation of the time ID variable is greater than *value*, the series is truncated. For example, END=&sysdate"D uses the automatic macro variable SYSDATE to extend or truncate the series to the current date. You can use this option and the START= option and the last observation of the time ID variable. The default is the value of the last observation of the time ID variable if either of the following is true:

- the START= option is not specified
- the value of the last observation of the time ID variable is greater than or equal to the START= value.

Otherwise, the default is the same as the START= value.

FORMAT=format

specifies the SAS format for the time ID values. If the FORMAT= option is not specified, the default format is implied from the INTERVAL= option.

START=value

specifies a SAS date, time, or datetime value that represents the beginning of the data. If the value of the first observation of the time ID variable is greater than *value* the series is prefixed with missing values. If the value of the first observation of the time ID variable is less than *value*, the series is truncated. You can specify this option and the END= option and the first observation of the time ID variable. The default is the value of the first observation of the time ID variable if either of the following is true:

- the END= option is not specified
- the value of the first observation of the time ID variable is less than or equal to the END= value.

Otherwise, the default is the same as the END= value.

VAR Statement

Copies input variables to the output calendar variables data set. If the VAR statement is omitted, all numeric variables are selected except those that appear in a BY or ID statement.

Syntax

VAR *variable(s)*;

Required Argument

variable

specifies numeric input variables to be copied to the output calendar variables data set.

Usage: DATEKEYS Procedure

Using the DATEKEYS Procedure

Functional Summary of the DATEKEYS Procedure

The following table summarizes the statements and options that control the DATEKEYS procedure.

Description	Statement	Option
Statements		

Description	Statement	Option
Specifies BY-group processing	BY	Not applicable
Specifies a calendar variable data set	DATEKEYCALENDAR	Not applicable
Specifies a datekey data set	DATEKEYDATA	Not applicable
Specifies a datekey definition	DATEKEYDEF	Not applicable
Specifies a datekey definition based on an existing datekey	DATEKEYKEY	Not applicable
Specifies a data set that contains a list of active periods by datekey	DATEKEYPERIODS	Not applicable
Specifies the time ID variable	ID	Not applicable
Specifies the variables to be copied to the calendar variable data set	VAR	Not applicable
Data Set Options		
Specifies the input data set	PROC DATEKEYS	DATA=
Specifies an output data set that contains calendar variables for datekeys	DATEKEYCALENDAR	OUT=
Specifies a datekeys input data set	DATEKEYDATA	IN=
Specifies a datekeys output data set	DATEKEYDATA	OUT=
Specifies that the datekeys output data set is to be condensed	DATEKEYDATA	CONDENSE
Specifies that the datekeys output data set contain only a list of datekeys	DATEKEYDATA	LIST
Specifies that the datekeys output data set contain no SAS predefined datekeys	DATEKEYDATA	NODEFAULTS
Specifies that input and output data sets process the specified locale	DATEKEYDSOPT	LOCALE=
Specifies an output data set that contains active dates for datekeys	DATEKEYPERIODS	OUT=
Specifies a starting time ID value	ID	START=

Description	Statement	Option
Specifies an ending time ID value	ID	END=
Specifies the format of the ID variable	ID	FORMAT=
Calendar Variable Format Options		
Extends the calendar variables past the end of the input time ID	PROC DATEKEYS	LEAD=
Specifies the frequency of the time ID variables	ID	INTERVAL=
Specifies the interval alignment	ID	ALIGN=
Miscellaneous Options		
Specifies that variables in output data sets are to be in sorted order	PROC DATEKEYS	SORTNAMES
Limits error and warning messages	PROC DATEKEYS	MAXERROR=

Datekey Definitions

The purpose of a datekey definition is to define time periods that are associated with the reference datekey. These time periods are then interpreted in other SAS procedures that define time-dependent features such as events. The time period of the datekey definition is compared to the time period of interest. If the time period of interest falls within the range of the datekey definition, then appropriate action is taken. The datekey definitions can be written to an output file by using the OUT= option of the DATEKEYDATA statement.

Once a datekey has been defined, it is referenced using its SAS variable name. When the datekey definition is written to an output file by using the DATEKEYDATA statement, the datekey is identified by its SAS variable name. As with SAS predefined datekeys, when an event is specified using a user-defined datekey name, a dummy variable is created using the datekey definition. The dummy variable name is the same as the datekey SAS reference name.

Each datekey must have a unique SAS variable name. If two datekey definitions have the same name, the following rules apply:

- If two DATEKEYDEF statements exist using the same name, the second statement is used.

- If a datekey is defined in both a DATEKEYDEF statement and in a data set specified using the DATEKEYDATA statement, the definition in the DATEKEYDEF statement is used.
- Any datekey that is defined using a DATEKEYDEF, DATEKEYKEY, or DATEKEYDATA statement is used, rather than a predefined SAS datekey.

Details of the Timing Value List Specification

Each DATEKEYDEF statement must be defined using one or more timing values. The timing values can be specified using a list. Each item in the list can be a predefined SAS date keyword, an integer, a SAS date, a SAS datetime value, or a *value-list*. For example, the following DATEKEYDEF statement specifies timing values that use each of these methods in the order listed:

```
datekeydef datekey1=USINDEPENDENCE 10 '25Dec2000'd
                                '01Mar1990:15:03:00'dt
                                '01Jan2000'd to '01Mar2000'd by month;
```

The timing values are interpreted as follows: July 4 of any relevant year; the 10th observation in a time series or data set; December 25, 2000; March 1, 1990 at 3:03PM; January 1, 2000; February 1, 2000; and March 1, 2000.

The following two DATEKEYDEF statements specify identical timing values:

```
datekeydef MyFirstDATEKEY='01Jan2000'd to '01Mar2000'd by month;
datekeydef MyNextDATEKEY=('01Jan2000'd, '01Feb2000'd, '01Mar2000'd );
```

The *timing-value list* can be enclosed in parentheses, and commas can separate the items in the list. Numbers must be integers and are always interpreted as observation numbers. The *value-list* can be based on observation numbers, SAS dates, or SAS datetime values. However, the first and second values in the list must be of the same type. SAS always expects the type of the second value to be the same as the type of the first value, and tries to interpret the statement in that way. The following statement yields erratic results:

```
datekeydef baddatekey='01Jan2000'd to '01Mar2000:00:00:00'dt by month;
```

Either the DATEKEYS procedure produces a list much longer than expected or the procedure does not have enough memory to execute.

Note: Do not mix date, datetime, integer, and value types in a *value-list*.

The following table shows the holiday date keywords that can be used in a *timing-value list*, and their definitions:

Table 16.2 Holiday Date Keywords and Definitions

Date Keyword	Definition
BOXING	December 26

Date Keyword	Definition
CANADA	July 1
CANADAOBSERVED	July 1, or July 2, if July 1 is a Sunday
CHRISTMAS	December 25
COLUMBUS	second Monday in October
EASTER ¹	Easter Sunday
FATHERS	third Sunday in June
HALLOWEEN	October 31
LABOR	first Monday in September
MLK	third Monday in January
MEMORIAL	last Monday in May
MOTHERS	second Sunday in May
N<n>W<w><MON>YR	date specified by NWKDOM(<i>n</i> , <i>w</i> , <i>m</i> , <i>year</i>), where <i>m</i> corresponds to the month that is specified by MON, and <i>year</i> is any year relevant to the data. Example: N4W5NOVYR is the same as THANKSGIVING
NEWYEAR	January 1
THANKSGIVING	fourth Thursday in November
THANKSGIVINGCANADA	second Monday in October
USINDEPENDENCE	July 4
USPRESIDENTS	third Monday in February (since 1971)
VALENTINES	February 14
VETERANS	November 11
VETERANSUSG	U.S. government observed date for Monday–Friday schedule

1. The date for Easter is calculated using a method described by Montes (2001).

Date Keyword	Definition
VETERANSUSPS	U.S. government observed date for Monday–Saturday schedule (U.S. Post Office)
VICTORIA	Monday on or preceding May 24

The following table shows the seasonal date keywords that can be used in a *timing value list*, and their definitions:

Table 16.3 Seasonal Date Keywords and Definitions

Date Keyword	Definition
SECOND_1, ...SECOND_60	the specified second.
MINUTE_1, ...MINUTE_60	the beginning of the specified minute.
HOURL_1, ...HOURL_24	the beginning of the specified hour.
SUNDAY, ...SATURDAY	all Sundays, and so on, in the time series.
WEEK_1, ...WEEK_53	the first day of the n th week of the year. PULSE=WEEK. n shifts this date for n NE 1.
TENDAY_1, ...TENDAY_36	the 1st, 11th, or 21st day of the appropriate month.
SEMIMONTH_1, ...SEMIMONTH_24	the 1st or 16th day of the appropriate month.
JANUARY, ...DECEMBER	the 1st day of the specified month.
QTR_1, QTR_2, QTR_3, QTR_4	the first date of the quarter. PULSE=QTR. n shifts this date for n NE 1.
SEMIYEAR_1, SEMIYEAR_2	the first date of the semiyear. PULSE=SEMIYEAR. n shifts this date for n NE 1.

Timing values are evaluated with respect to the application and relevant time specified by the user. Select the timing values that are consistent with the usage. In particular, date and datetime timing values are ignored when no date or time information is specified, and only observation numbers are available for analysis.

Details of Modifying Timing Values Using Options

The qualifier options define functional modifications to be applied at each timing value. The options are applied in the following order:

- 1 If a SHIFT= value is specified, the timing values in the list are shifted using the SHIFT= value, and the PULSE= value, if specified.
- 2 If a DURATION= value is specified in the BEFORE= or AFTER= options, then a continuous interval is defined around the shifted timing values based on the DURATION= and PULSE= values.
- 3 If PERIOD= values are specified, then periodic values are generated based on the timing values that result from steps 1 and 2.

Details of Identifying Active Intervals Based on Timing Values and Options

When a datekey is evaluated with respect to a time ID, active intervals are identified with respect to the time ID intervals. Therefore, the active periods are dependent on both the time ID and the datekey definition.

The observation that is specified by the shifted timing value, t_i , is the observation that contains the date that is generated by

INTNX(interval, timing-value, s, 'same'), where SHIFT=s and PULSE=interval. If no PULSE= value is specified, the default is PULSE=OBS, which is equivalent to PULSE=interval, where interval is the interval of the time ID.

Table 16.4 Calculating the Beginning and Ending Observations for Datekeys When Applied to Time Series

BEFORE=(DURATION=value)	PULSE=value	Definition of t_b
ALL	Not available	$t_b=1$, the first observation in the data set, or t_b = the observation specified by START=
$m=0$	Not specified	$t_b=t_i$, the observation specified by the shifted timing value
$m>0$	Not specified	$t_b=t_i-m$

BEFORE=(DURATION=value)	PULSE=value	Definition of t_b
$m \geq 0$	<i>interval</i>	t_b = the observation specified by the date <code>INTNX(interval, timing-value, -m, 'begin')</code>
AFTER=(DURATION=value	PULSE=value	Definition of t_e
ALL	Not available	t_e = the last observation in the data set, or t_e = the observation specified by <code>END=</code>
$n=0$	Not specified	$t_e = t_p$, the observation specified by the shifted timing value
$n > 0$	Not specified	$t_e = t_p + n$
$n \geq 0$	<i>interval</i>	t_e = the observation specified by the date <code>INTNX(interval, timing-value, n, 'end')</code>

The following table shows active time periods for datekey definitions:

Table 16.5 Active Time Periods for Datekey Definitions

BEFORE=(DURATION=m)	AFTER=(DURATION=n)	Active Observation
$m = \text{ALL}$	$n = \text{ALL}$	ξ_{it} , for all t
$m = \text{finite}$	$n = \text{ALL}$	ξ_{it} , for all $t \geq t_b$
$m = \text{ALL}$	$n = \text{finite}$	ξ_{it} , for all $t \leq t_e$
$m = \text{finite}$	$n = \text{finite}$	ξ_{it} , if $t_b \leq t \leq t_e$

If an observation is not within an active period for a datekey, then it is not altered by the datekey action.

The active period always occurs at the shifted timing value. You specify three observations before the timing value, one observation at the timing value, and four after the timing value. The total would be $3+1+4=8$ observations as follows:

```
datekeydef E1='01JAN1950'd / before=(duration=3)
                                after=(duration=4);
```

In this example, you specify three weeks before the timing value, the week of the timing value, and four weeks after the timing value by using a combination of the BEFORE=, AFTER=, and PULSE= options as follows:

```
datekeydef E1='01JAN1950'd / before=(duration=3)
                                after=(duration=4)
                                pulse=week;
```

DURATION=ALL implies that the active period should be extended to the beginning (BEFORE=) or end (AFTER=) of time. If only one DURATION= value is specified, the other value is assumed to be zero. When neither DURATION= value is specified, both DURATION= values are set to zero. DURATION=ALL is represented in the datekey definition data set as a special missing value displayed as "A". For more information, see [Missing Variable Values](#).

Using the DATEKEYKEY Statement

Creating New Datekeys from Existing Datekey Definitions

A DATEKEYKEY statement can be used with PROC DATEKEYS to create new datekeys from existing datekey definitions and make them available for processing. SAS events that are based on user-defined datekeys are also available directly through PROC HPFDIAGNOSE and PROC HPFENGINE by specifying the EVENTDS= system option.

A user-defined datekey variable has timing values and qualifiers that are defined by the user. A DATEKEYKEY variable that is defined using a predefined SAS datekey has a predefined set of timing values and qualifiers that are associated with the predefined datekey keyword. You can redefine the qualifiers by using the statement options. The options are the same as in the DATEKEYDEF statement. In [“Concepts: The DATEKEYS Procedure” on page 528](#), the default SAS variable name for an event based on a datekey is the datekey keyword. However, you can specify a different SAS name for the datekey. For example, you can rename the CHRISTMAS predefined datekey to XMAS by using the following statement:

```
datekeykey xmas=christmas;
```

If you redefine the qualifiers that are associated with a predefined SAS datekey and do not rename the datekey, then that has the impact of redefining the predefined SAS datekey. This redefinition occurs because any user definition takes precedence over a SAS predefined definition. The following example produces an event named

FALLHOLIDAYS with a pulse of 1 day at Halloween and a pulse of 1 month at Thanksgiving:

```
datekeykey thanksgiving / pulse=month;
eventcomb fallholidays=halloween thanksgiving;
```

The following table describes how to construct a predefined SAS datekey keyword. It also gives the default qualifier options for those predefined datekeys.

Table 16.6 Definitions for DATEKEYKEY Predefined Event Keywords

Variable Name or Variable Name Format	Description	Qualifier Option
AO<obs>OBS	outlier	BEFORE=(DURATION=0)
AO<date>D		AFTER=(DURATION=0)
AO<datetime>DT		
LS<obs>OBS	level shift	BEFORE=(DURATION=0)
LS<date>D		AFTER=(DURATION=ALL)
LS<datetime>DT		
TLS<obs>OBS<n>	temporary level shift	BEFORE=(DURATION=0)
TLS<date>D<n>		AFTER=(DURATION=<n>)
TLS<datetime>DT<n>		
NLS<obs>OBS	negative level shift	BEFORE=(DURATION=0)
NLS<date>D		AFTER=(DURATION=ALL)
NLS<datetime>DT		
CBLS<obs>OBS	U.S. Census Bureau level shift	SHIFT=-1
CBLS<date>D		BEFORE=(DURATION=ALL)
CBLS<datetime>DT		AFTER=(DURATION=0)
TC<obs>OBS	temporary change	BEFORE=(DURATION=0)
TC<date>D		AFTER=(DURATION=ALL)
TC<datetime>DT		
<date keyword>	date pulse	BEFORE=(DURATION=0) AFTER=(DURATION=0)

Variable Name or Variable Name Format	Description	Qualifier Option
LINEAR QUAD CUBIC	polynomial trends	BEFORE=(DURATION=ALL) AFTER=(DURATION=ALL) default timing value is 0 observation
INVERSE LOG	trends	BEFORE=(DURATION=0) AFTER=(DURATION=ALL) default timing value is 0 observation
<seasonal keywords>	seasonal	PULSE= depends on keyword BEFORE=(DURATION=0) AFTER=(DURATION=0) timing values based on keyword

Modifying or Cloning a User-Defined Datekey

A DATEKEYKEY statement can be used in a similar manner to modify or clone a user-defined datekey. In the following example, the DATEKEYDEF statement is used to define a simple event named SPRING. The DATEKEYKEY statement is used to modify the SPRING event definition. Then the DATEKEYKEY statement is used to create a new event named SPRINGBREAK that is based on the previously defined user event named SPRING. Therefore, the example defines a total of two datekeys, SPRING and SPRINGBREAK. The DATEKEYKEY statement can be used to modify the qualifiers. It cannot be used to modify the timing values.

```
datekeydef spring='20mar2005'd;
datekeykey spring / pulse=day;
datekeykey SPRINGBREAK=spring / pulse=week;
```

If the preceding datekeys are stored in a data set named SPRINGHOLIDAYS, the first DATEKEYKEY statement in the following example clones SPRING as a datekey named FirstDayOfSpring. The second DATEKEYKEY statement changes the case of the SPRINGBREAK datekey name.

```
datekeydata in=springholidays;
datekeykey FirstDayOfSpring=spring;
datekeykey Springbreak=springbreak;
```

Datekey names that refer to a previously defined datekey are not case sensitive. However, datekey names that are used to create a new datekey preserve the casing in the `_NAME_` variable of the DATEKEYDATA OUT= data set.

SAS Datekey Keywords

The date keywords that are described in [Table 16.52 on page 548](#) can be used in the DATEKEYDEF statement as predefined SAS datekey keywords. The timing values are as defined in [Table 16.52 on page 548](#). The default qualifiers are as shown in [Table 16.56 on page 554](#). [Table 16.53 on page 550](#) shows the seasonal keywords that can be used as predefined SAS datekey keywords. The default qualifiers for seasonal keywords are shown in [Table 16.56 on page 554](#). [Table 16.57 on page 556](#) describes how date and observation numbers are encoded into AO, LS, TLS, NLS, CBLs, and TC enter predefined datekeys.

SAS predefined date keywords have only active periods.

Table 16.7 Encoding Data Information into AO, LS, TLS, NLS, CBLs, and TC Type DATEKEYKEY Variable Names

Variable Name Format	Example	Refers To
AO<int>OBS	AO15OBS	15th observation
AO<date>D	AO01JAN2000D	'01JAN2000'D
AO<date>h<hr>m<min>s<sec>DT	AO01Jan2000h12m34s56DT	'01Jan2000:12:34:56'DT
TLS<int>OBS<n>	TLS15OBS10	15th observation
TLS<date>D<n>	TLS01JAN2000D10	'01JAN2000'D
TLS<date>h<hr>m<min>s<sec>DT<n>	TLS01Jan2000h12m34s56DT10	'01Jan2000:12:34:56'DT >

Details for the ID Statement

The ID statement accepts a SAS variable name and an interval. The ID variable's values are assumed to be SAS date, time, datetime values, or observation numbers. In addition, the ID statement specifies the desired frequency for examining active and inactive time periods. The information that is specified affects all variables that are generated by using the DATEKEYCALENDAR and DATEKEYPERIODS statements. If no DATEKEYCALENDAR or DATEKEYPERIODS statements are specified, the ID statement has no impact on processing, because the DATEKEYDEF definitions are independent of the time identification values and

frequencies. If the ID statement is specified, the INTERVAL= option must also be specified. If an ID statement is not specified, the observation number (with respect to the BY group) is used as the time ID. In this case, only datekey timing values that are based on observation numbers are applied to the input time ID to create calendar variables. Timing values that are based on SAS date or datetime values are ignored.

Data Set Output

Creating Data Set Output

The DATEKEYS procedure can create the DATEKEYCALENDAR OUT=, DATEKEYDATA OUT=, and DATEKEYPERIODS OUT= data sets. The DATEKEYDATA OUT= data set contains the datekey definitions that can be used for input to another SAS procedure. If the LIST option is specified in the DATEKEYDATA OUT= statement, then the output data set contains a list of available datekeys. The DATEKEYCALENDAR OUT= data set contains calendar indicator variables that show the active periods for the datekeys relative to the input time ID. The DATEKEYPERIODS OUT= data set contains information about datekeys that are active for the input time ID and the associated dates.

Identifying Variables in the DATEKEYCALENDAR OUT= Data Set

The DATEKEYCALENDAR OUT= data set contains the variables that are listed in the BY statement, the ID variable, any variables that are defined by the VAR statement, and any calendar variables that are generated by the procedure. The calendar indicator variables show the active periods for the datekeys relative to the input time ID. You can specify the SUM= option in the DATEKEYCALENDAR statement. In this case, the variable that is specified in the SUM= option is included in the DATEKEYCALENDAR OUT= data set and contains the sum of the calendar indicator variables.

Identifying Variables in the DATEKEYDATA OUT= Data Set

The DATEKEYDATA OUT= data set contains the following variables. The default values for the CONDENSE option are also given. When all the observations in the variable are equal to the default value, the variable can be omitted from the datekey definition data set.

CLASS

specifies that the class for all datekeys is DATEKEY. The default for _CLASS_ is DATEKEY.

DATEINTRVL

specifies the interval for the date *value-list*. The default for _DATEINTRVL_ is no interval, designated by ".".

DTINTRVL

specifies the interval for the datetime *value-list*. The default for _DTINTRVL_ is no interval, designated by ".".

_DUR_AFTER_

specifies the number of durations after the timing value. The default for _DUR_AFTER_ is 0.

_DUR_BEFORE_

specifies the number of durations before the timing value. The default for _DUR_BEFORE_ is 0.

ENDDATE

specifies the last date timing value to use in a *value-list*. The default for _ENDDATE_ is no date, designated by a missing value.

ENDDT

specifies the last datetime timing value to use in a *value-list*. The default for _ENDDT_ is no datetime, designated by a missing value.

ENDOBS

specifies the last observation number timing value to use in a *value-list*. The default for _ENDOBS_ is no observation number, designated by a missing value.

KEYNAME

specifies either a predefined datekey keyword or a user-defined datekey keyword. All _KEYNAME_ values are displayed in uppercase. However, if the _KEYNAME_ value refers to a user-defined keyword, then the actual name can be mixed case. The default for _KEYNAME_ is no keyname, designated by ".".

LABEL

specifies a label or description for the datekey. If you do not specify a label, then the default label value is displayed as ".". For more information, see the LABEL system option in *SAS System Options: Reference*.

LOCALE

specifies the locale for the datekey. _LOCALE_ values are the valid POSIX locale values. For more information, see the LOCALE system option in *SAS National Language Support (NLS): Reference Guide*. There is no default.

NAME

specifies a datekey reference name. _NAME_ is displayed with the case preserved. Because _NAME_ is a SAS variable name, the datekey can be referenced using any case. The _NAME_ variable is required. There is no default.

OBSINTRVL

specifies the interval length of the observation number *value-list*. The default for _OBSINTRVL_ is no interval, designated by ".".

PERIOD

specifies the frequency interval at which the datekey should be repeated. If this value is missing, then the datekey is not periodic. The default for **_PERIOD_** is no interval, designated by ".".

PULSE

specifies an interval that defines the units for the DURATION values. The default for **_PULSE_** is no interval (one observation), designated by ".".

RULE

specifies the rule to use when you combine the timing values of a datekey. The default for **_RULE_** for datekeys is OR.

SHIFT

specifies the number of PULSE= intervals to shift the timing value. The shift can be positive (forward in time) or negative (backward in time). If PULSE= is not specified, then the shift occurs in observations. The default for **_SHIFT_** is 0.

STARTDATE

specifies either the date timing value or the first date timing value to use in a *value-list*. The default for **_STARTDATE_** is no date, designated by a missing value.

STARTDT

specifies either the datetime timing value or the first datetime timing value to use in a *value-list*. The default for **_STARTDT_** is no datetime, designated by a missing value.

STARTOBS

specifies either the observation number timing value or the first observation number timing value to use in a *value-list*. The default for **_STARTOBS_** is no observation number, designated by a missing value.

Identifying Variables in the DATEKEYPERIODS OUT= Data Set

The DATEKEYPERIODS OUT= data set contains the following variables.

LOCALE

specifies the locale for the datekey. **_LOCALE_** values are the valid POSIX locale values. For more information, see the LOCALE system option in *SAS National Language Support (NLS): Reference Guide*. There is no default.

NAME

specifies a datekey reference name. **_NAME_** is displayed with the case preserved. Because **_NAME_** is a SAS variable name, the datekey can be referenced using any case. The **_NAME_** variable is required. There is no default.

STARTDATE

specifies either the date timing value or the first date timing value to use in a *value-list*. The default for **_STARTDATE_** is no date, designated by a missing value.

TIME ID
specifies the date.

Listing the Datekeys By Using the LIST Option

You can use the LIST option in the DATEKEYDATA statement to list datekeys. Specifying the LIST option in the DATEKEYDATA OUT= statement alters the output data set. The purpose of the LIST option is to create a data set that lists the available datekeys that are defined by the data set. When the LIST option is specified, the data set contains only the _LOCALE_, _NAME_, and _LABEL_ variables. The data set contains one observation for each datekey.

Creating a Data Set of BY Statement Variables

You can create a data set of BY statements by using the DATEKEYPERIODS statement. The DATEKEYPERIODS OUT= data set contains the following items:

- variables that are listed in the BY statement
- a list of active datekeys that are associated with the input time ID
- the ID variable dates that are associated with the active datekeys
- the starting date, datetime, or observation value that is associated with the active time ID period

In addition to the BY variables and the ID variable, the DATEKEYPERIODS OUT= data set contains the following variables:

NAME
specifies a datekey reference name. _NAME_ is displayed with the case preserved. The datekey that is displayed is active on the dates that are specified by the ID variable and the _STARTDATE_, _STARTDT_, or _STARTOBS_ values for the observation.

One of the following variables is included, based on the time ID variable:

STARTDATE
specifies the date that is associated with the beginning of the period that is specified by the INTERVAL= option relative to the value of the time ID variable.

STARTDT
specifies the datetime that is associated with the beginning of the period that is specified by the INTERVAL= option relative to the value of the time ID variable.

STARTOBS
specifies the observation number that is associated with the beginning of the period that is specified by the INTERVAL= option relative to the value of the time ID variable.

Written Output

The DATEKEYS procedure has no written output other than warning and error messages as recorded in the log.

Examples: DATEKEYS Procedure

Example 1: Methods for Constructing a Datekeys Definition Data Set

Features:	PROC DATEKEYS statement options
	ID
	DATEKEYDEF
	DATEKEYKEY
	DATEKEYDATA
	DATEKEYPERIODS

Details

This example uses two methods to construct a datekeys definition data set. The first method uses the DATA step to construct a datekey data set for Fridays when an item was on sale. The second method uses the DATEKEYS procedure to construct the datekey definition data set. The TimeSeriesDates data set contains the time ID variable for the time series.

The same dates that are defined in WhiteSaleDates in the DATA step can be defined using the DATEKEYS procedure. The active dates are the same as in the data set that is shown in [Output 16.89 on page 563](#). However, the definitions in the DATEKEYS procedure are continuous.

Program: Using the DATA Step

```
options eventds=(nodefaults);
```

```

data TimeSeriesDates(keep=date);
    set sashelp.citiday;
    format date date.;
run;

data WhiteSaleDates(keep=_name_ _startdate_);
    set TimeSeriesDates;
    _name_='WhiteSale';
    if (month(date)=1) then do;
        if (year(date)=1991 or year(date)=1992) then do;
            if (weekday(date)=6) then do;
                _startdate_=date;
            end;
        else delete;
    end;
    else delete;
end;
format _startdate_ date.;
run;

proc print data=WhiteSaleDates;
run;

```

Program Description

Set the EVENTDS= system option. The EVENTDS= system option specifies the data set that defines the event. The NODEFAULTS option specifies not to use default event definitions. The only events that are used are specified by the event-data-set list.

```
options eventds=(nodefaults);
```

Create the TimeSeriesDates data set. The DATA step uses sashelp.citiday to create the data set for this example:

```

data TimeSeriesDates(keep=date);
    set sashelp.citiday;
    format date date.;
run;

```

Construct a datekeys data set, WhiteSalesDates, for use with the EVENTDS= system option. When you construct a datekeys data set using a DATA step, all observations that define a specific datekey must be consecutive. If necessary, use the SORT procedure to sort the data set by the _NAME_ variable. The DATA step describes an item that was on sale during January 1991 and January 1992. The user-defined datekeys identify the Fridays that the item was on sale.

```

data WhiteSaleDates(keep=_name_ _startdate_);
    set TimeSeriesDates;
    _name_='WhiteSale';
    if (month(date)=1) then do;
        if (year(date)=1991 or year(date)=1992) then do;
            if (weekday(date)=6) then do;
                _startdate_=date;
            end;
        end;
    end;

```



```

        else delete;
      end;
    else delete;
  end;
else delete;
format _startdate_ date.;
run;

proc print data=WhiteSaleDates;
run;

```

Output: HTML

Output 16.1 User-Defined Datekeys Data Set

The SAS System		
Obs	_NAME_	_STARTDATE_
1	WhiteSale	04JAN91
2	WhiteSale	11JAN91
3	WhiteSale	18JAN91
4	WhiteSale	25JAN91
5	WhiteSale	03JAN92
6	WhiteSale	10JAN92
7	WhiteSale	17JAN92
8	WhiteSale	24JAN92
9	WhiteSale	31JAN92

Program: Using the DATEKEYS Procedure

```

proc datekeys data=sashelp.citiday;
  id date interval=day;
  datekeydef Years1991_1992='01JAN1991'd / pulse=year
after=(duration=1);
  datekeykey January / pulse=month;
  datekeydef WhiteSale=JANUARY FRIDAY Years1991_1992 / rule=and;
  datekeydata out=WhiteSaleDefinitions condense;
  datekeyperiods out=WhiteSaleActiveDates;
run;

```

```
proc print data=WhiteSaleActiveDates (where=(_name_='WhiteSale'));
run;

proc print data=WhiteSaleDefinitions;
run;
```

Program Description

Begin the DATEKEYS procedure. The DATA= option contains the variables that are used in the ID statement.

```
proc datekeys data=sashelp.citiday;
```

Specify the time ID variable. The ID statement names a numeric variable that identifies observations in the data set. If you use the ID statement, the INTERVAL= option must be used. In this case, INTERVAL=day.

```
id date interval=day;
```

Define a datekey. The DATEKEYDEF statement defines a datekey. PULSE= specifies the interval to be used with the DURATION= option to determine the width of the datekey.

```
datekeydef Years1991_1992='01JAN1991'd / pulse=year
after=(duration=1);
```

Alter or create a new datekey. Because January is used in the timing value list, it represents the first day of January. Using DATEKEYKEY with PULSE=MONTH creates a datekey for the entire month of January.

```
datekeykey January / pulse=month;
```

Define a datekey. The DATEKEYDEF statement defines a datekey. The RULE=AND option identifies days in January that are Fridays.

```
datekeydef WhiteSale=JANUARY FRIDAY Years1991_1992 / rule=and;
```

Write the datekey to an output data set. The DATEKEYDATA OUT= data set contains the definitions for the WhiteSale datekey. The CONDENSE option specifies that the output data set be condensed. Any variables that contain only default values are omitted from the data set.

```
datekeydata out=WhiteSaleDefinitions condense;
```

Write variables that list the active time periods in the input time ID for datekeys. The DATEKEYPERIODS OUT= data set contains the active time periods for the WhiteSale datekey, as shown in [Output 16.90 on page 565](#).

```
datekeyperiods out=WhiteSaleActiveDates;
```

Execute the example. The RUN statement causes the program to execute.

```
run;
```

Write the active sales dates data set. The results are shown in [Output 16.90 on page 565](#).

```
proc print data=WhiteSaleActiveDates (where=(_name_='WhiteSale'));
run;
```

Write the definitions for the WhiteSale datekey. The results are shown in [Output 16.91 on page 565](#).

```
proc print data=WhiteSaleDefinitions;
run;
```

HTML Output

Output 16.2 Datekeys Active Periods Data Set Output

The SAS System				
Obs	_LOCALE_	DATE	_NAME_	_STARTDATE_
557	.	04JAN1991	WhiteSale	04JAN1991
558	.	11JAN1991	WhiteSale	11JAN1991
559	.	18JAN1991	WhiteSale	18JAN1991
560	.	25JAN1991	WhiteSale	25JAN1991
561	.	03JAN1992	WhiteSale	03JAN1992
562	.	10JAN1992	WhiteSale	10JAN1992
563	.	17JAN1992	WhiteSale	17JAN1992
564	.	24JAN1992	WhiteSale	24JAN1992
565	.	31JAN1992	WhiteSale	31JAN1992

The following output shows the DATEKEYDATA OUT= data set definitions for the WhiteSale datekey:

Output 16.3 Datekey Definitions Data Set Output

The SAS System							
Obs	_NAME_	_CLASS_	_KEYNAME_	_STARTDATE_	_PULSE_	_DUR_AFTER_	_RULE_
1	JANUARY	DATEKEY	JANUARY	.	MONTH	0	OR
2	Years1991_1992	DATEKEY	.	01JAN1991	YEAR	1	OR
3	WhiteSale	DATEKEY	JANUARY	.	.	0	AND
4	WhiteSale	DATEKEY	FRIDAY	.	.	0	AND
5	WhiteSale	DATEKEY	YEARS1991_1992	.	.	0	AND

Example 2: Using User-Defined SAS Datekey Keywords Directly in Other SAS Procedures

Features:

- PROC DATEKEYS statements
 - DATEKEYDEF
 - DATEKEYKEY
 - DATEKEYDATA
- System option
 - EVENTDS=
- Other Procedures
 - HPFDIAGNOSE

Details

If a procedure supports an EVENT statement, then you can use either data set that is shown in [“Example 1: Methods for Constructing a Datekeys Definition Data Set” on page 561](#) without using PROC HPFEVENTS. PROC HPFEVENTS enables you to use a user-defined datekey. However, when the user-defined datekey is specified as an event, the event is created automatically, provided that the user-defined datekey definition data set has been specified using the EVENTDS= system option. This example uses the data set from [“Example 1: Methods for Constructing a Datekeys Definition Data Set” on page 561](#) and the EVENT statement in PROC HPFDIAGNOSE.

The output from PROC HPFDIAGNOSE shows that a model that included the WhiteSale event was selected. The event was a poor fit, but REQUIRED=YES was specified. REQUIRED=YES specifies that the events be included in the model as long as the model does not fail to be diagnosed.

Program

```
proc datekeys;
    datekeydef Years1991_1992='01JAN1991'd / pulse=year
after=(duration=1);
    datekeykey JANUARY / pulse=month;
    datekeydef WhiteSale=JANUARY FRIDAY Years1991_1992 / rule=and;
    datekeydata out=WhiteSaleDefinitions condense;
run;
options eventds=(WhiteSaleDefinitions);
```

```
proc hpfdiagnose data=sashelp.citiday
    print=all;

    id date interval=day;

    forecast snysecm;

    event WhiteSale / required=yes;

    arimax;

run;
```

Program Description

Begin the DATEKEYS procedure.

```
proc datekeys;
```

Define a datekey. The DATEKEYDEF statement defines a datekey. PULSE= specifies the interval to be used with the DURATION= option to determine the width of the datekey.

```
    datekeydef Years1991_1992='01JAN1991'd / pulse=year
    after=(duration=1);
```

Alter or create a new datekey. The DATEKEYKEY statement alters a user-defined or predefined SAS datekey, or creates a new datekey from another datekey. PULSE= specifies the interval to be used with the DURATION= option to determine the width of the datekey.

```
    datekeykey JANUARY / pulse=month;
```

Define a datekey. The DATEKEYDEF statement defines a datekey. The RULE=AND option identifies days in January 1991 and 1992 that are Fridays.

```
    datekeydef WhiteSale=JANUARY FRIDAY Years1991_1992 / rule=and;
```

Write the datekey to an output data set. The CONDENSE option specifies that the output data set be condensed. Any variables that contain only default values are omitted from the data set.

```
    datekeydata out=WhiteSaleDefinitions condense;
```

Execute the DATEKEYS procedure.

```
run;
```

Set the EVENTDS= system option. The EVENTDS= system option specifies the data set that defines the event.

```
options eventds=(WhiteSaleDefinitions);
```

Begin the HPFDIAGNOSE procedure. The HPFDIAGNOSE procedure automatically diagnoses the statistical characteristics of time series and identifies appropriate models.

```
proc hpfdiagnose data=sashelp.citiday
    print=all;
```

Specify the ID statement. The ID statement names a numeric variable that identifies observations in the input and output data sets. The ID variable's values are SAS date values. In addition, the ID statement specifies the desired frequency that is associated with the time series. The INTERVAL= option specifies the frequency of the input time ID.

```
id date interval=day;
```

List the variables in the data set that you want to diagnose. The FORECAST statement lists the variables to be diagnosed in the data set that is specified by the DATA= option. The variables are dependent variables or response variables that you want to forecast in the HPFENGINE procedure.

```
forecast snysecm;
```

Name the event. The EVENT statement name identifies the events. The REQUIRED option specifies that the events be included in the model as long as the model does not fail to be diagnosed.

```
event WhiteSale / required=yes;
```

Find an appropriate ARIMAX specification. An ARIMAX model specifies trend factors, seasonal factors, and regression variables, including events. The HPFDIAGNOSE procedure performs the intermittency test first. If the series is not intermittent, an ARIMAX model is fitted to the data.

```
arimax;
```

Execute the HPFDIAGNOSE procedure.

```
run;
```

HTML Output

The following output shows the results:

Output 16.4 Using the EVENT Statement in PROC HPFDIAGNOSE**The SAS System****The HPFDIAGNOSE Procedure**

Variable Information	
Name	SNYSECM
Label	STOCK MKT INDEX:NYSE COMPOSITE, (WSJ)
First	01JAN1988
Last	05FEB1992
Number of Observations Read	1497

Seasonal Dickey-Fuller Unit Root Test(Seasonality=7)				
Type	Rho	Pr < Rho	Tau	Pr < Tau
Zero Mean				
Single Mean				

Dickey-Fuller Unit Root Test Summary				
Variable	Seasonality	Zero Mean	Mean	Trend
SNYSECM	1	NO	NO	NO

Seasonal Dickey-Fuller Unit Root Test Summary				
Variable	Seasonality	Zero Mean	Mean	Trend
SNYSECM	7	NO	NO	

ARIMA Model Specification												
Variable	Functional Transform	Constant	p	d	q	P	D	Q	Seasonality	Model Criterion	Statistic	Status
SNYSECM	NONE	YES	0	0	4	0	0	2	7	RMSE	2.7772	OK

ARIMA Event Selection				
Event Name	Selected	d	D	Status
WHITESALE	REQUIRED	0	0	Not Improved

ARIMA Outlier Selection					
Variable	Type	Obs	Time	Chi-Square	Approx Pr > ChiSq
SNYSECM	LS	677	07NOV1989	334.86	<.0001
	LS	1138	11FEB1991	183.43	<.0001

ARIMA Model Specification After Adjusting for Events and Outliers														
Variable	Functional Transform	Constant	p	d	q	P	D	Q	Seasonality	Outlier	Event	Model Criterion	Statistic	Status
SNYSECM	NONE	YES	0	0	4	0	0	2	7	2	1	RMSE	2.2440	OK

Example 3: Obtaining a Calendar Variable By Using the DATEKEYS Procedure

Features: PROC DATEKEYS statements
 DATEKEYDEF
 DATEKEYKEY
 DATEKEYDATA
 ID
 DATEKEYCALENDAR
 System option
 EVENTDS=

Details

Consider the datekey definitions that are shown in [“Concepts: The DATEKEYS Procedure” on page 528](#). The DATEKEYS procedure can be used to create a calendar variable that indicates the active periods of the datekey definitions that are defined in the data set MyHolidays.

Program

```
options eventds=(nodefaults);

data Year2010;
  do date='01JAN2010'd to '31DEC2010'd;
    output;
  end;
  format date date.;
run;

proc datekeys;
  datekeydef SuperBowl=
    '15JAN1967'd '14JAN1968'd '12JAN1969'd '11JAN1970'd
    '17JAN1971'd '16JAN1972'd '14JAN1973'd '13JAN1974'd '12JAN1975'd
    '18JAN1976'd '09JAN1977'd '15JAN1978'd '21JAN1979'd '20JAN1980'd
    '25JAN1981'd '24JAN1982'd '30JAN1983'd '22JAN1984'd '20JAN1985'd
    '26JAN1986'd '25JAN1987'd '31JAN1988'd '22JAN1989'd '28JAN1990'd
    '27JAN1991'd '26JAN1992'd '31JAN1993'd '30JAN1994'd '29JAN1995'd
    '28JAN1996'd '26JAN1997'd '25JAN1998'd '31JAN1999'd '30JAN2000'd
    '28JAN2001'd '03FEB2002'd '26JAN2003'd '01FEB2004'd '06FEB2005'd
    '05FEB2006'd '04FEB2007'd '03FEB2008'd '01FEB2009'd '07FEB2010'd
    '06FEB2011'd '05FEB2012'd '03FEB2013'd '02FEB2014'd
    / pulse=day;
```



```

datekeydef GoodFriday=Easter / shift=-2 pulse=day;
datekeykey EasterMonday=Easter / shift=1 pulse=day;
datekeydata out=MyHolidays condense;

run;

options eventds=(MyHolidays);

proc datekeys data=Year2010;
    id date interval=day;
    datekeycalendar out=MyHolidaysIn2010;
run;

proc print data=MyHolidaysIn2010 (where=(month(date)=2));

run;

proc print data=MyHolidaysIn2010 (where=(month(date)=4));

run;

```

Program Description

Set the EVENTDS= system option. The EVENTDS= system option specifies the data set that defines the event. The NODEFAULTS option specifies not to use default event definitions. The only events that are used are specified by the event-data-set list.

```
options eventds=(nodefaults);
```

Create a SAS data set. Create the Year2010 data set, which contains a series of dates.

```

data Year2010;
    do date='01JAN2010'd to '31DEC2010'd;
        output;
    end;
    format date date.;
run;

```

Begin the DATEKEYS procedure.

```
proc datekeys;
```

Define a datekey. The DATEKEYDEF statement defines a datekey. PULSE= specifies the interval to be used with the DURATION= option to determine the width of the datekey.

```

datekeydef SuperBowl=
    '15JAN1967'd '14JAN1968'd '12JAN1969'd '11JAN1970'd
    '17JAN1971'd '16JAN1972'd '14JAN1973'd '13JAN1974'd '12JAN1975'd
    '18JAN1976'd '09JAN1977'd '15JAN1978'd '21JAN1979'd '20JAN1980'd
    '25JAN1981'd '24JAN1982'd '30JAN1983'd '22JAN1984'd '20JAN1985'd
    '26JAN1986'd '25JAN1987'd '31JAN1988'd '22JAN1989'd '28JAN1990'd
    '27JAN1991'd '26JAN1992'd '31JAN1993'd '30JAN1994'd '29JAN1995'd
    '28JAN1996'd '26JAN1997'd '25JAN1998'd '31JAN1999'd '30JAN2000'd
    '28JAN2001'd '03FEB2002'd '26JAN2003'd '01FEB2004'd '06FEB2005'd
    '05FEB2006'd '04FEB2007'd '03FEB2008'd '01FEB2009'd '07FEB2010'd

```

```
'06FEB2011'd '05FEB2012'd '03FEB2013'd '02FEB2014'd
/ pulse=day;
```

Define a datekey. The DATEKEYDEF statement defines a datekey named GoodFriday and makes it equal to the datekey timing value named Easter. When you specify the PULSE= option, DURATION= values are set to zero. SHIFT= specifies the number of pulses to shift the timing value. When the SHIFT= option is used, all timing values are shifted.

```
datekeydef GoodFriday=Easter / shift=-2 pulse=day;
```

Alter or create a new datekey. The value of the DATEKEYKEY statement with the datekey name EasterMonday is equal to the datekey named Easter. When Easter is treated as a datekey timing value, only the timing value definition is used, and qualifiers (such as SHIFT or PULSE) are not part of the definition. When Easter is treated as a datekey, the qualifiers that you select (such as SHIFT and PULSE) are part of the new definition. When you specify the PULSE= option, DURATION= values are set to zero. SHIFT= specifies the number of pulses to shift the timing value. When the SHIFT= option is used, all timing values are shifted.

```
datekeykey EasterMonday=Easter / shift=1 pulse=day;
```

Write the datekey to an output data set. The DATEKEYDATA statement writes datekeys to an output data set named MyHolidays. The CONDENSE option specifies that the output data set be condensed. Any variables that contain only default values are omitted from the data set.

```
datekeydata out=MyHolidays condense;
```

Execute the DATEKEYS procedure.

```
run;
```

Set the EVENTSDS= system option. The EVENTSDS system option specifies the data set that defines the event.

```
options eventds=(MyHolidays);
```

Begin the DATEKEYS procedure. The DATA= option names the data set that contains the variables that are used in the ID statement.

```
proc datekeys data=Year2010;
```

Specify the time ID variable. The ID statement names a numeric variable that identifies observations in the data set. If you use the ID statement, then the INTERVAL= option must be used. In this case, INTERVAL= day.

```
id date interval=day;
```

Create a data set that contains variables that indicate the active time periods for datekeys. The DATEKEYCALENDAR OUT= statement specifies the data set that is created, and contains calendar variables that are based on the information that is specified in the ID statement. The DATEKEYCALENDAR OUT= data set contains calendar variables that identify the holidays that are defined in the MyHolidays data set. Values for the month in which of February 2010 are shown [Output 16.93 on page 574](#). Values for the month in which of April 2010 are shown [Output 16.94 on page 575](#). Only the active and inactive periods that correspond to the input time ID are contained in the DATEKEYCALENDAR OUT= data set MyHolidaysIn2010. Because the input time ID has a daily frequency and spans the year 2010, the results are calendar variables for the year 2010.

```
datekeycalendar out=MyHolidaysIn2010;  
run;
```

Write the output for the MyHolidaysIn2010 data set for February. The results are shown in [Output 16.93 on page 574](#).

```
proc print data=MyHolidaysIn2010 (where=(month(date)=2));
```

Execute the DATEKEYS procedure.

```
run;
```

Write the output for the MyHolidaysIn2010 data set for April. The results are shown in [Output 16.94 on page 575](#).

```
proc print data=MyHolidaysIn2010 (where=(month(date)=4));  
run;
```

HTML Output

Output 16.5 *Holiday Calendar from the Daily 2010 Time ID for February*

The SAS System				
Obs	date	SuperBowl	GoodFriday	EasterMonday
32	01FEB2010	0	0	0
33	02FEB2010	0	0	0
34	03FEB2010	0	0	0
35	04FEB2010	0	0	0
36	05FEB2010	0	0	0
37	06FEB2010	0	0	0
38	07FEB2010	1	0	0
39	08FEB2010	0	0	0
40	09FEB2010	0	0	0
41	10FEB2010	0	0	0
42	11FEB2010	0	0	0
43	12FEB2010	0	0	0
44	13FEB2010	0	0	0
45	14FEB2010	0	0	0
46	15FEB2010	0	0	0
47	16FEB2010	0	0	0
48	17FEB2010	0	0	0
49	18FEB2010	0	0	0
50	19FEB2010	0	0	0
51	20FEB2010	0	0	0
52	21FEB2010	0	0	0
53	22FEB2010	0	0	0
54	23FEB2010	0	0	0
55	24FEB2010	0	0	0
56	25FEB2010	0	0	0
57	26FEB2010	0	0	0
58	27FEB2010	0	0	0
59	28FEB2010	0	0	0

Output 16.6 Holiday Calendar from the Daily 2010 Time ID for April**The SAS System**

Obs	date	SuperBowl	GoodFriday	EasterMonday
91	01APR2010	0	0	0
92	02APR2010	0	1	0
93	03APR2010	0	0	0
94	04APR2010	0	0	0
95	05APR2010	0	0	1
96	06APR2010	0	0	0
97	07APR2010	0	0	0
98	08APR2010	0	0	0
99	09APR2010	0	0	0
100	10APR2010	0	0	0
101	11APR2010	0	0	0
102	12APR2010	0	0	0
103	13APR2010	0	0	0
104	14APR2010	0	0	0
105	15APR2010	0	0	0
106	16APR2010	0	0	0
107	17APR2010	0	0	0
108	18APR2010	0	0	0
109	19APR2010	0	0	0
110	20APR2010	0	0	0
111	21APR2010	0	0	0
112	22APR2010	0	0	0
113	23APR2010	0	0	0
114	24APR2010	0	0	0
115	25APR2010	0	0	0
116	26APR2010	0	0	0
117	27APR2010	0	0	0
118	28APR2010	0	0	0
119	29APR2010	0	0	0
120	30APR2010	0	0	0

Example 4: Filtering Data Sets By Using the DATEKEYDSOPT Statement

Features:

- PROC DATEKEYS statements
 - DATEKEYKEY
 - DATEKEYDEF
 - DATEKEYDATA
 - ID
 - DATEKEYDSOPT
 - DATEKEYCALENDAR
- System option
 - EVENTDS=

Details

The DATEKEYS procedure can be used to filter input data sets and create data sets that contain only datekey data that is associated with a specified locale. The DATEKEYDSOPT statement applies to the data sets that are specified in the DATEKEYDATA, DATEKEYPERIODS, and DATEKEYCALENDAR statements.

This example creates the datekey definitions for the following English Canadian holidays: Canada, CanadaObserved, and Boxing. It also creates an English U.S. datekey definition for USINDEPENDENCE. The definition for NewYearsEve does not have a specified locale.

Program

```
options eventds=(nodefaults);

data December2010;
  do date='01DEC2010'd to '31DEC2010'd;
    output;
  end;
  format date date.;
run;

proc datekeys;

  datekeykey Canada / locale=en_CA;
  datekeykey CanadaObserved / locale=en_CA;
  datekeykey Boxing / locale='en_CA';
  datekeykey USINDEPENDENCE / locale=en_US;

  datekeydef NewYearsEve=NEWYEAR / shift=-1 pulse=day;
```

```

datekeydata out=MyHolidays condense;

run;

options eventds=(MyHolidays);

proc datekeys data=December2010;
    id date interval=day;
    datekeydsopt locale=en_CA;
    datekeycalendar out=AllCAHolidaysInDecember2010;
run;

proc datekeys data=December2010;
    id date interval=day;
    datekeydsopt locale=(ONLY)en_CA;
    datekeycalendar out=OnlyCAHolidaysInDecember2010;
run;

proc print data=MyHolidays;
run;

proc print data=AllCAHolidaysInDecember2010;
run;

proc print data=OnlyCAHolidaysInDecember2010;
run;

```

Program Description

Set the EVENTDS= system option. The EVENTDS= system option specifies the data set that defines the event. The NODEFAULTS option specifies not to use default event definitions. The only events that are used are specified by the event-data-set list.

```
options eventds=(nodefaults);
```

Create a SAS data set. Create the December2010 data set, which contains a series of dates:

```

data December2010;
    do date='01DEC2010'd to '31DEC2010'd;
        output;
    end;
    format date date.;
run;

```

Begin the DATEKEYS procedure.

```
proc datekeys;
```

Create a new datekey. The DATEKEYKEY statement creates a new datekey. The LOCALE= option specifies a locale that is associated with the datekey. The locale should be a POSIX locale value.

```
datekeykey Canada / locale=en_CA;
```

```
datekeykey CanadaObserved / locale=en_CA;
datekeykey Boxing / locale='en_CA';
datekeykey USINDEPENDENCE / locale=en_US;
```

Define a datekey. The DATEKEYDEF statement identifies a datekey. The SHIFT= option specifies the number of pulses to shift the timing value. When this option is used, all timing values in the list (including those that are generated by date keywords) are shifted. The PULSE= option specifies the interval to be used.

```
datekeydef NewYearsEve=NEWYEAR / shift=-1 pulse=day;
```

Write the datekey to an output data set. The DATEKEYDATA statement creates an output data set. The CONDENSE option specifies that the output data set be condensed. Any variables that contain only default values are omitted from the data set.

```
datekeydata out=MyHolidays condense;
```

Execute the DATEKEYS procedure.

```
run;
```

Set the EVENTDS= system option. The EVENTDS= system option specifies the data set that defines the event.

```
options eventds=(MyHolidays);
```

Begin the DATEKEYS procedure. The DATEKEYS procedure creates datekeys that are associated with time computations. The DATA= option contains the variables that are used in the ID statement.

```
proc datekeys data=December2010;
```

Specify the ID statement. The ID statement names a numeric variable that identifies observations in the input and output data sets. The ID variable's values are SAS date values. In addition, the ID statement specifies the desired frequency that is associated with the time series.

```
id date interval=day;
```

Create a data set for a specific locale. The DATEKEYDSOPT statement limits the processing of data sets to a specified locale. When you use this statement, only the datekey definitions with a locale value of en_CA, and datekey definitions with no specified locale, are used to create the AllCAHolidaysInDecember2010 data set.

```
datekeydsopt locale=en_CA;
```

Create a data set that contains variables that indicate the active time periods for datekeys. The DATEKEYCALENDAR statement writes variables that indicate the active time periods for datekeys. The OUT= option is the data set that is created, and contains calendar variables based on the information that is specified in the ID statement.

```
datekeycalendar out=AllCAHolidaysInDecember2010;
```

Execute the DATEKEYS procedure.

```
run;
```

Begin the DATEKEYS procedure. The DATA= option contains the variables that are used in the ID statement.

```
proc datekeys data=December2010;
```


Specify the ID statement. The ID statement names a numeric variable that identifies observations in the input and output data sets. The ID variable's values are SAS date values. In addition, the ID statement specifies the desired frequency that is associated with the time series.

```
id date interval=day;
```

Create a data set for a specific locale. The DATEKEYDSOPT statement creates a data set that contains only datekey data that is associated with a specified locale. If you specify LOCALE=(ONLY)en_CA, then only the specified locale is processed for both input and output data sets. The locale should be a POSIX locale value.

```
datekeydsopt locale=(ONLY)en_CA;
```

Create a data set that contains variables that indicate the active time periods for datekeys. The DATEKEYCALENDAR statement writes variables that indicate the active time periods for datekeys. The OUT= option specifies the data set that is created, and contains calendar variables based on the information that is specified in the ID statement.

```
datekeycalendar out=OnlyCAHolidaysInDecember2010;
```

Execute the DATEKEYS procedure.

```
run;
```

Write the MyHolidays data set. For results, see [Output 16.95 on page 580](#).

```
proc print data=MyHolidays;
run;
```

Write the AllCAHolidaysInDecember2010 data set. For results, see [Output 16.96 on page 581](#).

```
proc print data=AllCAHolidaysInDecember2010;
run;
```

Write the OnlyCAHolidaysInDecember2010 data set. For results, see [Output 16.97 on page 582](#).

```
proc print data=OnlyCAHolidaysInDecember2010;
run;
```

HTML Output

Output 16.7 All Holiday Definitions That Are Specified in MyHolidays

The SAS System						
Obs	_LOCALE_	_NAME_	_CLASS_	_KEYNAME_	_PULSE_	_SHIFT_
1	EN_CA	Canada	DATEKEY	CANADA	DAY	0
2	EN_CA	CanadaObserved	DATEKEY	CANADAOBSERVED	DAY	0
3	en_CA	Boxing	DATEKEY	BOXING	DAY	0
4	EN_US	USINDEPENDENCE	DATEKEY	USINDEPENDENCE	DAY	0
5	.	NewYearsEve	DATEKEY	NEWYEAR	DAY	-1

Output 16.8 Holiday Calendar for All Canadian Holidays in December 2010**The SAS System**

Obs	date	Canada	CanadaObserved	Boxing	NewYearsEve
1	01DEC2010	0	0	0	0
2	02DEC2010	0	0	0	0
3	03DEC2010	0	0	0	0
4	04DEC2010	0	0	0	0
5	05DEC2010	0	0	0	0
6	06DEC2010	0	0	0	0
7	07DEC2010	0	0	0	0
8	08DEC2010	0	0	0	0
9	09DEC2010	0	0	0	0
10	10DEC2010	0	0	0	0
11	11DEC2010	0	0	0	0
12	12DEC2010	0	0	0	0
13	13DEC2010	0	0	0	0
14	14DEC2010	0	0	0	0
15	15DEC2010	0	0	0	0
16	16DEC2010	0	0	0	0
17	17DEC2010	0	0	0	0
18	18DEC2010	0	0	0	0
19	19DEC2010	0	0	0	0
20	20DEC2010	0	0	0	0
21	21DEC2010	0	0	0	0
22	22DEC2010	0	0	0	0
23	23DEC2010	0	0	0	0
24	24DEC2010	0	0	0	0
25	25DEC2010	0	0	0	0
26	26DEC2010	0	0	1	0
27	27DEC2010	0	0	0	0
28	28DEC2010	0	0	0	0
29	29DEC2010	0	0	0	0
30	30DEC2010	0	0	0	0
31	31DEC2010	0	0	0	1

Output 16.9 Holiday Calendar for Only Canadian Holidays in December 2010

The SAS System				
Obs	date	Canada	CanadaObserved	Boxing
1	01DEC2010	0	0	0
2	02DEC2010	0	0	0
3	03DEC2010	0	0	0
4	04DEC2010	0	0	0
5	05DEC2010	0	0	0
6	06DEC2010	0	0	0
7	07DEC2010	0	0	0
8	08DEC2010	0	0	0
9	09DEC2010	0	0	0
10	10DEC2010	0	0	0
11	11DEC2010	0	0	0
12	12DEC2010	0	0	0
13	13DEC2010	0	0	0
14	14DEC2010	0	0	0
15	15DEC2010	0	0	0
16	16DEC2010	0	0	0
17	17DEC2010	0	0	0
18	18DEC2010	0	0	0
19	19DEC2010	0	0	0
20	20DEC2010	0	0	0
21	21DEC2010	0	0	0
22	22DEC2010	0	0	0
23	23DEC2010	0	0	0
24	24DEC2010	0	0	0
25	25DEC2010	0	0	0
26	26DEC2010	0	0	1
27	27DEC2010	0	0	0
28	28DEC2010	0	0	0
29	29DEC2010	0	0	0
30	30DEC2010	0	0	0
31	31DEC2010	0	0	0

References

- Montes, M. J. "Calculation of the Ecclesiastical Calendar." 2001. Available at <http://www.smart.net/~mmontes/ec-cal.html>.
- Montes, M. J. "Algorithm for Calculating the Date of Easter in the Gregorian Calendar." 2001. Available at <http://www.smart.net/~mmontes/nature1876.html>.

DELETE Procedure

Overview: DELETE Procedure	585
What Does the DELETE Procedure Do?	585
Concepts: DELETE Procedure	586
Concepts: DELETE Procedure	586
Syntax: DELETE Procedure	586
PROC DELETE Statement	587
Examples: DELETE Procedure	590
Example 1: Deleting Several SAS Data Sets	590
Example 2: Deleting the Base Version and All Historical Versions	591
Example 3: Deleting the Base Version and Renaming the Youngest Historical Version to the Base Version	591
Example 4: Deleting a Version with an Absolute Number	592
Example 5: Deleting All Historical Versions and Leaving the Base Version	592
Example 6: Using the MEMTYPE= Option	593
Example 7: Using the ENCRYPTKEY= Option	593
Example 8: Using the ALTER= Option	594
Example 9: Using the DATA= List Feature	595
Example 10: Using the LIBRARY= Option	595
Example 11: Using the LIBRARY= Option and List Feature	596

Overview: DELETE Procedure

What Does the DELETE Procedure Do?

The DELETE procedure deletes members in a SAS library. Use PROC DELETE to do the following:

- delete either permanent or temporary SAS files

- delete loaded CAS tables from a caslib. It does not delete the original files from the data source specified by the caslib.
- delete a list of data sets with the same name and a numeric suffix, such as

```
proc delete data=x1-x3;
run;
```
- delete a member type.
- delete generation data sets using the GENNUM= option.
- delete AES-encrypted data sets when using GENNUM=ALL and ENCRYPTKEY= options.

Concepts: DELETE Procedure

Concepts: DELETE Procedure

One of the benefits of using PROC DELETE instead of the DELETE statement in the DATASETS procedure is that it does not use the in-memory directory to delete SAS data sets. As a result, the DELETE procedure is faster.

Syntax: DELETE Procedure

Valid in: Compute Server, CAS

Tip: You can perform similar functions with the DELETE statement in the DATASETS procedure.

PROC DELETE DATA=SAS-file(s) <options>;

Statement	Task	Example
PROC DELETE	Delete SAS files from SAS libraries	Ex. 1 , Ex. 2 , Ex. 3 , Ex. 4 , Ex. 5

PROC DELETE Statement

Deletes SAS files from SAS libraries.

Syntax

```
PROC DELETE <LIBRARY=libref> DATA=SAS-file(s)
    (<GENNUM=ALL | HIST | REVERT | integer>
    <MEMTYPE=member-type>
    <ENCRYPTKEY=key-value>
    <ALTER=alter-password>);
```

Summary of Optional Arguments

ALTER=*alter-password*

provides the Alter password for any alter-protected SAS files.

ENCRYPTKEY=*key-value*

The ENCRYPTKEY= option unlocks the data sets that are protected by an AES-encrypted key value.

GENNUM=ALL | HIST | REVERT | *integer*

restricts processing for generation data sets.

LIBRARY=*libref*

specifies the SAS library that contains members to be deleted.

MEMTYPE=(*member-type(s)*)

restricts deletion to a certain type of SAS file.

Required Argument

DATA= SAS-file(s)

specifies one or more SAS files that you want to delete.

Note: You can also use a numbered range list. For more information, see [“Data Set Name Lists”](#) in *SAS Programmer's Guide: Essentials*. You cannot use a colon list.

TIP If you want to delete all files in a library, use the PROC DATASETS KILL option. Use PROC DATASETS LIB=*library name* KILL to delete all files including catalogs. For more information, see [“KILL”](#) on page 385.

Optional Arguments

ALTER=alter-password

provides the Alter password for any alter-protected SAS files.

See [“Using Passwords with the DATASETS Procedure” on page 467](#)

ENCRYPTKEY=key-value

The ENCRYPTKEY= option unlocks the data sets that are protected by an AES-encrypted key value. The ENCRYPTKEY= option is needed only when a data set must be opened.

See [“Example 7: Using the ENCRYPTKEY= Option” on page 593](#)

GENNUM=ALL | HIST | REVERT | integer

restricts processing for generation data sets. You use the option in parentheses after the name of each SAS file. The following is a list of valid values:

ALL	refers to the base version and all historical versions in a generation group.
HIST	refers to all historical versions, but excludes the base version in a generation group.
REVERT 0	deletes the base version and changes the most current historical version, if it exists, to the base version.
integer	is a number that references a specific version from a generation group. Specifying a positive number is an absolute reference to a specific generation number that is appended to a data set's name (that is, gennum=2 specifies MYDATA#002).

See [“Restricting Processing for Generation Data Sets” on page 468](#)

Examples [“Example 1: Deleting Several SAS Data Sets” on page 590](#)

[“Example 2: Deleting the Base Version and All Historical Versions” on page 591](#)

[“Example 3: Deleting the Base Version and Renaming the Youngest Historical Version to the Base Version” on page 591](#)

[“Example 4: Deleting a Version with an Absolute Number” on page 592](#)

[“Example 5: Deleting All Historical Versions and Leaving the Base Version” on page 592](#)

LIBRARY=libref

specifies the SAS library that contains members to be deleted.

Alias LIB=

MEMTYPE=(*member-type(s)*)

restricts deleting one or more member types. For example, if you have a data set and a catalog named MyFile in the MyLib library and you want to delete only the catalog, then use the MEMTYPE= option.

```
proc delete lib=MyLib data=MyFile (memtype=catalog);
run;
```

ACCESS

access descriptor files (created by SAS/ACCESS software)

CATALOG

SAS catalogs

DATA

SAS data files

FDB

financial database

MDDDB

multidimensional database

PROGRAM

stored compiled SAS programs

VIEW

SAS views

Aliases MTYPE=

 MT=

Default DATA

Example [“Example 6: Using the MEMTYPE= Option” on page 593](#)

Details

Working with Generation Groups

When you are working with generation groups, you can use PROC DELETE to delete the following versions:

- delete the base version and all historical versions. See [“Example 2: Deleting the Base Version and All Historical Versions” on page 591](#).
- delete the base version and rename the youngest historical version to the base version. See [“Example 3: Deleting the Base Version and Renaming the Youngest Historical Version to the Base Version” on page 591](#).
- delete an absolute version. See [“Example 4: Deleting a Version with an Absolute Number” on page 592](#).
- delete all historical versions and leave the base version. See [“Example 5: Deleting All Historical Versions and Leaving the Base Version” on page 592](#).

Deleting Specific Member Types

With the MEMTYPE= option, you can specify the member type that you want to delete. The default for the MEMTYPE= option is DATA. For more information, see “MEMTYPE=(member-type(s))” on page 589.

Working with Integrity Constraints

You cannot use the DELETE procedure to delete a data file that has a foreign key integrity constraint or a primary key with foreign key references. For data files that have foreign keys, you must remove the foreign keys before you delete the data file. For data files that have primary keys with foreign key references, you must remove the foreign keys that reference the primary key before you delete the data file.

Examples: DELETE Procedure

Example 1: Deleting Several SAS Data Sets

Features:	PROC DELETE statement options
	DATA=
	GENNUM=

Details

This example demonstrates the following tasks:

- deletes data sets from a library
- deletes all historical versions of each data set

Program

Delete SAS data sets named A, B, and C from a SAS library named MyLib. The GENNUM= option deletes all the historical versions for each of the data sets.

```
proc delete data=MyLib.A MyLib.B MyLib.C (gennum=all);  
run;
```

Example 2: Deleting the Base Version and All Historical Versions

Features: PROC DELETE statement options
DATA=
GENNUM=

Details

This example demonstrates the following tasks:

- deletes a data set from a library
- deletes all historical versions of the data set

Program

Deletes the data set named MyLib.A and all the historical versions.

```
proc delete data=MyLib.A (gennum=all);  
run;
```

Example 3: Deleting the Base Version and Renaming the Youngest Historical Version to the Base Version

Features: PROC DELETE statement options
DATA=
GENNUM=

Details

This example demonstrates the following tasks:

- deletes a data set from a library

- renames the youngest historical version

The following statement deletes the data set named MyLib.A and renames the youngest historical version.

Program

Deletes the data set named MyLib.A and renames the youngest historical version.

```
proc delete data=MyLib.A(gennum=revert1);
run;
```

Example 4: Deleting a Version with an Absolute Number

Features:	PROC DELETE statement options DATA= GENNUM=
-----------	---

Details

This example deletes the first historical version of the data set.

Program

GENNUM= option deletes the first historical version of the data set names MyLib.A. You use GENNUM=*integer* to select the historical version that you want to delete.

```
proc delete data=MyLib.A(gennum=1);
run;
```

Example 5: Deleting All Historical Versions and Leaving the Base Version

Features:	PROC DELETE statement options DATA= GENNUM=
-----------	---

Details

This example deletes all historical versions except the base version of a data set.

Program

Use the GENNUM=HIST option to delete all historical versions and retain the base version of the data set MyLib.A.

```
proc delete data=MyLib.A (gennum=hist);  
run;
```

Example 6: Using the MEMTYPE= Option

Features: PROC DELETE statement options
DATA=
MEMTYPE=

Details

This example deletes the CATALOG file in a specific SAS library.

Program

The MEMTYPE= option names the type of file to delete in a SAS library. If you have other member types named MyFile in the MyLib library, they will not be deleted.

```
proc delete lib=MyLib data=MyFile (memtype=catalog);  
run;
```

Example 7: Using the ENCRYPTKEY= Option

Features: PROC DELETE statement options
DATA=

```
ENCRYPTKEY=  
GENNUM=
```

Details

This example demonstrates the following tasks:

- unlocks an AES encrypted data set
- deletes all historical versions and the base AES data set named MyLib.A.

Program

Deletes the base AES data set named MyLib.A and all historical versions. The ENCRYPTKEY= option must be used if the data set is AES encrypted.

```
proc delete data=MyLib.A (gennum=ALL encryptkey=key-value);  
run;
```

Example 8: Using the ALTER= Option

Features: PROC DELETE statement options
ALTER=
DATA=

Details

This example deletes a password protected data set.

Program

Deletes a password protected data set named MyLib.A. If the data set is password protected, you must supply the password.

```
proc delete data=MyLib.A (alter=alter-password);  
run;
```

Example 9: Using the DATA= List Feature

Features: PROC DELETE statement options
DATA=

Program

Deletes multiple data sets named X1, X2, X3, X4, and X5. To use the list feature, the data sets must have the same name and end with a numeric suffix. If a LIBRARY= option is not specified, the data sets are deleted from the Work library.

```
proc delete data=X1-X5;  
run;
```

Example 10: Using the LIBRARY= Option

Features: PROC DELETE statement options
DATA=
LIB=

Details

The following statement deletes a data set from a specific SAS library.

Program

Deletes the A data set that is in the specified SAS library named MyLib. The alias for the LIBRARY= option is LIB=.

```
proc delete lib=MyLib data=A;  
run;
```

Example 11: Using the LIBRARY= Option and List Feature

Features: PROC DELETE statement options
DATA=
LIB=

Details

This example deletes data sets X1, X2, X3, X4, and X5 in the specified SAS library named MyLib. The alias for the LIBRARY= option is LIB=.

Note: The data sets must end with a numeric suffix.

Program

Deletes data sets X1, X2, X3, X4, and X5 from the specified SAS library named MyLib. When using the list feature, all the data sets must have the same name and end with a numeric suffix. The alias for the LIBRARY= option is LIB=.

```
proc delete lib=MyLib data=X1-X5;  
run;
```

DS2 Procedure

Overview: DS2 Procedure	597
What Does the DS2 Procedure Do?	598
Concepts: DS2 Procedure	598
Benefits of the DS2 Language	598
Data Source Support	599
Syntax: DS2 Procedure	600
PROC DS2 Statement	600
RUN CANCEL Statement	609
Usage: DS2 Procedure	610
Connecting to SAS and DBMS Data Sources	610
Connecting to the CAS Server	611
DS2 Functionality in SAS Cloud Analytic Services	613
RUN-Group Processing	614
Applying DS2 Table Options	614
Using Macro Variables in a Literal String	615
DS2 Automatic Variables	615
Passwords	616
Encryption	617
DS2 Data Type Support for SAS Data Sets	618
DS2 Data Type Support for CAS Tables	619
Examples: DS2 Procedure	620
Example 1: Introducing DS2 Code	620
Example 2: Creating a SAS Data Set	622
Example 3: Terminating the Current Step in Line Prompt Mode	624
Example 4: Routing Data to Tables Based on Values	626
Example 5: Run a DS2 Program in CAS	628

Overview: DS2 Procedure

What Does the DS2 Procedure Do?

The DS2 procedure enables you to submit SAS DS2 language programs. Using the DS2 procedure, you can submit DS2 programs to SAS and third-party data sources. Or, you can submit DS2 programs to the CAS server.

DS2 is a SAS programming language that is appropriate for advanced data manipulation. DS2 shares core features with the SAS DATA step. DS2 enhances the DATA step by adding variable scoping, ANSI SQL data types, predefined packages, and user-defined methods and packages. The DS2 SET statement accepts embedded FedSQL syntax, and the runtime-generated queries can exchange data interactively between DS2 and any supported database. This allows SQL preprocessing of input tables, which effectively combines the power of the two languages. For more information about the DS2 language, see [SAS DS2 Language Reference](#) and [SAS DS2 Programmer's Guide](#).

Note: The DS2 procedure does not use the CAS engine to connect to CAS. In order to submit DS2 programs to the CAS server, you must specify a caslib that was assigned using the CASLIB statement. For more information, see [“Connecting to the CAS Server” on page 611](#).

The DS2 language supports the same functionality on the CAS server as it does for SAS and third-party data sources, with a few exceptions.

Concepts: DS2 Procedure

Benefits of the DS2 Language

DS2 programs are written for applications that

- require the precision that results from using the additional data types that are available with the language

- benefit from using the new expressions or write methods or packages available in the DS2 syntax
- need to execute the SAS FedSQL language from within the DS2 program
- can execute inside and outside of SAS Viya (for example, on SAS Federation Server)
- take advantage of threaded processing.

Data Source Support

The following data sources are supported with the DS2 procedure.

- Amazon Redshift
- CAS server
- DB2 in the Linux operating environment
- Google BigQuery
- Greenplum
- Hadoop (Hive)
- Impala
- JDBC-compliant databases
- Microsoft SQL Server
- MongoDB
- MySQL
- Netezza
- ODBC-compliant databases
- Oracle
- PostgreSQL
- Salesforce
- SAP
- SAP HANA
- SAP IQ
- SAS data sets
- SAS Scalable Performance Data Engine data sets
- SAS Scalable Performance Data Server tables
- SingleStore
- Snowflake
- Spark

- Teradata
- Vertica
- Yellowbrick

You must have appropriate SAS/ACCESS software configured in order to access and manipulate DBMS data sources with PROC DS2. You must have appropriate SAS data connector software configured to load SAS and third-party data onto the CAS server for processing with PROC DS2.

Syntax: DS2 Procedure

```
PROC DS2 <connection-options><processing-options>;
...DS2 language statements
RUN;
RUN CANCEL;
QUIT;
```

Statement	Task	Example
PROC DS2	Specify that the subsequent input is DS2 language statements.	Ex. 1, Ex. 2, Ex. 3, Ex. 4, Ex. 5
RUN CANCEL	Cancel the previous DS2 language statements.	Ex. 3

PROC DS2 Statement

Specifies that the subsequent input is DS2 language statements.

- Restriction:

When the HTTP access method is in a locked-down state, the DS2 built-in package, HTTP, is not available. Your server administrator can re-enable this access method so that it is accessible in the locked-down state. For more information, see [“LOCKDOWN” in SAS Programmer’s Guide: Essentials](#).
- Requirement:

Follow the PROC DS2 statement with DS2 language statements. See [SAS DS2 Language Reference](#) for information about DS2 language statements. Also see [SAS DS2 Programmer’s Guide](#).
- Interactions:

The DS2 procedure requires the RUN statement to submit DS2 statements. That is, SAS reads the program statements that are associated with one task until it reaches a RUN statement. You can specify multiple RUN groups. To signal the end of a program, specify QUIT followed by a semicolon. See the [“RUN CANCEL Statement” on page 609](#).

By default, the DS2 procedure processes nonexistent values in CHAR(*n*) and DOUBLE columns as SAS missing values. To request that nonexistent values are processed as ANSI SQL null values, specify MODE=ANSI.

When creating tables with PROC DS2, note that by default, you cannot overwrite an existing table. To specify that an output table should be deleted before a replacement output table is created, use the OVERWRITE=YES table option. For more information, see [“OVERWRITE= Table Option” in SAS DS2 Language Reference](#).

Supports:	Threaded processing
Note:	Unless noted otherwise, all procedure options are supported for DS2 programs that run on the SAS Compute Server and DS2 programs that run in CAS.
Examples:	“Example 1: Introducing DS2 Code” on page 620 “Example 2: Creating a SAS Data Set” on page 622 “Example 3: Terminating the Current Step in Line Prompt Mode” on page 624 “Example 4: Routing Data to Tables Based on Values” on page 626 “Example 5: Run a DS2 Program in CAS” on page 628

Syntax

PROC DS2 <options>;

Summary of Optional Arguments

Connection

LIBS=*libref* | (*libref1libref2...librefn*)

restricts the data source connection to the specified libref or librefs, and the default library.

SESSREF=*session-reference*

specifies to run the DS2 statements in a CAS session. The CAS session is identified by its session name.

SESSUUID=*"session-uuid"*

specifies to run the DS2 statements in a CAS session. The CAS session is identified by its universally unique identifier (UUID).

General Processing

ANSIMODE

specifies that nonexistent values in fixed-length character columns and DOUBLE columns are processed as ANSI SQL null values.

ERRORSTOP

NOERRORSTOP

specifies whether the procedure stops executing if it encounters an error.

LABEL

NOLABEL

specifies whether to use the column label or the column name as the column heading.

LOGICAEXPR=STANDARD | OPTIMIZED

specifies how logical AND and OR expressions are evaluated.

MEMSIZE=n | nM | nG

specifies a limit for the amount of memory that is used for an underlying query (such as a SELECT statement), so that allocated memory is available to support other PROC DS2 operations.

MODE=ANSI | SAS

specifies whether nonexistent values in fixed-length character columns and DOUBLE columns are processed as SAS missing values or as ANSI NULL values.

NUMBER

specifies to include a column named Row, which is the row (observation) number of the data as the rows are retrieved.

STIMER

specifies to write a subset of system performance statistics, such as time-elapsed statistics, to the SAS log.

TREEALGORITHM=ITERATIVE | RECURSIVE

changes the default algorithm that the DS2 compiler uses to build the syntax tree.

XCODE=ERROR | WARNING | IGNORE

controls the behavior of the SAS session when an NLS transcoding failure occurs.

Message Control

MSGLIMIT=n | MIN | MAX

specifies the maximum number of error, warning, and note messages that can be written to the SAS log during the execution of the DS2 program.

MSGORDER=STANDARD | TEMPORAL

specifies whether DS2 writes error, warning, and note messages to the SAS log as they are produced or after the DS2 program completes.

REPORTLINE

NOREPORTLINE

specifies whether log line numbers are reported in DS2 program execution messages.

SCOND=WARNING | NONE | NOTE | ERROR

specifies the level of messages that PROC DS2 displays in the SAS log for the DS2 variable declaration strict mode, which requires that every variable must be declared in the DS2 program.

Optional Arguments

ANSIMODE

specifies that nonexistent values in fixed-length character columns and DOUBLE columns are processed as ANSI SQL null values. All other data types use ANSI NULL semantics all of the time.

- Default** Nonexistent values are processed as SAS missing values, which is consistent with the SAS programming runtime environment.
- Important** The `MODE=` procedure option supercedes the `ANSIMODE` option.

ERRORSTOP | NOERRORSTOP

specifies whether the procedure stops executing if it encounters an error. `ERRORSTOP` instructs the procedure to stop executing the statements but to continue checking the syntax after it has encountered an error.

`NOERRORSTOP` instructs the procedure to execute the statements and to continue checking the syntax after an error occurs.

- Default** `NOERRORSTOP` in an interactive SAS session; `ERRORSTOP` in a batch or noninteractive session
- Restriction** `ERRORSTOP` has an effect only when SAS is running in a batch or noninteractive execution mode.
- Tip** `NOERRORSTOP` is useful if you want a batch job to continue executing procedure statements after an error is encountered.

LABEL | NOLABEL

specifies whether to use the column label or the column name as the column heading. When a column does not have a label, the procedure uses the column's name as the column heading.

- Default** `LABEL`
- Interaction** When a column alias is used, the column alias supercedes the label or column name as the column heading.

LIBS=libref | (libref1libref2...librefn)

restricts the data source connection to the specified libref or librefs, and the default library.

The `LIBS=` option is useful when many `LIBNAME` statements are defined in a SAS session to limit the scope of the DS2 program to only the `LIBNAME` statement(s) to which the program applies. Use of the option can also avoid duplicate catalog errors when connecting to data sources that support native catalogs, such as Netezza.

When more than one libref is specified, enclose the librefs within parentheses. Separate each libref with a space.

- Alias** `LIBNAMES=`
- Restriction** `LIBS=` is not supported on the CAS server. If both `LIBS=` and `SESSREF=` (or `SESSUID=`) are specified in the procedure statement, `LIBS=` is ignored.
- Interactions** Specifying `LIBS=` does not preclude the need to specify librefs in your DS2 program, even when only one libref is specified in `LIBS=`. If you do not specify a libref with your table names, the default

library is used. For more information, see [“Connecting to SAS and DBMS Data Sources” on page 610](#).

If you specify LIBS= on the procedure statement more than once, only the libref(s) specified in the last instance of the procedure option are applied.

Tip If you want to know how LIBS= affects library assignments, set the MSGLEVEL=i system option before running a PROC DS2 request with LIBS=. The option will produce Include and Ignore messages for each of the LIBNAME statements that are processed in the procedure request.

Example The following PROC DS2 procedure statement specifies to create a data source connection that uses two of four librefs that are assigned in the SAS session and the default library.

```
proc ds2 libs=(mylib3 mylib4);
```

LOGICALEXPR=STANDARD | OPTIMIZED

specifies how logical AND and OR expressions are evaluated.

STANDARD specifies that DS2 may evaluate both operands of a logical AND or OR expression before returning a result.

OPTIMIZED specifies that DS2 optimize evaluation of the operands of logical AND and OR expressions by returning a result that is based on the first operand when possible.

Default STANDARD

Interaction The evaluation of logical AND and OR expressions can also be controlled with the DS2_OPTIONS statement. When a LOGICALEXPR value is set in the DS2_OPTIONS statement, that value takes precedence over the procedure option for the DATA, PACKAGE, or THREAD block on which it is specified. For more information, see the [DS2_OPTIONS statement](#).

Tip Bypassing evaluation of the second operand can avoid calculations that would otherwise be invalid.

See [“Short-Circuit Evaluation in DS2” in SAS DS2 Programmer’s Guide](#)

MEMSIZE=n | nM | nG

specifies a limit for the amount of memory that is used for an underlying query (such as a SELECT statement), so that allocated memory is available to support other PROC DS2 operations. Specify the memory limit in multiples of 1 (bytes); 1,048,576 (megabytes); or 1,073,741,824 (gigabytes). For example, the value 23M specifies 24,117,248 bytes of memory. The value 16G specifies 17,179,869,184 bytes of memory.

Default The procedure optimizes the setting based on the amount of memory on the host.

Note On the CAS server, MEMSIZE= specifies the memory for a single worker.

Tip Generally, specifying a memory limit is not necessary unless DS2 reports a memory problem error.

MODE=ANSI | SAS

specifies whether nonexistent values in fixed-length character columns and DOUBLE columns are processed as SAS missing values or as ANSI NULL values. All other data types use ANSI NULL semantics all of the time.

Default SAS

Interaction A preference for handling nonexistent data can also be specified in the DS2_OPTIONS statement. When a MODE= value is set in the DS2_OPTIONS statement, the value in the statement overrides the value in the procedure option for the program block on which it is specified. For more information, see the [DS2_OPTIONS statement](#).

Note It is important to understand the differences between nulls and missing values, or data can be lost. For information about processing differences, see “[DS2 Modes for Nonexistent Data](#)” in *SAS DS2 Programmer’s Guide*.

MSGLIMIT=n | MIN | MAX

specifies the maximum number of error, warning, and note messages that can be written to the SAS log during the execution of the DS2 program.

n specifies that the first *n* error, warning, and note messages that are produced during the execution of the DS2 program are written to the log. When the message limit is reached, DS2 writes an error message to the SAS log stating that the limit was reached. The program continues executing to completion; however, subsequent error, warning, and note messages are not written to the SAS log.

MIN specifies that no error, warning, or note messages that are produced during the execution of the DS2 program are written to the log. Specifying MIN is the same as specifying MSGLIMIT=0.

MAX specifies that all error, warning, and note messages that are produced during the execution of the DS2 program are written to the log.

Default 1,024 messages for each execution of a program

Restriction The MSGLIMIT= option is valid only when MSGORDER=TEMPORAL is set.

Interaction The number of messages can also be controlled with the DS2_OPTIONS statement. When a MSGLIMIT= value is set in the DS2_OPTIONS statement, that value overrides the setting of the procedure option.

Notes *n* is an integer that can be specified in terms of 1 (*n*); 1,024 (*n*K); 1,048,576 (*n*M); or 1,073,741,824 (*n*G). For example, the value 8 specifies eight messages, and the value 8K specifies 8,192 messages.

The MSGLIMIT= option has no effect on error messages that are produced during DS2 program compilation, messages that are produced by the PUT statement, and messages that are written to the SAS logging facility.

Tip Messages that are suppressed by MSGLIMIT= can be captured by enabling the App.TableServices.DS2.Runtime.Log logger. Enable the logger at level INFO before running your program.

MSGORDER=STANDARD | TEMPORAL

specifies whether DS2 writes error, warning, and note messages to the SAS log as they are produced or after the DS2 program completes.

STANDARD specifies that DS2 writes the messages after the DS2 program completes. PUT statement messages precede the diagnostic messages in the log.

TEMPORAL specifies that DS2 writes the messages as they are produced by the DS2 program. PUT statement messages are interspersed with the diagnostic messages in the log.

Default STANDARD

Interaction Message order can also be controlled with the DS2_OPTIONS statement. When a MSGORDER= value is set in the DS2_OPTIONS statement, that value overrides the setting of the procedure option.

Tip Temporal message ordering can aid DS2 program debugging by reporting PUT statement messages and error, warning, and note messages in the order that they are produced. You can insert PUT statements at strategic points in your DS2 program's logic flow to assist in diagnosing the cause of error conditions that produce error, warning, or note messages.

NUMBER

specifies to include a column named Row, which is the row (observation) number of the data as the rows are retrieved.

Default No row numbers.

REPORTLINE | NOREPORTLINE

specifies whether log line numbers are reported in DS2 program execution messages. The log line numbers are included in error, warning, and note messages and identify the location of the DS2 statement that was being executed when the message was generated in the log. Line numbers that are reported for stored packages and threads are the line numbers that are offset from the start of the package or thread block rather than SAS log line numbers.

NOREPORTLINE suppresses generation of the line numbers.

Default	REPORTLINE
Interaction	Line numbers in program execution messages can also be controlled with the DS2_OPTIONS statement. When a REPORTLINE value is set in the DS2_OPTIONS statement, that value takes precedence over the DS2 procedure option. For more information, see the DS2_OPTIONS statement .
Note	NOREPORTLINE does not affect line numbers for DS2 compilation messages.

SCOND=WARNING | NONE | NOTE | ERROR

specifies the level of messages that PROC DS2 displays in the SAS log for the DS2 variable declaration strict mode, which requires that every variable must be declared in the DS2 program. For more information about the DS2 variable declaration strict mode, see the *SAS DS2 Language Reference*.

WARNING

writes warning messages to the SAS log.

NONE

no messages are written to the SAS log.

NOTE

writes notes to the SAS log.

ERROR

writes error messages to the SAS log.

Default	The default is determined by the DS2SCOND= system option. The default for DS2SCOND= is WARNING. For information about the DS2SCOND= system option, see <i>SAS DS2 Language Reference</i> .
Interactions	Specifying the SCOND= option in the PROC DS2 statement takes precedence over the DS2SCOND= system option. The DS2 variable declaration strict mode setting can also be controlled with the DS2_OPTIONS statement. When an SCOND value is set in the DS2_OPTIONS statement, that value takes precedence over the DS2 procedure option for the DATA, PACKAGE, or THREAD block on which it is specified. For more information about precedence rules, see the DS2_OPTIONS statement.

SESSREF=session-reference

specifies to run the DS2 statements in a CAS session. The CAS session is identified by its session name.

Requirements	A CAS server must be configured for your system. The CAS session must have been previously established by using the CAS statement. If the specified CAS session does not exist, the procedure terminates.
--------------	---

Interaction Use SESSREF= (or SESSUUID=) to connect to the CAS session. If both options are specified, the last option in the procedure statement is applied.

See [“Connecting to the CAS Server” on page 611](#)

[“DS2 Functionality in SAS Cloud Analytic Services” on page 613](#)

[“Example 5: Run a DS2 Program in CAS” on page 628](#)

SESSUUID=*“session-uuid”*

specifies to run the DS2 statements in a CAS session. The CAS session is identified by its universally unique identifier (UUID).

Requirements A CAS server must be configured for your system.

The CAS session must have been previously established by using the CAS statement. The CAS statement generates a UUID value. If the specified CAS session does not exist, the procedure terminates.

Interaction Use SESSUUID= (or SESSREF=) to connect to the CAS session. If both options are specified, the last option in the procedure statement is applied.

See [“Connecting to the CAS Server” on page 611](#)

[“DS2 Functionality in SAS Cloud Analytic Services” on page 613](#)

Example Here is a PROC DS2 procedure statement that specifies SESSUUID=:

```
proc ds2 sessuuid="76904741-fb09-554d-a8de-6cbce2a0e0e5";
```

The UUID value can be enclosed in single or double quotation marks.

STIMER

specifies to write a subset of system performance statistics, such as time-elapsed statistics, to the SAS log. When STIMER is in effect, the procedure writes to the SAS log a list of computer resources used for each step and the entire SAS session.

Default No performance statistics are written to the SAS log.

Interaction If the SAS system option FULLSTIMER is in effect, the complete list of computer resources is written to the SAS log.

TREEALGORITHM=ITERATIVE | RECURSIVE

changes the default algorithm that the DS2 compiler uses to build the syntax tree. When you are using the RECURSIVE option and you encounter a stack overflow error, changing the algorithm can sometimes resolve stack overflow errors.

Defaults SAS programming runtime environment: RECURSIVE

CAS: ITERATIVE

Interaction The default tree walker algorithm can also be changed in the DS2_OPTIONS statement. When a TREEALGORITHM= value is set in the DS2_OPTIONS statement, the value in the statement overrides the value in the procedure option for the program block on which it is specified. For more information, see the [DS2_OPTIONS statement](#).

XCODE=ERROR | WARNING | IGNORE

controls the behavior of the SAS session when an NLS transcoding failure occurs. Transcoding failures can occur during row input or output operations, or during string assignment. Transcoding is the process of converting character data from one encoding to another encoding.

ERROR

specifies that a run-time error occurs, which causes row processing to halt. An error message is written to the SAS log. This is the default behavior.

WARNING

specifies that the incompatible character is set to a substitution character. A warning message is written to the SAS log.

IGNORE

specifies that the incompatible character is set to a substitution character. No messages are written to the SAS log.

Default ERROR

RUN CANCEL Statement

Cancels the previous DS2 language statements.

Tip: The RUN CANCEL statement is useful for canceling processing of a portion of a program in the event of an error.

Example: [“Example 3: Terminating the Current Step in Line Prompt Mode” on page 624](#)

Syntax

RUN CANCEL;

Usage: DS2 Procedure

Connecting to SAS and DBMS Data Sources

You connect PROC DS2 to a SAS or DBMS data source by specifying a libref.

- 1 First, submit LIBNAME statements that specify the SAS and SAS/ACCESS engines for the data sources that you want to connect to. For information about how to define a LIBNAME statement for a given data source, see these resources:

SAS data sets

[SAS Global Statements: Reference](#)

Relational DBMS data sources

[SAS/ACCESS for Relational Databases: Reference](#)

MongoDB and Salesforce

[SAS/ACCESS for Nonrelational Databases: Reference](#)

SPD Engine data sets

[SAS Scalable Performance Data Engine: Reference](#)

SPD Server tables

[SAS Scalable Performance Data Server: User's Guide](#)

Note: The library paths that you specify in the LIBNAME statements must be known to the SAS Compute Server.

Note: Do not use a CAS engine libref with PROC DS2. See [“Connecting to the CAS Server” on page 611](#) for instructions to run DS2 programs to the CAS server.

- 2 In your DS2 programs, use a two-part name in the form *libref.table-name* to refer to tables. The libref tells the DS2 language where to create or locate a table.
- 3 Then, submit PROC DS2.

This example illustrates how PROC DS2 accesses a data source by using the attributes of a previously assigned libref.

```
libname myfiles v9 'library-path'; 1

proc ds2;
  data myfiles.table1; 2
```



```

      dcl double j j2;
      method run();
        do j = 1 to 1000;
          j2 = 2*j;
          output;
        end;
      end;
    enddata;
  run;
quit;

```

- 1 The LIBNAME statement assigns the libref Myfiles and then specifies the physical location for the SAS data set.
- 2 The DS2 program then creates the SAS data set Myfiles.Table1. The program uses a name in the form *libref.table-name* to set the location of the output table.

When you submit a program with the DS2 procedure, the procedure builds a data source connection string internally that includes all active librefs. The procedure then sends this internal connection string to the DS2 program. The DS2 language uses the librefs in the program to locate properties for the specific data source connection from within this internal connection string.

This connection string primarily includes libref attributes for data source connection (such as engine, physical location, and authentication). The string might not include libref attributes that define behavior. For example, if a previously submitted LIBNAME statement for the SAS V9 engine specifies that SAS data sets are to be compressed, the compression attribute is not used by the DS2 procedure. In order to request compression, you must specify the COMPRESS= table option in your DS2 program. Some behavior attributes are used by the procedure.

You can determine which LIBNAME options are used by the procedure by setting the MSGLEVEL=i system option before submitting a LIBNAME statement. Not all LIBNAME statement options have a corresponding table option.

When a table is referenced in a DS2 program without a libref, or when an engine name is omitted from a LIBNAME statement, the default SAS V9 engine is used. Tables that are referenced without a libref are created in the SAS Work library, or in the User library, if a User library is defined.

Connecting to the CAS Server

To submit a DS2 program to the CAS server:

- 1 Submit the CAS statement to establish a CAS session in the SAS session.
- 2 Assign a caslib or use a pre-defined caslib for the data source that you want to interact with in the session. You can assign a caslib and list the caslibs that are available to your CAS session by using the CASLIB statement. A caslib uses a SAS Data Connector (or SAS Data Connector Accelerator) to access data. For information about SAS Data Connectors, see [SAS Cloud Analytic Services: User's Guide](#).

- 3 Specify the SESSREF= (or SESSUUID=) procedure option with the CAS session name or session identifier in the PROC DS2 statement. When SESSREF= (or SESSUUID=) is specified, both the default connection mechanism and the LIBS= procedure option are ignored. Instead, the procedure connects to the specified CAS session.
- 4 You can now reference tables in the caslib. Tables that are in the active caslib can be referenced simply by name. A two-level name in the form *caslib.table-name* is required to access tables that are not in the active caslib.

The following example illustrates a PROC DS2 request to the CAS server:

```
cas; 1

caslib castera desc='Teradata Caslib' 2
  datasource=(srctype='teradata'
    username='myname'
    password='mypw'
    server='testserver',
    db='testdb');

proc ds2 sessref=casauto; 3
  data newtable;
  method run();
  set {select * from employees}; 4
end;
enddata;
run;
quit;
```

- 1 This example establishes a CAS session. This CAS session has the default name Casauto.
- 2 The CASLIB statement assigns caslib Castera. Castera becomes the active caslib.
- 3 The PROC DS2 statement specifies the SESSREF= procedure option and the CAS session name Casauto.
- 4 The FedSQL SELECT statement in the DS2 SET statement identifies Teradata table Employees by name. There is no need to qualify the names of the tables that exist in the active caslib with a caslib. In the active caslib, use of the caslib in the table name is optional.

For more information about the CAS statement, CASLIB statement, and CASUTIL procedure in [SAS Cloud Analytic Services: User's Guide](#).

The previous example uses a data connector to automatically load data into CAS. When you use a caslib to reference a table, the table is automatically loaded into the specified CAS session if it was not previously loaded. Data can also be loaded into CAS from path-based locations. For an example that explicitly loads data into the CAS session, see [“Example 5: Run a DS2 Program in CAS” on page 628](#).

The CAS tables that you create and work with using PROC DS2 are in-memory session tables. That is, the tables are available for the duration of the CAS session and are accessible only to the current session. PROC DS2 does not provide a way to persist a table to a data source or to share the table with other CAS sessions. To persist or share a CAS table, use the CASUTIL procedure.

Although CAS tables are in-memory tables, you must specify the `OVERWRITE=` table option to overwrite an initial output table with a replacement output table.

For more information about DS2 use in CAS, see [“DS2 Functionality in SAS Cloud Analytic Services” on page 613](#).

DS2 Functionality in SAS Cloud Analytic Services

SAS Cloud Analytic Services is an alternative environment for processing DS2 requests. When you specify the `SESSREF=` (or `SESSUID=`) option, PROC DS2 prepares and executes your DS2 programs on a CAS server. When you run PROC DS2 with `SESSREF=` or `SESSUID=`, you are actually using the `runDS2` action. For more information, see the `runDS2` action in the *SAS Viya Platform: System Programming Guide*.

The CAS server is a symmetric multiprocessing (SMP) server. A DS2 program executing on the CAS server can perform manipulations on multiple data observations concurrently, thereby reducing the time required to process large data sets. Based on the structure of the DS2 program, the DS2 compiler determines which operations can be performed on multiple observations concurrently. It also determines which operations must be applied to each observation sequentially.

A DS2 program is classified as either a serial program, a parallel program, or a parallel-serial program. In order to benefit from the CAS server, the DS2 program must be structured as either a DS2 parallel program or a DS2 parallel-serial program.

- A DS2 parallel program contains no operations with data dependencies across observations. Therefore, multiple data observations can be processed in parallel. Each CAS worker can process a subset of the input data and generate a subset of the result set.
- A DS2 parallel-serial program contains some operations with data dependencies across observations and some operations without data dependencies. The processing of the operations is divided into two stages: a parallel stage and a serial stage. During the parallel stage, each CAS worker processes a subset of the input data set and generates a subset of an intermediate data set. During the serial stage, one CAS worker processes the complete intermediate data set and generates the complete result set.

Most of the functionality of the DS2 language is supported for use on the CAS server. However, there are some exceptions. For information about the DS2 functionality that is supported in CAS, see [SAS DS2 Programmer's Guide](#).

For an example of how a DS2 parallel program is submitted to the CAS server, see [“Example 5: Run a DS2 Program in CAS” on page 628](#).

DS2 output tables in CAS are in-memory tables. The tables are created in the user's CAS session. You must use other CAS actions to promote the output tables for global use in CAS or to store data to caslib data sources.

RUN-Group Processing

PROC DS2 supports RUN-group processing. RUN-group processing enables you to submit RUN groups without ending the procedure.

To use RUN-group processing, you start the procedure and then submit multiple RUN-groups. A RUN-group is a group of statements that contains at least one action statement and ends with a RUN statement. As long as you do not terminate the procedure, it remains active and you do not need to resubmit the PROC statement.

Note: When using PROC DS2, DS2 programs are delimited by RUN statements. If additional DS2 code is found after a RUN statement, then this code composes a new, distinct DS2 program from the DS2 program before the previous RUN statement.

To end RUN-group processing, submit a RUN CANCEL statement. Statements that have not been submitted are terminated. To stop the procedure, submit a QUIT statement. Statements that have not been submitted are terminated as well.

Applying DS2 Table Options

When you access a data source with PROC DS2, you can apply DS2 table options in the subsequent DS2 statements. A table option specifies actions that enable you to perform operations on a table such as assigning buffer page size or specifying passwords.

DS2 table options are used to apply options for DS2 language processing, like SAS data set options apply options for DATA step processing. SAS data set options cannot be used with the DS2 language. For example, the following code applies a table option to a SAS data set to specify the size of a permanent buffer page for the new table:

```
libname myfiles v9 'library-path';

proc ds2;
  data myfiles.table1 (bufsize=16k);
    dcl double j j2;
    method run();
      do j = 1 to 1000;
        j2 = 2*j;
        output;
      end;
    end;
enddata;
run;
```

```
quit;
```

Table options are data source specific. For a list of available table options, see “[DS2 Statement Table Options by Data Source](#)” in *SAS DS2 Language Reference*..

Using Macro Variables in a Literal String

Note: This information does not apply to DS2 use on the CAS server.

Macro variables enable you to dynamically modify text in a program through symbolic substitution. When you reference a macro variable in a program, the macro processor replaces the reference with the value of the specified macro variable.

With PROC DS2, you can use a macro variable on a subsequent DS2 statement. However, if a macro variable occurs within a literal string, you cannot enclose the string in double quotation marks. The macro processor requires double quotation marks to resolve the macro variable reference. DS2 statements consider a string enclosed in double quotation marks to be a delimited (case sensitive) identifier such as a table or column name.

To reference a macro variable in a literal string, use the SAS macro function %TSLIT. %TSLIT overrides the need for double quotation marks around the literal string and puts single quotation marks around the input value. For example, the following statement includes the %TSLIT function to specify the &SYSHOSTNAME macro variable, which returns the host name of the computer on which it is executed:

```
if hostname = %tslit(&syshostname) then ...
```

The %TSLIT macro function is stored in the default autocall macro library. For more information, see “Referencing a Macro Variable in a Delimited Identifier” in the *[SAS DS2 Programmer's Guide](#)*.

DS2 Automatic Variables

The DS2 language sets up automatic variables with certain values after it executes each statement. These automatic variables are useful when diagnosing the cause of a problem across DS2 threads. They are also useful for providing context when debugging with PUT statements.

Table 18.1 DS2 Automatic Variables

Variable	Description
HOSTNAME	Returns the name of the worker node or host on which the DS2 program is running.

Variable	Description
<code>_NTHREADS_</code>	Returns the total number of DS2 threads running in the program. In a parallel environment, <code>_NTHREADS_</code> is the total number of DS2 threads across all nodes on which the DS2 program is running.
<code>_THREADID_</code>	Returns the <code>_THREADID_</code> . A serial program (which does not contain a thread component) is assigned <code>_THREADID_ = 0</code> . In a parallel program, the executing data program is assigned <code>_THREADID_ = 0</code> and each executing thread program is assigned a unique <code>_THREADID_</code> from 1 to the number of threads.

Here is an example of how the automatic variables might be used:

```
proc ds2;
  thread thd / overwrite = yes;
    dcl double x;
    method init();
      put 'THREAD: thread' _threadid_ 'on' _hostname_;
    end;
  endthread;

  data _null_;
    dcl thread thd t;
    method init();
      put 'DATA: thread' _threadid_ 'on' _hostname_;
    end;
    method run();
      set from t threads=8;
    end;
  enddata;
run;
quit;
```

For more information, see [“Automatic Variables That Are Useful in DS2 Threading” in SAS DS2 Programmer’s Guide](#).

Passwords

This information does not apply to DS2 use on the CAS server.

SAS software enables you to restrict access to SAS data sets and SPD Engine data sets by assigning SAS passwords to the files. You can specify three levels of protection: read, write, and alter.

With PROC DS2, you assign or specify a password for a data source using the DS2 table options `ALTER=`, `PW=`, `READ=`, and `WRITE=`. For example, the following code applies the DS2 table option `PW=` in order to assign READ, WRITE, and ALTER passwords to a SAS data set:

```
libname myfiles v9 'library-path';
```

```

proc ds2;
  data myfiles.table1 (pw=luke);
    dcl double j j2;
    method run();
      do j = 1 to 1000;
        j2 = 2*j;
        output;
      end;
    end;
enddata;
run;
quit;

```

A SAS password does not control access to a SAS file beyond SAS. You should use the operating system-supplied utilities and file-system security controls to control access to SAS files outside SAS. For more information about SAS passwords, see [“Protecting Files” in SAS Programmer’s Guide: Essentials](#).

CAS tables do not support SAS passwords. You cannot assign a password for a CAS table. When accessing password-protected data from CAS, passwords are specified in the CAS language element that is used to access the data. For example, you can specify passwords in the CASUTIL procedure, which loads data into CAS, or in the CASLIB statement. For more information, see the SAS Data Connector documentation in *SAS Cloud Analytic Services: User’s Guide*.

Encryption

SAS enables you to encrypt the contents of a SAS data set, SPD Engine data set, and SPD Server table. SAS supports SAS proprietary encryption and AES encryption.

SAS proprietary encryption is performed by specifying the ENCRYPT= table option with the PW= or READ= table option. A data set or table encrypted with SAS proprietary encryption must be decrypted by specifying the PW= or READ= table option with the appropriate password.

AES encryption is performed by specifying the ENCRYPT= table option with the ENCRYPTKEY= table option. A data set or table encrypted with AES encryption is later decrypted by specifying the ENCRYPTKEY= table option with the appropriate key value. SAS supports two levels of AES encryption: AES and AES2. AES2 meets newer and more secure encryption standards than AES. For more information, see [“ENCRYPT= Table Option” in SAS DS2 Language Reference](#).

DS2 currently does not support the encryption attribute for CAS tables. When accessing SAS encrypted and AES-encrypted data sets from CAS, passwords and encryption keys are specified in the CAS language element that is used to access the data. For example, you can specify passwords and encryption keys in the CASUTIL procedure, which loads data into CAS, or in the CASLIB statement. For more information, see the SAS Data Connector documentation in [SAS Cloud Analytic Services: User’s Guide](#).

DS2 Data Type Support for SAS Data Sets

In PROC DS2, when you submit DS2 statements, all DS2 language data types are supported. However, when reading and creating data, the DS2 data types are translated to and from the data types of the target data source. The following table lists the DS2 data types and how they are translated to and from the SAS data types:

Table 18.2 DS2 Data Type Translation for SAS Data Sets

DS2 Data Type	Legacy SAS Data Type	Description
BIGINT	SAS numeric	Because a SAS numeric is a DOUBLE, which is an approximate numeric data type rather than an exact numeric data type, there is potential for loss of precision.
BINARY(<i>n</i>)	SAS character	Applies the SAS format \$n.
CHAR(<i>n</i>)	SAS character	Applies the SAS format \$n.
DATE	SAS numeric ¹	Applies the SAS format DATE9.
DECIMAL NUMERIC(<i>p,s</i>)	SAS numeric	
DOUBLE	SAS numeric	
FLOAT	SAS numeric	
INTEGER	SAS numeric	
NCHAR(<i>n</i>)	SAS character	Applies the SAS format \$n.
NVARCHAR(<i>n</i>)	SAS character	Applies the SAS format \$n.
REAL	SAS numeric	
SMALLINT	SAS numeric	
TIME(<i>p</i>)	SAS numeric	Applies the SAS format TIME8.
TIMESTAMP(<i>p</i>)	SAS numeric	Applies the SAS format DATETIME19.2.
TINYINT	SAS numeric	

DS2 Data Type	Legacy SAS Data Type	Description
VARBINARY(<i>n</i>)	SAS character	Applies the SAS format \$n.
VARCHAR(<i>n</i>)	SAS character	Applies the SAS format \$n.

¹ Valid SAS date values are in the range from 1582-01-01 to 9999-12-31. Dates outside the SAS date range are not supported and are treated as invalid dates.

For more information about the DS2 data types, see [SAS DS2 Language Reference](#). For information about data type support for DBMS data sources, see “Data Type Reference” in [SAS DS2 Language Reference](#).

DS2 Data Type Support for CAS Tables

When DS2 programs are executed in CAS, DS2 data types are translated to CAS data types. The CAS server supports more data types than the SAS data engines. DS2 can create, read, and write columns of type BIGINT, CHAR(*n*), DOUBLE, INTEGER, VARBINARY, and VARCHAR in CAS. Some DS2 data types are not supported in CAS. The following table lists the DS2 data types and how they are translated to CAS data types.

Table 18.3 DS2 Data Type Translation for CAS Tables

DS2 Data Type	CAS Data Type	Description
BIGINT	INT64	Large signed, exact whole number.
BINARY(<i>n</i>)	Not supported ¹	
CHAR(<i>n</i>)	CHAR	Applies the SAS format \$n.
DATE	DOUBLE ²	Applies the SAS format DATE9.
DECIMAL NUMERIC(<i>p,s</i>)	Not supported	
DOUBLE	DOUBLE	Stores a signed, approximate, double-precision, floating-point number. For SAS Cloud Analytic Services, this is a 64-bit double precision, floating-point number.
FLOAT	DOUBLE	
INTEGER	INT32	Regular signed, exact whole number.

DS2 Data Type	CAS Data Type	Description
NCHAR(<i>n</i>)	CHAR	Applies the SAS format \$ <i>n</i> .
NVARCHAR(<i>n</i>)	VARCHAR	
REAL	DOUBLE	
SMALLINT	INT32	Small signed, exact whole number.
TIME(<i>p</i>)	DOUBLE	Applies the SAS format TIME8.
TIMESTAMP(<i>p</i>)	DOUBLE	Applies the SAS format DATETIME25.6.
TINYINT	INT32	Very small signed, exact whole number.
VARBINARY(<i>n</i>)	VARBINARY	Varying-length binary opaque data. This data type is used to store image data, audio, documents, and other unstructured data.
VARCHAR(<i>n</i>)	VARCHAR	Stores a varying-length character string.

- 1 A DS2 program that runs in CAS can create a column with a BINARY data type. However, it cannot read a BINARY data column from a CAS table or write to an existing BINARY column in a CAS table. DS2 returns an error if an attempt is made to do so. Use the DROP data set option to exclude existing BINARY columns when reading or writing to a CAS table that contains a BINARY column.
- 2 Valid SAS date values are in the range from 1582-01-01 to 9999-12-31. Dates outside of the SAS date range are not supported and are treated as invalid dates.

CAS tables use the UTF-8 character set by default.

When you specify to create DATE, TIME, and TIMESTAMP columns in a CAS table, DS2 creates a DOUBLE column instead and applies the SAS date, time, and datetime formats noted in the table to the columns, as appropriate. When reading numeric columns that contain date, time, and timestamp values from SAS data sets and SPD Engine data sets into CAS, DS2 preserves any existing formats that are defined.

Examples: DS2 Procedure

Example 1: Introducing DS2 Code

Features:

- PROC DS2 statement
- LIBS= procedure option

QUIT statement
DS2 language statements

Details

This example uses a simple DS2 program that displays **Hello World!** in the SAS log. The example shows basic differences between DS2 and the SAS DATA step. The code looks similar to the SAS DATA step, but there are syntax elements that are different, such as default system methods (INIT, RUN, and TERM). Also, DS2 supports the most common SQL data types such as DECIMAL, INTEGER, and VARCHAR to make operations more native for DBMS data.

Program

```
proc ds2 libs=work;
data _null_;
  method init();
    dcl varchar(16) str;
    str = 'Hello World!';
    put str;
  end;
enddata;

run;

quit;
```

Program Description

Execute the PROC DS2 statement. The LIBS= connection option specifies to execute the request in the SAS Work library. The PROC DS2 statement sets up the environment to submit DS2 language statements.

```
proc ds2 libs=work;
```

Enter the DS2 language statements. _NULL_ on the DS2 DATA statement indicates that there is no automatic output generated. The DS2 PUT statement writes to the SAS log.

```
data _null_;
  method init();
    dcl varchar(16) str;
    str = 'Hello World!';
    put str;
  end;
enddata;
```

Submit the DS2 statements. The RUN statement submits the DS2 statements. The RUN statement is required. SAS reads the program statements that are associated with one task until it reaches a RUN statement.

```
run;
```

Stop the procedure. The QUIT statement stops the procedure.

```
quit;
```

Example Code 18.1 SAS Log Showing Hello World!

```
48  proc ds2 libs=work;
49  data _null_;
50      method init();
51          dcl varchar(16) str;
52          str = 'Hello World!';
53          put str;
54      end;
55  enddata;
56  run;
Hello World!
NOTE: Execution succeeded. No rows affected.
57  quit;

NOTE: PROCEDURE DS2 used (Total process time):
      real time           16.38 seconds
      cpu time            0.15 seconds
```

Example 2: Creating a SAS Data Set

Features:

- PROC DS2 statement
- LIBS= procedure option
- QUIT statement
- LIBNAME statement
- DS2 language statements
- PROC PRINT

Details

This example creates a SAS data set in a SAS session by submitting the DS2 procedure, and then submitting DS2 language statements. The output shows the first ten rows of the data set.

Program

```
libname myfiles base 'library-path';

proc ds2 libs=myfiles;

    data myfiles.basetable;
        declare double j j2;
        method run();
            do j = 1 to 1000;
                j2 = 2*j;
                output;
            end;
        end;
    enddata;

run;

quit;

proc print data=myfiles.basetable (obs=10);
run;
```

Program Description

Assign a library reference to the SAS data set to be created. The LIBNAME statement assigns the libref Myfiles, specifies the SAS V9 engine, and specifies the physical location for the SAS data set.

```
libname myfiles base 'library-path';
```

Execute the PROC DS2 statement. The PROC DS2 statement connects to the data source by using the libref Myfiles and sets up the environment to submit DS2 language statements.

```
proc ds2 libs=myfiles;
```

Enter the DS2 language statements. The DS2 DATA statement creates an output table named Myfiles.BaseTable. The two-level name in the DATA statement specifies the catalog identifier Myfiles. The DECLARE statement assigns the data type DOUBLE to the variables J and J2. The METHOD statement identifies the RUN system method that is used to create output. The OUTPUT statement writes a row to table Myfiles.BaseTable after each execution of the DO loop.

```
data myfiles.basetable;
    declare double j j2;
    method run();
        do j = 1 to 1000;
            j2 = 2*j;
            output;
        end;
    end;
enddata;
```

Submit the DS2 language statements. The RUN statement submits the DS2 statements. The RUN statement is required. SAS reads the program statements that are associated with one task until it reaches a RUN statement.

```
run;
```

Stop the procedure. The QUIT statement stops the procedure.

```
quit;
```

Print the SAS data set. The PRINT procedure prints the observations in the SAS data set. The OBS= data set option limits the output to 10 observations.

```
proc print data=myfiles.basetable (obs=10);  
run;
```

Output: Creating a SAS Data Set

Output 18.1 PROC PRINT Output of Myfiles.V9Table

The SAS System

Obs	j	j2
1	1	2
2	2	4
3	3	6
4	4	8
5	5	10
6	6	12
7	7	14
8	8	16
9	9	18
10	10	20

Example 3: Terminating the Current Step in Line Prompt Mode

Features:

- PROC DS2 statement
- RUN CANCEL statement
- QUIT statement
- DS2 language statements

Details

The following example shows the usefulness of the RUN CANCEL statement in a line prompt mode session. The sixth statement in the code contains an invalid value for the column (Z instead of Y). RUN CANCEL ends the PROC DS2 step and prevents it from executing.

Program

```
proc ds2;
data xy_data;
  declare double x y;
  method init();
    do x = 1 to 5;
      z = 2*x;
    end;
  end;
enddata;
run cancel;
quit;
```

Example Code 18.2 SAS Log Showing the Canceled PROC DS2 Step

```
17  proc ds2 libs=work;
NOTE: Writing HTML Body file: sashtml1.htm
18  data xy_data;
19    dcl double x y;
20    method init();
21      do x = 1 to 5;
22        z = 2*x;
23      end;
24    end;
25  enddata;
26  run cancel;
NOTE: DS2 query cancelled per user request.
27  quit;

NOTE: PROCEDURE DS2 used (Total process time):
      real time          14.33 seconds
      cpu time           0.71 seconds
```

Example 4: Routing Data to Tables Based on Values

Features: PROC DS2 statement
 QUIT statement
 DS2 language statements

Details

This example illustrates how to create tables based on a condition. Programs 1 and 2 create two tables, Dept1_Items and Dept2_Items, that hold costs for items used by two departments. The third program creates two tables, Highcosts and Lowcosts, based on the costs of the items in the two items tables. Programs 4 and 5 output the contents of the costs tables.

Program

```
proc ds2;
/* Program 1 */
data dept1_items (overwrite=yes);
  dcl varchar(20) item;
  dcl double cost;
  method init();
    item = 'staples';   cost = 1.59; output;
    item = 'pens';     cost = 3.26; output;
    item = 'envelopes'; cost = 11.42; output;
  end;
enddata;
run;
/* Program 2 */
data dept2_items (overwrite=yes);
  dcl varchar(20) item;
  dcl double cost;
  method init();
    item = 'erasers'; cost = 5.43; output;
    item = 'paper';   cost = 26.92; output;
    item = 'toner';   cost = 62.29; output;
  end;
enddata;
run;
/* Program 3 */
data lowCosts (overwrite=yes) highCosts (overwrite=yes);
  method run();
```



```

set dept1_items dept2_items;
if cost <= 10.00 then
  output lowCosts;
else
  output highCosts;
end;
enddata;
run;
/* Program 4 */
data;
  method run();
    set lowCosts;
  end;
enddata;
run;
/* Program 5 */
data;
  method run();
    set highCosts;
  end;
enddata;
run;
quit;

```

Output: Creating Tables Based on a Condition

Output 18.2 Output of the Costs Tables

The SAS System

item	cost
staples	1.59
pens	3.26
erasers	5.43

The SAS System

item	cost
envelopes	11.42
paper	26.92
toner	62.29

Example 5: Run a DS2 Program in CAS

Features:	CAS system options
	CAS statement
	CASLIB statement
	CASUTIL procedure
	PROC DS2 statement
	SESSREF= procedure option
	DS2 language statements

Details

Here is an example of a DS2 parallel program that is run in CAS. A DS2 parallel program is a DS2 program that contains a thread program and a data program and does not contain any data manipulation statements. That is, the data program does not contain any statements besides SET FROM and OUTPUT. Operations in the thread program are applied to multiple data observations in parallel. Each CAS worker processes a subset of the data set and generates a subset of the result set.

Program

```
cas;

caslib casdata datasource=(srctype=path)
  path="testdata/cas";

proc casutil;
  load casdata="cars_single.sashdat" incaslib="casdata"
  casout="cars_single";
run;

proc ds2 sessref=casauto;

  thread cars_thd / overwrite=yes;
    method run();
      set cars_single;
      if (msrp > 100000) then do;
        put make= model= msrp=;
        output;
      end;
    end;
  endthread;

data cars_luxury /overwrite=yes;
  dcl thread cars_thd t;
```

```

        method run();
            set from t threads=4;
        end;
    enddata;

run;

quit;

```

Program Description

Establish a CAS session. You establish a CAS session by submitting the CAS statement. This CAS session is created with the default name Casauto.

```
cas;
```

Define a caslib to access your input data. We want to access a file named Cars_Single.sashdat, which is located in the Testdata/Cas subdirectory of the computer on which the CAS server is running. The CASLIB statement assigns caslib Casdata to the directory location. To access a directory on a different computer, specify an absolute pathname.

```
caslib casdata datasource=(src=type=path)
    path="testdata/cas";
```

Load the table into CAS for processing. The CASUTIL procedure is used to load table Cars_Single.sashdat into the CAS session. The CASOUT= parameter assigns the loaded table the name Cars_Single.

```
proc casutil;
    load casdata="cars_single.sashdat" incaslib="casdata"
    casout="cars_single";
run;
```

Issue the PROC DS2 statement and specify the SESSREF= procedure option. In the SESSREF= procedure option, specify the CAS session name. The SESSREF= option specifies Casauto. The SESSREF= option instructs the procedure to pass the DS2 program that follows to the DS2.runDS2 action and to process them in CAS session Casauto.

```
proc ds2 sessref=casauto;
```

Enter DS2 statements. The DS2 THREAD statement creates a thread program named Cars_Thd that specifies criteria for selecting data from loaded table Cars_Single. The DS2 DATA statement creates an output table, Cars_Luxury, and specifies to set the results of the thread program as the content of the new table. The data program specifies to use four threads to execute the thread program.

```

thread cars_thd / overwrite=yes;
method run();
    set cars_single;
    if (msrp > 100000) then do;
        put make= model= msrp=;
        output;
    end;
end;
end;
```

```
endthread;  
  
data cars_luxury /overwrite=yes;  
  dcl thread cars_thd t;  
  method run();  
    set from t threads=4;  
  end;  
enddata;
```

Submit the DS2 language statements. The RUN statement submits the DS2 statements. The RUN statement is required. SAS reads the program statements that are associated with one task until it reaches a RUN statement.

```
run;
```

Stop the procedure. The QUIT statement stops the procedure.

```
quit;
```

DSTODS2 Procedure

Overview: DSTODS2 Procedure	631
What Does the DSTODS2 Procedure Do?	631
Concepts: DSTODS2 Procedure	632
Input and Output Files	632
Supported and Unsupported Syntax	632
Considerations and Limitations with PROC DSTODS2	635
Syntax: DSTODS2 Procedure	636
PROC DSTODS2 Statement	636
Examples: DSTODS2 Procedure	637
Example 1: Data Input	637
Example 2: PUT Statement with Line Specifiers	638
Example 3: Arrays	640

Overview: DSTODS2 Procedure

What Does the DSTODS2 Procedure Do?

The DSTODS2 procedure enables you to translate a subset of your SAS DATA step code into DS2 code. Then you can revise your program to take advantage of DS2 features and submit your program using PROC DS2.

For more information, see [Chapter 18, “DS2 Procedure,”](#) on page 597.

Concepts: DSTODS2 Procedure

Input and Output Files

PROC DSTODS2 works with text files for input and output, and these files are presented either as directly named files or as SAS filerefs.

PROC DSTODS2 requires an input file containing the source to be translated. The input file is specified via the IN= argument. PROC DSTODS2 also requires an output filename where the translated source is to be written. This file is specified via the OUT= argument.

Note: The resulting output file might not be syntactically complete. You might have to clean up the output file to create a DS2 program that can be compiled and executed.

Note: PROC DSTODS2 adds this code line to your output file. You should not remove it from your .ds2 file during cleanup:

```
_return: ;
```

Supported and Unsupported Syntax

Overview

PROC DSTODS2 cannot translate all possible DATA step syntax. Its main purpose is to support typical SAS Enterprise Miner scoring syntax, which is a subset of the full DATA step syntax.

Supported Syntax

The DATA step syntax that PROC DSTODS2 supports includes the following items.

- The following statements are supported:

ARRAY statements and array initializers. Implicit arrays give a warning.

Assignment statements that use arrays and scalars.

Attribute statements, although they have no semantic meaning in PROC DSTODS2. Only the syntax is supported—that is, you do not get a syntax error for them.

BY

Comment

CONTINUE

DATA statement and associated data sets

DO (all forms except DO OVER)

DROP

END

FORMAT (only when a LENGTH statement is also included in the program)

GOTO

IF and IF-THEN/ELSE

KEEP

LABEL

LEAVE

LENGTH

MERGE

Null

OUTPUT

Basic PUT statement functionality: printing variables, arrays, `_ALL_`, and formatted values. Other PUT arguments are commented out inline.

RENAME

RETAIN

RETURN

RUN (syntactically supported but has no semantic meaning)

SELECT

SET

STOP

Sum

- Partial support for the hash object.
- DROP, IN, KEEP, and RENAME data set options
- All DATA step expressions
- All functions and formats are translated as-is. However, only functions and formats that are supported by DS2 are valid. For more information, see [“DS2 Functions” in SAS DS2 Language Reference](#).
- Constant lists (as used in IN clauses)
- Variable lists except for type modifiers in lists, for example, `x-numeric-a`

Note: Because of the depth of the DATA step language, this list is not exhaustive.

Unsupported Syntax

The DATA step syntax that PROC DSTODS2 does not support includes the following items:

- The following statements are not supported:

ABORT	INFILE
ATTRIB	INFORMAT
CALL	INPUT (all forms)
CARDS	LINK
CARDS4	LIST
DATALINES	LOSTCARD
DATALINES4	MODIFY
DELETE	PUT statement with formatting of any kind
DESCRIBE	PUTLOG
DISPLAY	REMOVE
DO OVER	REPLACE
ERROR	UPDATE
EXECUTE	WHERE
FILE	WINDOW
FORMAT (undeclared variables)	

- Data set options other than DROP, IN, KEEP, and RENAME
- DATA step executable options like DEBUG and PASSTHRU
- The following I/O options are not supported:

CONTROL	NOBS
CUROBS	OPEN
END	POINT
INDSNAME	KEYRESET
KEY	

- Implicit arrays
- INPUT and PUT modifiers
- All objects other than the Hash object
- Argument tags in any method
- View and stored program syntax
- Constant ranges, such as 1:100, in constant lists
- Format justifiers (L, R, C)

- CALL statements except DMNORM and STREAMINIT
- Bitstring expressions
- Other procedures

Note: Because of the depth of the DATA step language, this list is not exhaustive.

Considerations and Limitations with PROC DSTODS2

- Only one DATA step program can be converted at a time.
- The resulting DS2 program contains two lines of code that should be removed before saving your final DS2 program:


```
ds2_options sas tkgmac;
_return: ;
```
- Existing comments are removed.
- Code lines that cannot be translated are placed in comments.

These comments are placed in-line in the original position as much as possible to enable you to easily compare the original code with the translated code. But there are situations where comments and other code might be moved to other positions. For example, the code might be moved to the top of a block (the outermost scope).
- The resulting DS2 program might not be syntactically complete. The resulting program can result in one of the following outcomes:
 - ☐ compile and execute without error. This is unlikely if any of the new DATA step syntax is used.
 - ☐ compile but fail to execute. This is particularly true of any programs that contain commented-out sections.
 - ☐ not compile. This could happen because there was a warning, or something that translates syntactically but has no corresponding support in DS2. This could be the case for certain functions and formats. It could happen for some of the variable list features also.
 - ☐ produce a fatal syntax error. This could happen for some previously undiscovered feature or for some aspect of the previously mentioned variable list syntax.
- The resulting DS2 program does not invoke the SAS In-Database Code Accelerator. You must take the part of the code that can be run in parallel and create a thread program to accompany the data program.
- If any code line is longer than 32767 characters, it is truncated.
- DATA step fixed-length characters are measured in bytes. DS2 fixed-length characters are measured in characters. If your data has multibyte encoding, you

might have to adjust your LENGTH statement before running PROC DSTODS2 to account for the length of the variable in characters to avoid truncation. There is no issue with single-byte encodings.

- If you run PROC DSTODS2 on a DATA step program with the SESSREF= option to run on the CAS server, you must move the SESSREF= option from the DATA statement to the PROC DS2 statement before running on the CAS server.

Syntax: DSTODS2 Procedure

Restriction: This procedure is not supported on the CAS server.

```
PROC DSTODS2 IN=datastep-program-filename OUT=ds2-program-filename
<OUTDIR="output-directory-name">
RUN;
<QUIT;>
```

Statement	Task	Example
PROC DSTODS2	Translates DATA step code into DS2 code.	Ex. 1, Ex. 2, Ex. 3

PROC DSTODS2 Statement

Translates DATA step code into DS2 code.

Syntax

```
PROC DSTODS2 IN="datastep-program-filename" OUT="ds2-program-filename"
<OUTDIR="output-directory-name">
```

Required Arguments

IN=datastep-program-filename

specifies the name of the DATA step file or a SAS fileref.

Interaction If you use a fileref for the *datastep-program-filename*, PROC DSTODS2 does not look at any fileref options (for example, encoding).

Tip You can specify a full pathname.

OUT=ds2-program-filename

specifies the name of the DS2 file that is created.

Restriction You must specify only a single, output filename. It cannot include a pathname.

Note You should remove the following two lines from your output .ds2 program:

```
ds2_options sas tkgmac;
_return: ;
```

Tip If you cannot or do not specify the OUTDIR= directory, the output file is automatically written to the current working directory. You can use this code to find your current working directory:

```
data _null_;
file 'name.txt';
put x;
run;
```

Optional Argument

OUTDIR="output-directory-name"

specifies the output directory name for the file.

Examples: DSTODS2 Procedure

Example 1: Data Input

Features: PROC DSTODS2 statement

Details

This example uses PROC DSTODS2 to translate the following DATA step program, dsEx1.sas, that has a SET statement and a BY statement.

```
data _null_;
  length x y z w 8;
  set x;
  by x--z w;
  put x=;
run;
```

Execute the PROC DSTODS2 statement. Without a currently assigned libref, the PROC DSTODS2 statement simply sets up the environment to submit DS2 language statements.

```
proc dstods2 in="dsEx1.sas" out="ds2Ex1.ds2";
```

Submit the DSTODS2 statements. The RUN statement submits the DSTODS2 statements. The RUN statement is required. SAS reads the program statements that are associated with one task until it reaches a RUN statement.

```
run;
```

Example Code 19.1 SAS Log Showing Results

```
4 proc dstods2 in="dsEx1.sas" out="ds2Ex1.ds2";
5 run;

NOTE: PROCEDURE DSTODS2 used (Total process time):
      real time          0.20 seconds
      cpu time           0.07 seconds
```

Output: ds2Ex1.ds2 Program

PROC DSTODS2 translates dsEx1.sas into DS2 code and stores the result in dsEx1.ds2. As you can see, the LENGTH statement has been translated to DECLARE statements, the METHOD statement has been added, and the variable list, **x--z**, has been commented out.

```
data _NULL_;
  decl double X;
  decl double Y;
  decl double Z;
  decl double W;
  method run();
  set X;
  by /* X--Z */ W;
  put X=;
  ;
  _return: ;
end;
enddata;
```

Example 2: PUT Statement with Line Specifiers

Features: PROC DSTODS2 statement

Details

This example uses PROC DSTODS2 to translate the following DATA step program, dsEx2.sas, that has a DO statement and several PUT statements.

```
data _null_;
  file temp linesize=32600;
  do i = 1 to 5330;
    put i 6. @;
  end;
  put @31990 'N1=72' @;
  put @32580 'N2=32413' @;
  put @32001 'N4=32 N3=783424123' @;
  put @32477 'N5=1977' @;
  put @32222 'N6=1981' ;
run;
```

Execute the PROC DSTODS2 statement. Without a currently assigned libref, the PROC DSTODS2 statement simply sets up the environment to submit DS2 language statements.

```
proc dstods2 in="dsEx2.sas" out="ds2Ex2.ds2";
```

Submit the DSTODS2 statements. The RUN statement submits the DSTODS2 statements. The RUN statement is required. SAS reads the program statements that are associated with one task until it reaches a RUN statement.

```
run;
```

Example Code 19.2 SAS Log Showing Results

```
4  proc dstods2 in="dsEx2.sas" out="ds2Ex2.ds2";
5  run;

NOTE: PROCEDURE DSTODS2 used (Total process time):
      real time           0.18 seconds
      cpu time            0.06 seconds
```

Output: ds2Ex2.ds2 Program

PROC DSTODS2 translates dsEx2.sas into DS2 code and stores the result in dsEx2.ds2. As you can see, the METHOD statement has been added and the FILE statement has been commented out. The PUT statement is translated but the @ line-hold specifier and constants have been commented out.

```

data _NULL_;
method run();
/* FILE TEMP LINESIZE = 32600 */;
do I = 1.0 to 5330.0;
put I 6. /* Put statement contains unsupported feature(s) @ */ ;
end;
put /* Put statement contains unsupported feature(s) @ 31990 'N1=72'
@ */ ;
put /* Put statement contains unsupported feature(s) @ 32580 'N2=32413'
@ */ ;
put /* Put statement contains unsupported feature(s) @ 32001 'N4=32
N3=783424123'
@ */ ;
put /* Put statement contains unsupported feature(s) @ 32477 'N5=1977'
@ */ ;
put /* Put statement contains unsupported feature(s) @ 32222 'N6=1981'
*/ ;
;
_return: ;
end;
enddata;

```

Example 3: Arrays

Features: PROC DSTODS2 statement

Details

This example uses PROC DSTODS2 to translate the following DATA step program, dsEx2.sas, that has an ARRAY statement.

```

data _null_;
  array a(*) a1-a5;
  retain a1-a5 (1*(10 10 10 10 10));
  put _all_;
run;

```

Execute the PROC DSTODS2 statement. Without a currently assigned libref, the PROC DSTODS2 statement simply sets up the environment to submit DS2 language statements.

```
proc dstods2 in="dsEx3.sas" out="ds2Ex3.ds2";
```

Submit the DSTODS2 statements. The RUN statement submits the DSTODS2 statements. The RUN statement is required. SAS reads the program statements that are associated with one task until it reaches a RUN statement.

```
run;
```

Example Code 19.3 SAS Log Showing Results

```
6   proc dstods2 in="dsex3.sas" out="ds2ex3.ds2";
7   run;
```

NOTE: PROCEDURE DSTODS2 used (Total process time):

real time	1.83 seconds
cpu time	1.14 seconds

Output: ds2Ex3.ds2 Program

PROC DSTODS2 translates dsEx3.sas into DS2 code and stores the result in dsEx3.ds2. As you can see, the METHOD statement has been added and the ARRAY statement has been translated into a VARARRAY statement.

```
data _NULL_;
retain A1-A5 (1 * (10, 10, 10, 10, 10)) ;
vararray double A[*] A1-A5;
  method run();
put _ALL_;
;
_return: ;
end;
enddata;
```


EXPORT Procedure

Overview: EXPORT Procedure	643
What Does the EXPORT Procedure Do?	643
Format Catalog Encodings in SAS Viya Platform	644
Support for the VARCHAR Data Type	644
Syntax: EXPORT Procedure	645
PROC EXPORT Statement	646
DBENCODING Statement	651
DELIMITER Statement	651
FMTLIB Statement	652
META Statement	652
PUTNAMES Statement	653
Examples: EXPORT Procedure	654
Example 1: Exporting to a Delimited External Data Source	654
Example 2: Exporting a Subset of Observations to a CSV File	658
Example 3: Exporting to a Tab Delimited File with the PUTNAMES= Statement ..	661

Overview: EXPORT Procedure

What Does the EXPORT Procedure Do?

The EXPORT procedure reads data from a SAS data set or table and writes it to an external data source. External data sources include:

- dBase files
- delimited files
- JMP files

- Paradox files
- SPSS files
- Stata files

The CAS engine also supports JMP, SPSS, and Stata files for exporting a CAS table in memory on the CAS server to the PC file format. To use the CAS engine to import or export CAS data, a connection to the CAS server must be made. When exporting CAS tables, all VARCHAR data types are converted to the CHAR data type in the PC file format. For more information about the CAS engine and the CAS LIBNAME statement, see [SAS Cloud Analytic Services: User's Guide](#).

In delimited files, a delimiter can be a blank, comma, or tab that separates columns of data values. If you have a license for SAS/ACCESS Interface to PC Files, you can also export to additional file formats, such as to a Microsoft Access database, Microsoft Excel workbook, and DBF file..

You can export a SAS data set to a JMP 7 or later file, and JMP variables can be up to 255 characters long. Extended attributes are now used automatically, and the META= statement is no longer supported for JMP files. For more information, see ["JMP Files" in SAS/ACCESS Interface to PC Files: Reference for the SAS Viya Platform](#).

The EXPORT procedure uses one of these methods to export data:

- generated DATA step code
- generated SAS/ACCESS code

You control the results with options and statements that are specific to the output data source. The EXPORT procedure generates the specified output file and writes information about the export to the SAS log. The log displays the DATA step or the SAS/ACCESS code that the EXPORT procedure generates. If a translation engine is used, then no code is submitted.

Format Catalog Encodings in SAS Viya Platform

SAS Viya platform supports only the UTF-8 encoding.

For more information about the encodings of format catalogs, see [Migrating Data to UTF-8 for the SAS Viya Platform](#) and [SAS/ACCESS Interface to PC Files: Reference for the SAS Viya Platform](#).

Support for the VARCHAR Data Type

PROC EXPORT supports the VARCHAR data type in CAS. VARCHAR stores a character variable that can have a varying length. The length that you specify for the variable represents the maximum number of characters that you want to store.

The VARCHAR data type is similar to the CHAR data type. CHAR variables have a length that is measured in terms of bytes. VARCHAR variables have a length that is

measured in terms of characters rather than bytes. For information about using VARCHAR, see [SAS Cloud Analytic Services: DATA Step Programming](#).

In the following example, the CAS engine is used with the LENGTH statement to create a VARCHAR variable and a CHAR variable. The VARCHAR variable, X, has a length of 30 and the CHAR variable, Y, also has a length of 30.

```
libname mycas cas;
data mycas.string;
  length x varchar(30);
  length y $30;
  x = 'abc'; y = 'def';
run;
proc contents data=mycas.string; run;
```

Here is the output that the code produces.

The CONTENTS Procedure			
Data Set Name	MYCAS.STRING	Observations	1
Member Type	DATA	Variables	2
Engine	CAS	Indexes	0
Created	09/11/2016 13:48:30	Observation Length	48
Last Modified	09/11/2016 13:48:30	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64		
Encoding	utf-8 Unicode (UTF-8)		

Engine/Host Dependent Information	
Data Limit	100MB

Alphabetic List of Variables and Attributes			
#	Variable	Type	Len
1	x	Varchar	30
2	y	Char	30

Syntax: EXPORT Procedure

Restrictions: The EXPORT procedure is available for the following operating environments:

- Linux

A pathname for a file can have a maximum length of 201 characters.

Tip: PROC EXPORT supports the VARCHAR data type for CAS tables. For more information, see [“Support for the VARCHAR Data Type” on page 644](#).

PROC EXPORT DATA=<libref | caslib.tablename.>SAS data set <(SAS data set options)>

OUTFILE="filename" | OUTTABLE="tablename"

```
<DBMS=identifier> <REPLACE> <LABEL>;
statements for exporting to delimited files

DELIMITER=char | 'nn'x;
PUTNAMES=YES | NO;
statements for exporting to JMP files

DBENCODING=12-char SAS encoding-value;
FMTLIB=<libref.>format-catalog;
META=libref.member-data-set;
```

Statement	Task	Example
PROC EXPORT	Export SAS data sets to an external data file	Ex. 1 , Ex. 2
DBENCODING	Indicate the encoding used to save data in JMP files	
DELIMITER	Specify the delimiter to separate columns of data in the delimited output file	Ex. 1
FMTLIB	Write SAS format values defined in the format catalog to the JMP file for the value labels	
META	Write SAS metadata information to the JMP file	
PUTNAMES	Write the SAS variable names as column headings to the first row of the exported data file.	Ex. 3

PROC EXPORT Statement

Exports SAS data sets to an external data file.

- Tips:

PROC EXPORT supports all access types that are available in the FILENAME statement.

PROC EXPORT supports the VARCHAR data type for CAS tables. For more information, see [“Support for the VARCHAR Data Type” on page 644](#).
- See:

[“FILENAME Statement” in SAS Global Statements: Reference](#)

Syntax

```
PROC EXPORT DATA=<libref SAS data set | caslib.tablename.> <(SAS data set options)>
  OUTFILE="filename" | OUTTABLE="tablename"
  <DBMS=identifier> <REPLACE> <LABEL>;
```

Summary of Optional Arguments

(SAS data set options)

specifies SAS data set options.

DBMS=*identifier*

specifies the type of external data source that the EXPORT procedure creates. To export to a DBMS table, you must specify the DBMS option by using a valid database identifier.

LABEL

specifies a variable label name.

REPLACE

overwrites an existing file.

Required Arguments

DATA=<libref.>SAS data set | caslib.tablename

identifies the input SAS data set with either a one- or two-level SAS name (library and member name). If you specify a one-level name, by default, the EXPORT procedure uses either the USER library (if assigned) or the WORK library.

caslib.tablename specifies the input CAS table. You must use the DATA= two-level name (caslib and table) because you cannot specify only a one-level name for a CAS table.

The EXPORT procedure can export a SAS data set only if the data target supports the format of a SAS data set. The amount of data must also be within the limitations of the data target. For example, some data files have a maximum number of rows or columns. Some data files cannot support SAS user-defined formats and informats. If the SAS data set that you want to export exceeds the limits of the target file, the EXPORT procedure might not be able to export it correctly. In many cases, the procedure attempts to convert the data to the best of its ability. However, conversion is not possible for some types.

A SAS data set name can contain a single quotation mark when the VALIDMEMNAME=EXTEND system option is also specified. Using VALIDMEMNAME= expands the rules for the names of certain SAS members, such as a SAS data set name. For more information, see [“Rules for Naming SAS Data Sets and Variables” in SAS Programmer’s Guide: Essentials](#).

Default	If you do not specify a SAS data set to export, the EXPORT procedure uses the most recently created SAS data set. SAS keeps track of the data sets with the system variable _LAST_. To be certain that the EXPORT procedure uses the correct data set, you should identify the SAS data set.
Restriction	PROC EXPORT does not support the DROP KEEP data set options with Name Range Lists for DBMS=CSV TAB DLM.
Examples	“Example 1: Exporting to a Delimited External Data Source” on page 654

[“Example 2: Exporting a Subset of Observations to a CSV File” on page 658](#)

OUTFILE=“*filename*” | “*fileref*”

specifies the complete path and file name or a fileref for the output PC file, spreadsheet, or delimited external file. A fileref is a SAS name that is associated with the physical location of a file. To assign a fileref, use the FILENAME statement.

If you specify a fileref, or if the complete path and file name do not include special characters (such as the backslash in a path), lowercase characters, or spaces, you can omit the quotation marks.

Alias FILE=

Restrictions The EXPORT procedure does not support device types or access methods for the FILENAME statement except for DISK. For example, the EXPORT procedure does not support the TEMP device type, which creates a temporary external file.

PROC EXPORT does not support the DROP|KEEP data set options with Name Range Lists for DBMS=CSV | TAB | DLM.

See [SAS/ACCESS Interface to PC Files: Reference for the SAS Viya Platform](#) for more information about PC file formats.

Examples [“Example 1: Exporting to a Delimited External Data Source” on page 654](#)

[“Example 2: Exporting a Subset of Observations to a CSV File” on page 658](#)

OUTTABLE=“*tablename*”

specifies the table name of the output DBMS table. If the name does not include special characters (such as question marks), lowercase characters, or spaces, you can omit the quotation marks. Note that the DBMS table name might be case sensitive.

Alias TABLE=

Requirements You must have a license for SAS/ACCESS Interface to PC Files to export to a DBMS table.

When you export a DBMS table, you must specify the DBMS option.

Optional Arguments

DBMS=*identifier*

specifies the type of external data source that the EXPORT procedure creates. To export to a DBMS table, you must specify the DBMS option by using a valid database identifier.

CSV

The output data source is a comma-separated delimited file. The file extension is .CSV.

DBF

dBASE 5.0, IV, III+, and III files. The file extension is .dbf.

DBFMEMO

dBASE 5.0, IV, III+, and III files with memos.

FoxPro and Visual FoxPro files with memos.

The file extensions are .dbf, .dbt, .fbt.

DLM

The output data source is a delimited file. You can use DELIMITER='char' to specify the type of data to export.

The file extension is .*

Default The default delimiter is a blank (or space).

DTA

Stata file. The file extension is .dta.

JMP

The output data source is a JMP file, Version 7 or later format. The file extension is .JMP.

PARADOX

Paradox DB files. The file extension is .db.

SAV

SPSS files, compressed and uncompressed binary files. The file extension is .sav.

TAB

The output data source is a tab delimited file. The file extension is .txt.

XLS

Microsoft Excel 97, 2000, 2002, or 2003 spreadsheet using file formats. The file extension is .xls.

XLSX

Excel 2007, 2010, and later spreadsheet, using .xlsx file format. The file extension is .xlsx.

Transcoding is not supported for DBMS=XLS. Save your .xls file to an .xlsx file and use DBMS=XLSX to support transcoding.

You can specify DBMS=XLSX as well as DBMS=XLS in order to read and write to Excel workbooks under Linux. You do not need to use the SAS PC Files Server.

Note: SAS recommends using .xlsx for its support and enhancements.

When exporting to existing Excel workbook .xls or .xlsx file, you can use DBMS=XLS (or DBMS=XLSX). However, a .bak file is created only when DBMS=XLS or DBMS=XLSX is used.

This table shows the files that SAS creates using various versions of Microsoft Excel that you can open and read.

Table 20.1 *Exported Data: Microsoft Excel Workbook Readability*

File extension	Excel 2007 and later	Excel 97, 2000, 2002, 2003
.xlsx	Yes	No
.xls	Yes	Yes

If you use the EXPORT procedure with DBMS=XLSX and the page has formulas that reference other pages, an error message is written to the log. The error states that the page cannot be replaced because it has formulas that reference other pages.

This table specifies the DBMS data source identifiers to use for Excel files.

Table 20.2 *DBMS Identifiers*

DBMS=	Linux
EXCEL	no
XLS	yes
XLSX	yes

Restrictions Only Excel 2007 and later can use .xlsx format.

The availability of an output external data source depends on the operating environment and, in some cases, the platform as specified in the previous table. The availability also depends on whether your site has a license for SAS/ACCESS Interface to PC Files. If you do not have a license, only delimited and JMP files are available.

Example [“Example 1: Exporting to a Delimited External Data Source” on page 654](#)

LABEL

specifies a variable label name. SAS writes these to the exported table as column names. If the label names do not already exist, SAS writes them to the exported table.

REPLACE

overwrites an existing file. If you do not specify REPLACE, the EXPORT procedure does not overwrite an existing file.

Example [“Example 2: Exporting a Subset of Observations to a CSV File” on page 658](#)

(SAS data set options)

specifies SAS data set options. For example, if the data set that you are exporting has an assigned password, you can use the ALTER=, PW=, READ=, or WRITE= data set options. To export a subset of data that meets a specified condition, you can use the WHERE option. For information about SAS data set options, see [SAS Data Set Options: Reference](#).

Example [“Example 2: Exporting a Subset of Observations to a CSV File” on page 658](#)

DBENCODING Statement

Indicates the encoding used to save data in JMP files.

Interaction: The DBENCODING statement is valid only when DBMS=JMP.

Syntax

DBENCODING=*12-char SAS encoding-value*;

Required Argument

12-char SAS encoding-value

indicates the encoding used to save data in JMP files. Encoding maps each character in a character set to a unique numeric representation, which results in a table of code points. A single character can have different numeric representations in different encodings. This value can be up to 12 characters long.

DELIMITER Statement

Specifies the delimiter to separate columns of data in the output file.

Default: Blank space

Interaction: If you specify DBMS=DLM, you must also specify the DELIMITER statement.

Tip: You can enclose *nn* in single or double quotation marks.

Example: [“Example 1: Exporting to a Delimited External Data Source” on page 654](#)

Syntax

DELIMITER=*char* | 'nnx';

Required Argument

***char* | 'nn'x**

specifies the delimiter to use to separate values in the output file. You can specify the delimiter as a single BYTE character or as a hexadecimal value. For example, if you want columns of data to be separated by an ampersand, specify DELIMITER='&'. A single character that requires two bytes to be represented in a DBCS environment is not valid.

FMTLIB Statement

Write SAS format values defined in the format catalog to the JMP file for the value labels.

Interaction: The FMTLIB statement is valid only when DBMS=JMP.

Syntax

FMTLIB=<libref> *format-catalog*;

Required Argument

<libref>format-catalog

specifies the format catalog to be written to the JMP file.

META Statement

Writes SAS metadata information to the JMP file. (Deprecated)

Interaction: The META statement is valid only when DBMS=JMP.

Syntax

META=*libref.member-data-set*;

Required Argument

libref.member-data-set

specifies the SAS data set that contains the metadata information to be written to the JMP file.

The META statement is no longer supported and is ignored. Instead, extended attributes are automatically used. When exporting a SAS data set to JMP, PROC EXPORT looks for extended attributes on the SAS data set. If the attributes exist, the procedure uses the attributes to build the new JMP file.

The META statement can remain in your programs, yet it generates a NOTE in the log saying that META has been replaced by extended attributes and is ignored.

PUTNAMES Statement

Writes SAS variable names as column headings to the first row of the exported data file.

Default: YES

Restriction: Valid only for the EXPORT procedure.

Note: If you specify the LABEL= option, the SAS variable labels (not the variable names) are written as column headings.

Example: [“Example 3: Exporting to a Tab Delimited File with the PUTNAMES= Statement” on page 661](#)

Syntax

PUTNAMES=YES | NO;

Required Arguments

YES

specifies that the EXPORT procedure is to do the following tasks:

- Write the SAS variable names as column names (or headings) to the first row of the exported data file.
- Write the first row of the SAS data set to the second row of the exported data file.

NO

specifies that the EXPORT procedure is to write the first row of SAS data set values to the first row of the exported data file.

Examples: EXPORT Procedure

Example 1: Exporting to a Delimited External Data Source

Features: PROC EXPORT statement options
 DATA=
 DBMS=
 OUTFILE=
 REPLACE
 DELIMITER= statement

Details

This example exports the Sashelp.Class data set to a delimited external file. The following example is the Sashelp.Class data set before it is exported:

Output 20.1 PROC PRINT of Sashelp.Class

The SAS System					
Obs	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69.0	112.5
2	Alice	F	13	56.5	84.0
3	Barbara	F	13	65.3	98.0
4	Carol	F	14	62.8	102.5
5	Henry	M	14	63.5	102.5
6	James	M	12	57.3	83.0
7	Jane	F	12	59.8	84.5
8	Janet	F	15	62.5	112.5
9	Jeffrey	M	13	62.5	84.0
10	John	M	12	59.0	99.5
11	Joyce	F	11	51.3	50.5
12	Judy	F	14	64.3	90.0
13	Louise	F	12	56.3	77.0
14	Mary	F	15	66.5	112.0
15	Philip	M	16	72.0	150.0
16	Robert	M	12	64.8	128.0
17	Ronald	M	15	67.0	133.0
18	Thomas	M	11	57.5	85.0
19	William	M	15	66.5	112.0

Program

```
proc export data=sashelp.class
  outfile="c:\myfiles\class"
  dbms=dlm replace;

  delimiter='&';
run;
```

Program Description

Specify the input data set. Note that the filename does not contain an extension. DBMS=DLM specifies that the output file is a delimited file.

```
proc export data=sashelp.class  
  outfile="c:\myfiles\class"  
  dbms=dlm replace;
```

The DELIMITER option specifies that an & (ampersand) will delimit data fields in the output file.

```
  delimiter='&;  
run;
```

Log

This partial SAS log displays this information about the successful export, including the generated SAS DATA step.

Example Code 20.1 SAS Log of Creating a Delimited File

```

1  proc export data=sashelp.class outfile="c:\myfiles\class" dbms=dlm replace;
1  !                                     delimiter='&' ; run;

2  /*****
3  *   PRODUCT:    SAS
4  *   VERSION:    9.3
5  *   CREATOR:    External File Interface
6  *   DATE:       31JAN11
7  *   DESC:       Generated SAS Datastep Code
8  *   TEMPLATE SOURCE: (None Specified.)
9  *****/
10  data _null_;
11  %let _EFIERR_ = 0; /* set the ERROR detection macro variable */
12  %let _EFIREC_ = 0; /* clear export record count macro variable */
13  file 'c:\myfiles\class' delimiter='&' DSD DROPOVER lrecl=32767;
14  if _n_ = 1 then          /* write column names or labels */
15  do;
16      put
17          "Name"
18          '&'
19          "Sex"
20          '&'
21          "Age"
22          '&'
23          "Height"
24          '&'
25          "Weight"
26      ;
27  end;
28  set SASHELP.CLASS end=EFIEOD;
29  format Name $8. ;
30  format Sex $1. ;
31  format Age best12. ;
32  format Height best12. ;
33  format Weight best12. ;
34  do;
35      EFIOUT + 1;
36      put Name $ @;
37      put Sex $ @;
38      put Age @;
39      put Height @;
40      put Weight ;
41  ;
42  end;
43  if _ERROR_ then call symputx('_EFIERR_',1); /* set ERROR detection macro variable */
44  if EFIEOD then call symputx('_EFIREC_',EFIOUT);
45  run;

NOTE: The file 'c:\myfiles\class' is:
      Filename=c:\myfiles\class,
      RECFM=V,LRECL=32767,File Size (bytes)=0,
      Last Modified=31Jan2011:09:37:14,
      Create Time=31Jan2011:09:37:14

```

Output

The EXPORT procedure produces this external file:

Output 20.2 External File

```

Name&Sex&Age&Height&Weight
Alfred&M&14&69&112.5
Alice&F&13&56.5&84
Barbara&F&13&65.3&98
Carol&F&14&62.8&102.5
Henry&M&14&63.5&102.5
James&M&12&57.3&83
Jane&F&12&59.8&84.5
Janet&F&15&62.5&112.5
Jeffrey&M&13&62.5&84
John&M&12&59&99.5
Joyce&F&11&51.3&50.5
Judy&F&14&64.3&90
Louise&F&12&56.3&77
Mary&F&15&66.5&112
Philip&M&16&72&150
Robert&M&12&64.8&128
Ronald&M&15&67&133
Thomas&M&11&57.5&85
William&M&15&66.5&112

```

Example 2: Exporting a Subset of Observations to a CSV File

Features: PROC EXPORT statement options

- DATA=
- DBMS=
- OUTFILE=
- REPLACE

Details

This example exports the SAS data set Sashelp.Class to a delimited file.

Program

Specify the data set to be exported. The WHERE option requests a subset of the observations. The OUTFILE= option specifies the output file. The DBMS= option specifies that the output file is a CSV file, and overwrites the target CSV, if it exists.

```

proc export data=sashelp.class (where=(sex='F'))
  outfile="c:\myfiles\Femalelist.csv"

```



```
dbms=csv  
replace;  
run;
```

Log

This partial SAS log displays this information about the successful export, including the generated SAS DATA step.

Example Code 20.2 Exporting to a CSV File

```

579 proc export data=sashelp.class (where=(sex='F'))
580     outfile="c:\myfiles\Femalelist.csv"
581     dbms=csv
582     replace;
583 run;

584 /*****
585 *   PRODUCT:   SAS
586 *   VERSION:   9.4
587 *   CREATOR:   External File Interface
588 *   DATE:      18APR14
589 *   DESC:      Generated SAS Daststep Code
590 *   TEMPLATE SOURCE: (None Specified.)
591 *****/
592 data _null_;
593     %let _EFIERR_ = 0; /* set the ERROR detection macro variable */
594     %let _EFIREC_ = 0; /* clear export record count macro variable */
595     file 'c:\myfiles\Femalelist.csv' delimiter=',' DSD DROPOVER lrecl=32767
596 ! ;
597     if _n_ = 1 then          /* write column names or labels */
598     do;
599         put
600             "Name"
601             ' '
602             "Sex"
603             ' '
604             "Age"
605             ' '
606             "Height"
607             ' '
608             "Weight"
609         ;
610     set SASHELP.CLASS(where=(sex='F')) end=EFIEOD;
611     format Name $8. ;
612     format Sex $1. ;
613     format Age best12. ;
614     format Height best12. ;
615     format Weight best12. ;
616     do;
617         EFIOUT + 1;
618         put Name $ @;
619         put Sex $ @;
620         put Age @;
621         put Height @;
622         put Weight ;
623     ;
624     end;
625     if _ERROR_ then call symputx('_EFIERR_',1); /* set ERROR detection
626 ! macro variable */
627     if EFIEOD then call symputx('_EFIREC_',EFIOUT);
628 run;

```

NOTE: The file 'c:\myfiles\Femalelist.csv' is:
 Filename=c:\myfiles\Femalelist.csv,
 RECFM=V,LRECL=32767,File Size (bytes)=0,
 Last Modified=18Apr2014:11:32:07,
 Create Time=18Apr2014:11:32:07

Output

The EXPORT procedure produces this external CSV file:

Output 20.3 CSV File

Name	Sex	Age	Height	Weight
Alice	F	13	56.5	84
Barbara	F	13	65.3	98
Carol	F	14	62.8	102.5
Jane	F	12	59.8	84.5
Janet	F	15	62.5	112.5
Joyce	F	11	51.3	50.5
Judy	F	14	64.3	90
Louise	F	12	56.3	77
Mary	F	15	66.5	112

Example 3: Exporting to a Tab Delimited File with the PUTNAMES= Statement

Features: PROC EXPORT statement options
DATA=
DBMS=
OUTFILE=
PUTNAMES=
REPLACE

Details

This example shows exporting a SAS data set, WORK.INVOICE, to a tab-delimited file. The first program uses PROC EXPORT with the PUTNAMES= statement and the second program does not. They show how the use of this statement affects column headings in a tab-delimited file.

The following display shows the SAS data set, WORK.INVOICE, before it is exported to a tab-delimited file:

Output 20.4 PROC PRINT of WORK.INVOICE

Obs	Invoice_ID	Billed_To	Amount_Billed_in_Local_Currency	Country	Amount_Billed_in_US_Dollars	Billed_By	Billed_On	Paid_On
1	11270	39045213	8738600640.00	Brazil	3875848866.21	239185	05OCT2004	18OCT2004
2	11271	18543489	11063836.00	USA	11063836.00	457232	05OCT2004	.
3	11273	19783482	252148.50	USA	252148.50	239185	08OCT2004	11NOV2004
4	11276	14324742	1934460.00	USA	1934460.00	135673	09OCT2004	20OCT2004
5	11278	14868028	1400825.00	USA	1400825.00	239185	09OCT2004	19OCT2004
6	11280	39045213	8738600640.00	Brazil	3875848866.21	423286	07OCT2004	20OCT2004
7	11282	19783482	252148.50	USA	252148.50	457232	07OCT2004	25OCT2004
8	11285	38763919	2234301.30	Argentina	772847.05	239185	10OCT2004	30NOV2004
9	11286	43459747	14938604.08	Australia	11386352.06	423286	10OCT2004	.
10	11287	15432147	252148.50	USA	252148.50	457232	11OCT2004	04NOV2004
11	12051	39045213	8738600640.00	Brazil	3875848866.21	457232	02NOV2004	.
12	12102	18543489	11063836.00	USA	11063836.00	239185	17NOV2004	.
13	12283	19783482	252148.50	USA	252148.50	423286	05DEC2004	.
14	12468	14868028	1400825.00	USA	1400825.00	135673	24DEC2004	02JAN2005
15	12471	39045213	8738600640.00	Brazil	3875848866.21	457232	27DEC2004	.
16	12476	38763919	2234301.30	Argentina	772847.05	135673	24DEC2004	.
17	12478	15432147	252148.50	USA	252148.50	423286	24DEC2004	02JAN2005

Program

```

PROC PRINT DATA=WORK.INVOICE;
RUN;

PROC EXPORT DATA=WORK.INVOICE
  OUTFILE="c:\temp\invoice_names.txt"
  DBMS=TAB REPLACE;
  PUTNAMES=YES;
RUN;
PROC PRINT;
RUN;

PROC EXPORT DATA=WORK.INVOICE
  OUTFILE="c:\temp\invoice_data_1st.txt"
  DBMS=TAB REPLACE;
  PUTNAMES=NO;
RUN;

PROC PRINT;
RUN;

```

Program Description

Use the PUTNAMES=YES statement in the EXPORT procedure. After WORK.INVOICE is printed, using the PUTNAMES=YES statement writes the SAS variables names as column names to the first row of the exported delimited file,

Invoice_names.txt. The first row of data is then written to the second row of the delimited file.

```
PROC PRINT DATA=WORK.INVOICE;
RUN;

PROC EXPORT DATA=WORK.INVOICE
  OUTFILE="c:\temp\invoice_names.txt"
  DBMS=TAB REPLACE;
  PUTNAMES=YES;
RUN;
PROC PRINT;
RUN;
```

Impact of the PUTNAMES=NO statement. When you set this statement to NO, PROC EXPORT writes the first row of data to the first row of the exported delimited file. Therefore, the SAS variable names are skipped, and the columns are left unlabeled.

```
PROC EXPORT DATA=WORK.INVOICE
  OUTFILE="c:\temp\invoice_data_1st.txt"
  DBMS=TAB REPLACE;
  PUTNAMES=NO;
RUN;

PROC PRINT;
RUN;
```

SAS Log

This SAS log displays information about the successful export, including the generated SAS DATA step. The log is divided into sections only for documentation appearances.

```

490 PROC EXPORT DATA= WORK.INVOICE
491             OUTFILE= "c:\temp\invoice_names.txt"
492             DBMS=TAB REPLACE;
493             PUTNAMES=YES;
494 RUN;

495 /*****
496 *   PRODUCT:   SAS
497 *   VERSION:   9.4
498 *   CREATOR:   External File Interface
499 *   DATE:      24MAY14
500 *   DESC:      Generated SAS Datasheet Code
501 *   TEMPLATE SOURCE: (None Specified.)
502 *****/
503 data _null_;
504     %let _EFIERR_ = 0; /* set the ERROR detection macro variable */
505     %let _EFIREC_ = 0; /* clear export record count macro variable */
506     file 'c:\temp\invoice_names.txt' delimiter='09'x DSD DROPOVER lrecl=32767;
507     if _n_ = 1 then /* write column names or labels */
508     do;
509         put
510             "INVNUM"
511             '09'x
512             "BILLEDTO"
513             '09'x
514             "AMTBILL"
515             '09'x
516             "COUNTRY"
517             '09'x
518             "AMTINUS"
519             '09'x
520             "BILLEDBY"
521             '09'x
522             "BILLEDON"
523             '09'x
524             "PAIDON"
525         ;
526     end;
527     set WORK.INVOICE end=EFIEOD;
528     format INVNUM best12. ;
529     format BILLEDTO $8. ;
530     format AMTBILL dollar18.2 ;
531     format COUNTRY $20. ;
532     format AMTINUS dollar18.2 ;
533     format BILLEDBY best12. ;
534     format BILLEDON date9. ;
535     format PAIDON date9. ;
536     do;
537         EFIOUT + 1;
538         put INVNUM @;
539         put BILLEDTO $ @;
540         put AMTBILL @;
541         put COUNTRY $ @;
542         put AMTINUS @;
543         put BILLEDBY @;
544         put BILLEDON @;
545         put PAIDON ;
546     ;
547     end;
548     if _ERROR_ then call symputx('_EFIERR_',1); /* set ERROR detection macro
variable */
549     if EFIEOD then call symputx('_EFIREC_',EFIOUT);
550 run;

```

```

NOTE: The file 'c:\temp\invoice_names.txt' is:
      Filename=c:\temp\invoice_names.txt,
      RECFM=V,LRECL=32767,File Size (bytes)=0,
      Last Modified=24May2014:15:46:36,
      Create Time=24May2014:15:46:36

NOTE: 18 records were written to the file 'c:\temp\invoice_names.txt'.
      The minimum record length was 60.
      The maximum record length was 84.
NOTE: There were 17 observations read from the data set WORK.INVOICE.
NOTE: DATA statement used (Total process time):
      real time          0.04 seconds
      cpu time           0.03 seconds

17 records created in c:\temp\invoice_names.txt from WORK.INVOICE.

NOTE: "c:\temp\invoice_names.txt" file was successfully created.
NOTE: PROCEDURE EXPORT used (Total process time):
      real time          0.20 seconds
      cpu time           0.17 seconds

551  PROC PRINT; RUN;

NOTE: No observations in data set SASUSER.SASMBC.
NOTE: PROCEDURE PRINT used (Total process time):
      real time          0.01 seconds
      cpu time           0.01 seconds

552
553
554  PROC EXPORT DATA= WORK.INVOICE
555          OUTFILE= "c:\temp\invoice_data_1st.txt"
556          DBMS=TAB REPLACE;
557          PUTNAMES=NO;
558  RUN;

```

```

559  /*****
560  *   PRODUCT:   SAS
561  *   VERSION:   9.4
562  *   CREATOR:   External File Interface
563  *   DATE:      24MAY14
564  *   DESC:      Generated SAS Datasheet Code
565  *   TEMPLATE SOURCE: (None Specified.)
566  *****/
567  data _null_;
568  %let _EFIERR_ = 0; /* set the ERROR detection macro variable */
569  %let _EFIREC_ = 0; /* clear export record count macro variable */
570  file 'c:\temp\invoice_data_1st.txt' delimiter='09'x DSD DROPOVER
  lrecl=32767;
571  set WORK.INVOICE end=EFIEOD;
572  format INVNUM best12. ;
573  format BILLEDTO $8. ;
574  format AMTBILL dollar18.2 ;
575  format COUNTRY $20. ;
576  format AMTINUS dollar18.2 ;
577  format BILLEDBY best12. ;
578  format BILLEDON date9. ;
579  format PAIDON date9. ;
580  do;
581  EFIOUT + 1;
582  put INVNUM @;
583  put BILLEDTO $ @;
584  put AMTBILL @;
585  put COUNTRY $ @;
586  put AMTINUS @;
587  put BILLEDBY @;
588  put BILLEDON @;
589  put PAIDON ;
590  ;
591  end;
592  if _ERROR_ then call symputx('_EFIERR_',1); /* set ERROR detection macro
variable */
593  if EFIEOD then call symputx('_EFIREC_',EFIOUT);
594  run;

NOTE: The file 'c:\temp\invoice_data_1st.txt' is:
      Filename=c:\temp\invoice_data_1st.txt,
      RECFM=V,LRECL=32767,File Size (bytes)=0,
      Last Modified=24May2014:15:46:36,
      Create Time=24May2014:15:46:36

NOTE: 17 records were written to the file 'c:\temp\invoice_data_1st.txt'.
      The minimum record length was 60.
      The maximum record length was 84.
NOTE: There were 17 observations read from the data set WORK.INVOICE.
NOTE: DATA statement used (Total process time):
      real time           0.03 seconds
      cpu time            0.03 seconds

17 records created in c:\temp\invoice_data_1st.txt from WORK.INVOICE.

NOTE: "c:\temp\invoice_data_1st.txt" file was successfully created.
NOTE: PROCEDURE EXPORT used (Total process time):
      real time           0.07 seconds
      cpu time            0.07 seconds

595
596  PROC PRINT; RUN;

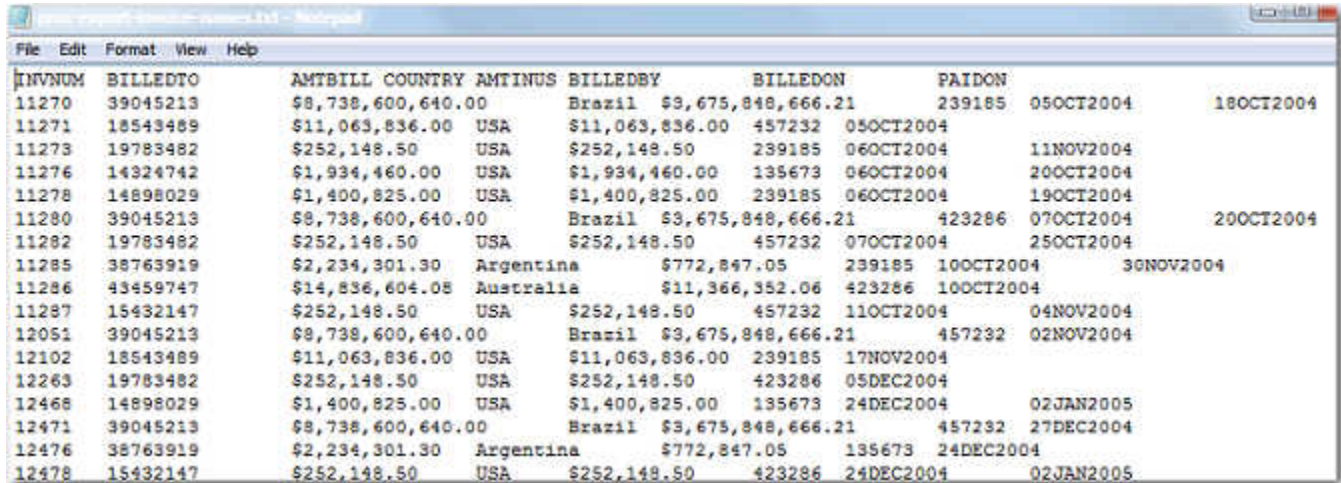
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds

```


Output Example

Using the PROC EXPORT PUTNAMES=YES statement, the SAS variable names are mapped to the column headings in the tab-delimited file. This is the default behavior.

Output 20.5 SAS Data Exported to a Tab-Delimited File Using the PUTNAMES=YES Statement

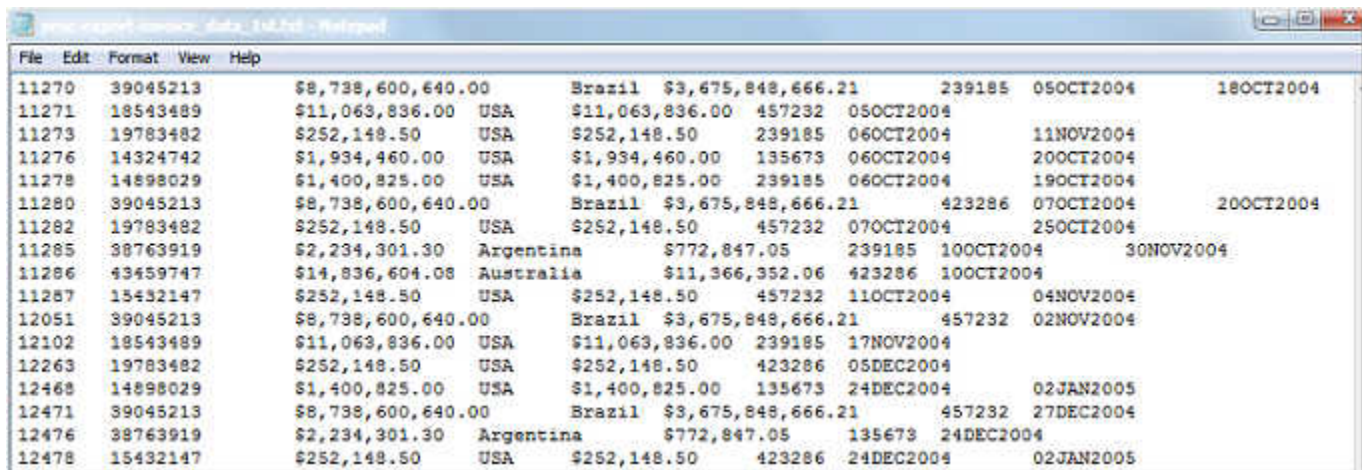


INNUM	BILLEDTO	AMTBILL	COUNTRY	AMTINUS	BILLEDBY	BILLEDON	PAIDON	
11270	39045213	\$8,738,600,640.00		Brazil	\$3,675,848,666.21	239185	05OCT2004	18OCT2004
11271	18543489	\$11,063,836.00	USA	\$11,063,836.00	457232	05OCT2004		
11273	19783482	\$252,148.50	USA	\$252,148.50	239185	06OCT2004	11NOV2004	
11276	14324742	\$1,934,460.00	USA	\$1,934,460.00	135673	06OCT2004	20OCT2004	
11278	14898029	\$1,400,825.00	USA	\$1,400,825.00	239185	06OCT2004	19OCT2004	
11280	39045213	\$8,738,600,640.00		Brazil	\$3,675,848,666.21	423286	07OCT2004	20OCT2004
11282	19783482	\$252,148.50	USA	\$252,148.50	457232	07OCT2004	25OCT2004	
11285	38763919	\$2,234,301.30	Argentina	\$772,847.05	239185	10OCT2004	30NOV2004	
11286	43459747	\$14,836,604.08	Australia	\$11,366,352.06	423286	10OCT2004		
11287	15432147	\$252,148.50	USA	\$252,148.50	457232	11OCT2004	04NOV2004	
12051	39045213	\$8,738,600,640.00		Brazil	\$3,675,848,666.21	457232	02NOV2004	
12102	18543489	\$11,063,836.00	USA	\$11,063,836.00	239185	17NOV2004		
12263	19783482	\$252,148.50	USA	\$252,148.50	423286	05DEC2004		
12468	14898029	\$1,400,825.00	USA	\$1,400,825.00	135673	24DEC2004	02JAN2005	
12471	39045213	\$8,738,600,640.00		Brazil	\$3,675,848,666.21	457232	27DEC2004	
12476	38763919	\$2,234,301.30	Argentina	\$772,847.05	135673	24DEC2004		
12478	15432147	\$252,148.50	USA	\$252,148.50	423286	24DEC2004	02JAN2005	

Output Example

Using the PROC EXPORT PUTNAMES=NO statement results in unnamed columns in the tab-delimited file.

Output 20.6 SAS Data Exported to a Tab-Delimited File Using the PUTNAMES=NO Statement



11270	39045213	\$8,738,600,640.00		Brazil	\$3,675,848,666.21	239185	05OCT2004	18OCT2004
11271	18543489	\$11,063,836.00	USA	\$11,063,836.00	457232	05OCT2004		
11273	19783482	\$252,148.50	USA	\$252,148.50	239185	06OCT2004	11NOV2004	
11276	14324742	\$1,934,460.00	USA	\$1,934,460.00	135673	06OCT2004	20OCT2004	
11278	14898029	\$1,400,825.00	USA	\$1,400,825.00	239185	06OCT2004	19OCT2004	
11280	39045213	\$8,738,600,640.00		Brazil	\$3,675,848,666.21	423286	07OCT2004	20OCT2004
11282	19783482	\$252,148.50	USA	\$252,148.50	457232	07OCT2004	25OCT2004	
11285	38763919	\$2,234,301.30	Argentina	\$772,847.05	239185	10OCT2004	30NOV2004	
11286	43459747	\$14,836,604.08	Australia	\$11,366,352.06	423286	10OCT2004		
11287	15432147	\$252,148.50	USA	\$252,148.50	457232	11OCT2004	04NOV2004	
12051	39045213	\$8,738,600,640.00		Brazil	\$3,675,848,666.21	457232	02NOV2004	
12102	18543489	\$11,063,836.00	USA	\$11,063,836.00	239185	17NOV2004		
12263	19783482	\$252,148.50	USA	\$252,148.50	423286	05DEC2004		
12468	14898029	\$1,400,825.00	USA	\$1,400,825.00	135673	24DEC2004	02JAN2005	
12471	39045213	\$8,738,600,640.00		Brazil	\$3,675,848,666.21	457232	27DEC2004	
12476	38763919	\$2,234,301.30	Argentina	\$772,847.05	135673	24DEC2004		
12478	15432147	\$252,148.50	USA	\$252,148.50	423286	24DEC2004	02JAN2005	

FCMP Procedure

Overview: FCMP Procedure	669
What Does the FCMP Procedure Do?	670
Concepts: FCMP Procedure	671
Creating Functions and Subroutines	671
Creating Functions and Subroutines: An Example	672
Writing Your Own Functions	673
Using Functions as Formats	677
Using DATA Step Statements with PROC FCMP	677
Syntax: FCMP Procedure	677
PROC FCMP Statement	679
ABORT Statement	684
ARRAY Statement	684
ATTRIB Statement	686
DECLARE Statement	687
DELETFUNC Statement	688
DELETESUBR Statement	688
FUNCTION Statement	689
LABEL Statement	690
LENGTH Statement	691
LISTFUNC Statement	692
LISTSUBR Statement	693
OUTARGS Statement	693
STATIC Statement	694
STRUCT Statement	696
SUBROUTINE Statement	697
Usage: FCMP Procedure	698
PROC FCMP and DATA Step Differences	698
Working with Arrays	706
Using Macros with PROC FCMP Routines	707
Variable Scope in PROC FCMP Routines	707
Recursion	709
Directory Traversal	710
Identifying the Location of Compiled Functions and Subroutines:	
The CMPLIB= System Option	714
Using PROC FCMP Component Objects	723
PROC FCMP and ASTORE	723
Using Python Functions in PROC FCMP	729

Returning a Variable by Name Using VNAME Arguments	730
Examples: FCMP Procedure	731
Example 1: Creating a Function and Calling the Function from a DATA Step	731
Example 2: Creating and Saving Functions with PROC FCMP	733
Example 3: Using Numeric Data in the FUNCTION Statement	735
Example 4: Using Character Data with the FUNCTION Statement	736
Example 5: Using Variable Arguments with an Array	737
Example 6: Using the SUBROUTINE Statement with a CALL Statement	738
Example 7: Using Graph Template Language (GTL) with User- Defined Functions	738
Example 8: Standardizing Each Row of a Data Set	741
References	745

Overview: FCMP Procedure

What Does the FCMP Procedure Do?

The SAS Function Compiler (FCMP) procedure enables you to create, test, and store SAS functions, CALL routines, and subroutines before you use them in other SAS procedures or DATA steps. PROC FCMP provides the ability to build functions, CALL routines, and subroutines using DATA step syntax that is stored in a data set. The procedure accepts slight variations of DATA step statements, and you can use most features of the SAS programming language in functions and CALL routines that are created by PROC FCMP. You can call PROC FCMP functions and CALL routines from the DATA step just as you would any other SAS function, CALL routine, or subroutine. This feature enables programmers to more easily read, write, and maintain complex code with independent and reusable subroutines. You can reuse the PROC FCMP routines in any DATA step or SAS procedure that has access to their storage location.

The FCMP procedure uses the SAS language compiler to compile and execute SAS programs. The compiler subsystem generates machine language code for the computer on which SAS is running. By specifying values with the CMPOPT option, the machine language code can be optimized for efficient execution. For information about the type of code generation optimizations to use in the SAS language compiler, see [“CMPOPT= System Option” in SAS System Options: Reference](#).

You can use the functions and subroutines that you create in PROC FCMP with the DATA step, the WHERE statement, the Output Delivery System (ODS), and with the following procedures:

PROC CALIS	PROC OPTMODEL
------------	---------------

PROC FORMAT	PROC PHREG
PROC GA	PROC QUANTREG
PROC GENMOD	PROC REPORT COMPUTE blocks
PROC GLIMMIX	SAS Risk Dimensions procedures
PROC MCMC	PROC SEVERITY
PROC MODEL	PROC SIMILARITY
PROC NLIN	PROC SQL (functions with array arguments are not supported)
PROC NLMIXED	PROC SURVEYPHREG
PROC NLP	PROC TMODEL
PROC OPTLSO	PROC VARMAX

Concepts: FCMP Procedure

Creating Functions and Subroutines

PROC FCMP enables you to write functions and CALL routines using DATA step syntax. PROC FCMP functions and CALL routines are stored in a data set and can be called from the DATA step, as well as several SAS/STAT, SAS/ETS, or SAS/OR procedures such as the NLIN, MODEL, and NLP procedures. You can create multiple functions and CALL routines in a single FCMP procedure step.

Functions are equivalent to routines that are used in other programming languages. They are independent computational blocks that require zero or more arguments. A subroutine is a special type of function where return values are optional. All variables that are created within a function or subroutine block are local to that subroutine.

Creating Functions and Subroutines: An Example

This following example defines a function and a subroutine. The function begins with the FUNCTION statement, and the subroutine begins with the SUBROUTINE statement. The DAY_DATE function converts a date to a numeric day of the week, and the INVERSE subroutine calculates a simple inverse. Each ends with an ENDSUB statement.

```
proc fcmp outlib=work.MySubs.Fncs;
  function day_date(indate);
    wkday=weekday(indate);
    return(wkday);
  endsub;

  subroutine inverse(in, inv);
    outargs inv;
    if in=0
      then inv=.;
      else inv=1/in;
    endsub;
quit;

option cmplib=work.MySubs;

data _null_;
  daysDate=day_date(today());
  put daysDate=;

  CALL inverse(12, inverseValue);
  put inverseValue=;
run;
```

These statements produce these results:

```
daysDate=2
inverseValue=0.0833333333
```

Note: The output will change based on the value of "today()".

The function and subroutine follow DATA step syntax. Functions and subroutines that are already defined in the current FCMP procedure step, as well as most DATA step functions, can be called from within these routines as well. In the example above, the DATA step function WEEKDAY is called by DAY_DATE.

The routines in the example are saved to the data set work.MySubs, inside a package called MathFncs. A package is any collection of related routines that are specified by the user. It is a way of grouping related subroutines and functions within the data set. The OUTLIB= option in the PROC FCMP statement tells PROC

FCMP where to store the subroutines that it compiles, and the LIBRARY= option tells it where to read in libraries (C or SAS).

Note: Function and subroutine names must be unique within a package. However, different packages can have subroutines and functions with the same names. To select a specific subroutine when there is ambiguity, use the package name and a period as the prefix to the subroutine name. For example, to access the MthFncs version of INVERSE, use MthFncs.inverse.

Writing Your Own Functions

Advantages of Writing Your Own Functions and CALL Routines

PROC FCMP enables you to write functions and CALL routines by using DATA step syntax. The advantages of writing user-defined functions and CALL routines include the following:

- The function or CALL routine makes a program easier to read, write, and modify.
- The function or CALL routine is independent. A program that calls a routine is not affected by the routine's implementation.
- The function or CALL routine is reusable. Any program that has access to the data set where the function or routine is stored can call the routine.

Note: PROC FCMP routines that you create cannot have the same name as built-in SAS functions. If the names are the same, then SAS generates an error message stating that a built-in SAS function or subroutine already exists with the same name.

IMPORTANT FCMP function names must not share leading characters with any licensed SAS function name. Conflicting leading characters can cause incorrect licensing checks that prevent the FCMP function from compiling or executing. SAS returns the following error:

ERROR: The product with which the function/subroutine is associated is either not licensed f

To resolve this error, you must rename your function so that the leading characters do not match the leading characters of any licensed SAS function names.

For more information, see the Knowledge Base article [PROC FCMP returns "ERROR: The product with which the function/subroutine is associated is either not licensed. .or the product license has expired"](#)

Writing a User-Defined Function

The following program shows the syntax that is used to create and call a PROC FCMP function from a DATA step. This example computes the study day during a drug trial.

The example creates a function named STUDY_DAY in a package named TRIAL. A package is a collection of routines that have unique names and is stored in the data set work.Funcs. STUDY_DAY accepts two numeric arguments, *intervention_date* and *event_date*. The body of the routine uses DATA step syntax to compute the difference between the two dates, where days that occur before *intervention_date* begin at -1 and become smaller, and days that occur after and including *intervention_date* begin at 1 and become larger. This function never returns 0 for a study day.

STUDY_DAY is called from DATA step code as if it were any other function. When the DATA step encounters a call to STUDY_DAY, it does not find this function in its traditional library of functions. Instead, SAS searches each of the libraries or data sets that are specified in the CMPLIB= system option for a package that contains STUDY_DAY. In this example, STUDY_DAY is located in work.Funcs.Trial. The program calls the function, passing the variable values for *start_date* and *end_date*, and returns the result in the variable *sd*.

```
proc fcmp outlib=work.funcs.trial;
  function study_day(intervention_date, event_date);
    n=event_date-intervention_date;
    if n <= 0 then
      n=n+1;
    return(n);
  endsub;

options cmplib=work.funcs;
data _null_;
  start_date='15Feb2006'd;
  end_date='27Mar2006'd;
  sd=study_day(start, today);
  put sd=;
run;
```

Example Code 21.1 Results from the STUDY_DAY User-Defined Function

```
sd=40
```


Using Library Options

You can use PROC FCMP with the OUTLIB= or INLIB= options. The syntax for this procedure has the following form:

```
proc fcmp
  outlib=libname.dataset.package
           inlib=in-libraries;
  routine-declarations;
```

The OUTLIB= option is required and specifies the package where routines declared in the *routine-declarations* section are stored.

Routines that are declared in the *routine-declarations* section can call FCMP routines that exist in other packages. To find these routines and to check the validity of the call, SAS searches the data sets that are specified in the INLIB= option. The format for the INLIB= option is as follows:

```
inlib=library.dataset
inlib=(library1.dataset1 library2.dataset2 ... libraryN.datasetN)
inlib=library.datasetM - library.datasetN
```

If the routines that are being declared do not call FCMP routines in other packages, then you do not need to specify the INLIB= option.

Declaring Functions

You declare one or more functions or CALL routines in the *routine-declarations* section of the program. A routine consists of four parts:

- a name
- one or more parameters
- a body of code
- a RETURN statement

You specify these four parts between the FUNCTION or SUBROUTINE keyword and an ENDSUB keyword. For functions, the syntax has the following form:

```
function
  name(argument-1 <, argument-2, ...>);
  program-statements;
  return(expression);
endsub;
```

After the FUNCTION keyword, you specify the name of the function and its arguments. Arguments in the function declaration are called formal arguments and can be used within the body of the function. To specify a string argument, place a dollar sign (\$) after the argument name. For functions, all arguments are passed by value. This means that the value of the actual argument, variable, or value that is passed to the function from the calling environment is copied before being used by

the function. This copying ensures that any modification of the formal argument by the function does not change the original value.

The RETURN statement is used to return a value to a function. The RETURN statement accepts an expression that is enclosed in parentheses, and contains the value that is returned to the calling environment. The function declaration ends with an ENDSUB statement.

Declaring CALL Routines

CALL routines are declared within *routine-declarations* by using the SUBROUTINE keyword instead of the FUNCTION keyword. Functions and CALL routines have the same form, except CALL routines do not return a value, and CALL routines can modify their parameters. You specify the arguments to be modified in an OUTARGS statement. The syntax of a CALL routine declaration is as follows:

```
subroutine
name(<argument-1 , argument-2, ...>);
  outargs <out-argument-1 , out-argument-2, ...>;
  program-statements;
  return;
endsub;
```

The formal arguments that are listed in the OUTARGS statement are passed by reference instead of by value. This means that any modification of the formal argument by the CALL routine modifies the original variable that was passed. It also means that the value is not copied when the CALL routine is invoked. Reducing the number of copies can improve performance when you pass large amounts of data between a CALL routine and the calling environment.

A RETURN statement is optional within the definition of the CALL routine. When a RETURN statement executes, execution is immediately returned to the caller. A RETURN statement within a CALL routine does not return a value.

Writing Program Statements

The program-statements section of the program is a series of DATA step and or FCMP statements that describe the work to be done by the function or CALL routine. Most DATA step statements and functions are accessible from PROC FCMP routines. The DATA step file and the data set I/O statements (for example, INPUT, FILE, SET, and MERGE) are not available from PROC FCMP routines. However, some functionality of the PUT statement is supported. For more information, see [“PROC FCMP and DATA Step Differences” on page 698](#).

Using Functions as Formats

PROC FCMP enables you to use functions to format values by first performing a function on a value. By using a function to format values, you can create customized formats.

Using DATA Step Statements with PROC FCMP

You can use DATA step statements with PROC FCMP. However, there are some differences in the syntax and functionality for PROC FCMP. For more information, see [“PROC FCMP and DATA Step Differences” on page 698](#).

The behaviors of the DROP, KEEP, FORMAT, and LENGTH statements are the same in PROC FCMP and in the DATA step.

The following DATA step statements are not supported in PROC FCMP:

- DATA
- SET
- MERGE
- UPDATE
- MODIFY
- INPUT
- INFILE

The support for the FILE statement is limited to LOG and PRINT destinations in PROC FCMP. The OUTPUT statement is supported in PROC FCMP, but it is not supported within a function or subroutine.

The following statements are supported in PROC FCMP but not in the DATA step:

- FUNCTION
- STRUCT
- SUBROUTINE
- OUTARGS

Syntax: FCMP Procedure

PROC FCMP *options*;

```

ABORT ;
ARRAY array-name [dimensions] < /NOSYMBOLS > | <variable(s)> |
  <constant(s)> | <initial-values> ;
ATTRIB variable(s) < FORMAT=format-name > < LABEL='label' > <
  LENGTH=length > ;
DELETFUNC function-name ;
DELETESUBR subroutine-name ;
FUNCTION function-name(argument(s)) < VARARGS > < $ > <length>
  <KIND | GROUP='string' < LABEL='string-2' >> ;
LABEL variable ='label' ;
LISTFUNC function-name ;
LISTSUBR subroutine-name ;
STRUCT structure-name variable ;
SUBROUTINE subroutine-name (argument(s)) < VARARGS >
  <LABEL='label'> <KIND | GROUP='string'> ;
OUTARGS out-argument(s) ;

```

Statement	Task	Example
PROC FCMP	Create, test, and store SAS functions for use by other SAS procedures	Ex. 1, Ex. 2, Ex. 8, Ex. 7
ABORT	Terminate the execution of the current DATA step, SAS job, or SAS session	
ARRAY	Associate a name with a list of variables and constants	Ex. 8
ATTRIB	Specify format, label, and length information for a variable	
DECLARE	Declare an object.	
DELETFUNC	Delete a function from the function library that is specified in the OUTLIB option	
DELETESUBR	Delete a subroutine from the function library that is specified in the OUTLIB option	
FUNCTION	Return changed variable values	Ex. 1, Ex. 2, Ex. 7
LABEL	Specify a label for variables	
	Specifies the number of bytes for storing character and numeric variables, or the number of characters for storing VARCHAR variables.	
LISTFUNC	Write the source code of a function in the SAS listing	

Statement	Task	Example
LISTSUBR	Write the source code for a subroutine in the SAS listing	
STATIC	retains a variable's value from a previous call until the variable is reassigned.	
STRUCT	Declare (create) structure types	
SUBROUTINE	Declare (create) independent computational blocks of code	Ex. 8
OUTARGS	(Use only with the SUBROUTINE statement.) Specify arguments from the argument list that the subroutine should update	Ex. 2 , Ex. 8

PROC FCMP Statement

Creates, tests, and stores SAS functions, CALL routines, and subroutines.

Examples:

[“Example 1: Creating a Function and Calling the Function from a DATA Step” on page 731](#)

[“Example 2: Creating and Saving Functions with PROC FCMP” on page 733](#)

[“Example 7: Using Graph Template Language \(GTL\) with User-Defined Functions” on page 738](#)

[“Example 8: Standardizing Each Row of a Data Set” on page 741](#)

Syntax

PROC FCMP *options*;

Optional Arguments

DATA=filename

reads an input data set into the PROC FCMP step.

Note: The DATA option can send inputs to a function or subroutine that are defined in the PROC FCMP step. The PROC FCMP step iterates through each observation and calls the function or subroutine during each iteration.

Note: For more information about the DATA= and OUT= options, see [“Data Set Input and Output” on page 699](#).

Note: This example demonstrates using the DATA= and OUT= options in the PROC FCMP statement.

```
Example  proc fcmp outlib=work.funcs.trial;
          function priceWithTax_func(price);
            static taxRate;
            if (taxRate EQ .) then
              taxRate = (7.25 / 100);
            taxedPrice = (price * (1 + taxRate));
            return (taxedPrice);
          endsub;
        run;

        option CMPLIB=work.funcs;

        proc fcmp data=sashelp.cars out=work.carPriceWithTax;
          format msrpWithTax DOLLAR8.;
          msrpWithTax = priceWithTax_func(msrp);
        run;

        proc print data=work.carPriceWithTax(obs=5 keep=make model
          invoice msrp msrpWithTax);
        run;
```

ENCRYPT

specifies to encode the source code in a data set.

Alias HIDE

FLOW

specifies printing a message for each statement in a program as it is executed. This option produces extensive output.

GETCASCMLIB

Alias GETCMPLIB

specifies to display the cmplib setting for CAS and certain environments. The result is output to the SAS log.

GETCASCMPOPT

Alias GETCMPOPT

specifies to display the cmpopt setting for CAS and certain environments. The result is output to the SAS log.

LIBRARY=*library.dataset*

LIBRARY=(*library-1.dataset library-2.dataset ... library-n.dataset*)

LIBRARY=*library.datasetM - library.datasetN*

specifies that previously compiled libraries are to be linked into the program. These libraries are created by a previous PROC FCMP step or by using PROC PROTO (for external C routines).

Alias INLIB

Tips Libraries are created by the OUTLIB= option and are stored as members of a SAS library that have the type CMPSUB. Only subroutines and functions are read into the program when you use the LIBRARY= option.

If the routines that are being declared do not call PROC FCMP routines in other packages, then you do not need to specify the LIBRARY= option. Use the *libref.dataset* format to specify the two-level name of a library. The *libref* and *dataset* names must be valid SAS names that are not longer than eight characters.

You can specify a list of files with the LIBRARY= option, and you can specify a range of names by using numeric suffixes. When you specify more than one file, you must enclose the list in parentheses, except in the case of a single range of names. The following are syntax examples:

```
proc fcmp library=work.exsubs
;

proc fcmp library=(work.exsubs
                    work.examples
                );

proc fcmp library=lib1-lib10
;

```

LIST

specifies that both the LISTSOURCE and LISTPROG options are in effect.

Tip Printing both the source code and the compiled code and then comparing the two listings of assignment statements is one way of verifying that the assignments were compiled correctly.

LISTALL

specifies that the LISTCODE, LISTPROG, and LISTSOURCE options are in effect.

LISTCODE

specifies that the compiled program code be printed. LISTCODE lists the chain of operations that are generated by the compiler.

Tip Because LISTCODE output is somewhat difficult to read, use the LISTPROG option to obtain a more readable listing of the compiled program code.

LISTFUNCS

specifies that prototypes for all visible FCMP functions or subroutines be written to the SAS listing.

LISTPROG

specifies that the compiled program be printed. The listing for assignment statements is generated from the chain of operations that are generated by the compiler. The source statement text is printed for other statements.

Tip The expressions that are printed by the LISTPROG option do not necessarily represent how the expression is actually calculated, because intermediate results for common subexpressions can be reused. However, the expressions are printed in expanded form by the LISTPROG option. To see how the expression is actually evaluated, see the listing from the LISTCODE option.

LISTSOURCE

specifies that source code statements for the program be printed.

OUT=filename

creates an output data set.

OUTFILE=filename

writes referenced functions and the main program to a text file. Programs that have been parsed by PROC FCMP, including macro variables, can be exported.

OUTITEMSTORE=path name

exports symbols, referenced functions, and the main program to the specified item store. OUTITEMSTORE does not support a fileref. You must use a quoted path.

OUTLIB=libname.dataset.package

specifies the three-level name of an output data set to which the compiled subroutines and functions are written when the PROC FCMP step ends. This argument is required. The following are syntax examples:

```
proc fcmp outlib=work.fcmpsups.pkt1
;

proc fcmp outlib=work.mysubs.math
;
```

Tips Use this option when you want to save subroutines and functions in an output library.

Only those subroutines that are declared inside the current PROC FCMP step are saved to the output file. Those subroutines that are loaded by using the LIBRARY= option are not saved to the output file. If you do not specify the OUTLIB= option, then no subroutines that are declared in the current PROC FCMP step are saved.

PRINT

specifies printing the result of each statement in a program as it is executed. This option produces extensive output.

SETCASCMLIB="library-1.dataset library-2.dataset library-n.dataset"

Alias SETCMLIB

specifies one or more SAS data sets that contain compiler subroutines to include during compilation for CAS and certain environments.

SETCASCMPOPT="option-1 option-2 option-n"

Alias SETCMPOPT

specifies one or more code generation optimizations for CAS and certain environments. For a list of code generation optimizations, see [CMPOPT](#)

TRACE

specifies printing the results of each operation in each statement in a program as it is executed. These results are produced in addition to the information that is printed by the FLOW option. The TRACE option produces extensive output.

Notes Specifying TRACE is equivalent to specifying FLOW, PRINT, and PRINTALL.

The TRACE option works when the data is contained within PROC FCMP, not when calling FCMP functions from the DATA step. This example and output demonstrate the functionality of the TRACE option.

Example

```
proc fcmp outlib=work.funcs.trial TRACE;
    function study_day(intervention_date, event_date);
        n=event_date - intervention_date;
        if n >= 0 then n=n + 1;
        return(n);
    endsub;
    start='15Feb2010'd;
    today='27Mar2010'd;
    sd=study_day(start, today);
run;
```

The FCMP Procedure

```
--- Program Execution Starting.
1   1 (20:4)   Executing Stmt           : ASSIGN start =
1       (20:9)   start = 18308
1   2 (21:4)   Executing Stmt           : ASSIGN today =
1       (21:9)   today = 18348
1   3 (22:4)   Executing Stmt           : ASSIGN sd =

--- Subroutine study_day Execution Starting.
1   1 (15:4)   Executing Stmt           : FUNCTION
1   2 (16:4)   Executing Stmt           : ASSIGN n =
1       (16:17)  n = (event_date=18348) - (intervention_date=18308)
= 40
1   3 (17:4)   Executing Stmt           : IF
1       (17:10)  _temp1 = (n=40) >= 0
1   4 (17:17)   Executing Stmt           : ASSIGN n =
1       (17:21)  n = (n=41) + 1 = 41
1   5 (18:7)   Executing Stmt           : RETURN _study_day_ =
1       (18:14)  _study_day_ = (n=41) = 41
1   6 (19:4)   Executing Stmt           : ENDSUB
--- Subroutine study_day Execution Finished.

1       (22:16)  sd = study_day( start=18308, today=18348 ) = 41
--- Program Execution Finished.
```

ABORT Statement

Terminates the current DATA step, job, or SAS session.

Syntax

ABORT;

Without Arguments

The ABORT statement in PROC FCMP has no arguments.

ARRAY Statement

Associates a name with a list of variables and constants.

Example: [“Example 8: Standardizing Each Row of a Data Set” on page 741](#)

Syntax

ARRAY *array-name*[*dimensions*] <*\$length* | *VARCHAR (length)* | *VARCHAR(*)*> < /
NOSYMBOLS > | <*variable(s)* > | <*constant(s)* > | <*initial-values* >;

Required Arguments

array-name

specifies the name of the array.

dimensions

is a numeric representation of the number of elements in a one-dimensional array or the number of elements in each dimension of a multidimensional array.

Optional Arguments

\$

specifies that the preceding variables are character variables of type CHAR.

length

specifies a numeric constant for storing variable values. For numeric and character variables, this constant is the maximum number of bytes stored in the variable.

VARCHAR

specifies that the preceding variables are of type VARCHAR.

length

specifies the maximum number of characters stored for VARCHAR variables.

*

specifies to support the maximum length allowed: 536,870,911 UTF-8 characters.

variable

specifies the variables of the array.

constant

specifies a number or a character string that indicates a fixed value. Enclose character constants in quotation marks.

initial-values

gives initial values for the corresponding elements in the array. You can specify internal values inside parentheses.

Options that Follow a Slash Delimiter

/NOSYMBOLS

specifies that an array of numeric or character values be created without the associated element variables. In this case, the only way that you can access elements in the array is by array subscripting.

Tips /NOSYMBOLS is used in exactly the same way as _TEMPORARY_.

You can save memory if you do not need to access the individual array element variables by name.

Details

ARRAY Statement Basics

The ARRAY statement in PROC FCMP is similar to the ARRAY statement that is used in the DATA step. The ARRAY statement associates a name with a list of variables and constants. You use the array name with subscripts to refer to items in the array.

The ARRAY statement that is used in PROC FCMP does not support all the features of the ARRAY statement in the DATA step. Here is a list of differences that apply only to PROC FCMP:

- All array references must have explicit subscript expressions.
- PROC FCMP uses parentheses after a name to represent a function call. When you reference an array, use square brackets [] or braces { }.
- The ARRAY statement in PROC FCMP does not support lower-bound specifications.
- You can use a maximum of six dimensions for an array.

- PROC FCMP supports VARCHAR arrays.

You can use both variables and constants as array elements in the ARRAY statement that is used in PROC FCMP. You cannot assign elements to a constant array. Although dimension specification and the list of elements are optional, you must provide one of these values. If you do not specify a list of elements for the array, or if you list fewer elements than the size of the array, PROC FCMP creates array variables by adding a numeric suffix to the elements of the array to complete the element list.

Passing Array References to PROC FCMP Routines

If you want to pass an array to a CALL routine and the CALL routine modifies the values of the array, you must specify the name for the array argument in an OUTARGS statement in the CALL routine.

Example

Here are some examples of the ARRAY statement:

```
array spot_rate[3] 1 2 3;
array spot_rate[3] (1 2 3);
array y[4] y1-y4;
array xx[2,3] x11 x12 x13 x21 x22 x23;
array pp p1-p12;
array q[1000] /nosymbols;
array foo[1] $ 30
array bar[5] varchar(10);
```

ATTRIB Statement

Specifies format, label, and length information for variables.

Syntax

ATTRIB *variable(s)* < FORMAT=*format-name* LABEL='*label*' LENGTH=*length* >;

Required Argument

variable

specifies the variables that you want to associate with attributes.

Optional Arguments

FORMAT=*format-name*

associates a format with variables in the *variable* argument.

LABEL=*label*

associates a label with variables in the *variable* argument.

LENGTH=*length*

specifies the length of the variable in the *variable* argument.

Example

Here are some examples of the ATTRIB statement:

```
attrib x1 format=date7. label='variable x1' length=5;
attrib x1 format=date7. label='variable x1' length=5
      x2 length=5
      x3 label='var x3' format=4.
      x4 length=$2 format=$4.;
```

DECLARE Statement

Declares an object.

Alias: DCL

Syntax

Form 1: **DECLARE** *object-type object-reference* ;

Form 2: **DECLARE OBJECT** *object-reference (object-type)*

Required Arguments

object-reference

specifies the reference name for the object.

object-type

specifies the type of object to declare.

Details

Details

Form 1 of the DECLARE statement applies to the Hash, Hiter, and Dictionary objects.

Form 2 of the DECLARE statement applies to the Python and ASTORE objects.

See Also

- “DECLARE Statement: Hash Object and Hash Iterator Object” in *SAS Component Objects: Reference*
- “DECLARE Statement: Dictionary Object” in *SAS Component Objects: Reference*
- “DECLARE Statement: Python Object” in *SAS Component Objects: Reference*
- “PROC FCMP and ASTORE”

DELETEFUNC Statement

Causes a function to be deleted from the function library that is specified in the OUTLIB option.

Syntax

DELETEFUNC *function-name*;

Required Argument

function-name

specifies the name of a function to be deleted from the function library that is specified in the OUTLIB option.

DELETESUBR Statement

Causes a subroutine to be deleted from the function library that is specified in the OUTLIB option.

Syntax

DELETESUBR *subroutine-name*;

Required Argument

subroutine-name

specifies the name of a subroutine to be deleted from the function library that is specified in the OUTLIB option.

FUNCTION Statement

Specifies a subroutine declaration for a routine that returns a value.

Examples: [“Example 1: Creating a Function and Calling the Function from a DATA Step” on page 731](#)
 [“Example 2: Creating and Saving Functions with PROC FCMP” on page 733](#)
 [“Example 3: Using Numeric Data in the FUNCTION Statement” on page 735](#)
 [“Example 4: Using Character Data with the FUNCTION Statement” on page 736](#)
 [“Example 5: Using Variable Arguments with an Array” on page 737](#)
 [“Example 7: Using Graph Template Language \(GTL\) with User-Defined Functions” on page 738](#)

Syntax

```
FUNCTION function-name(argument-1 < , argument-2, ... >) < VARARGS > < $ |  

VARCHAR > < length >  

< KIND | GROUP = 'string' > < LABEL = 'string-2' >;  

    ... more-program-statements ...  

    RETURN (expression );  

ENDSUB;
```

Required Arguments

function-name
 specifies the name of the function.

argument
 specifies one or more arguments for the function. You specify character arguments by placing a dollar sign (\$) after the argument name. *VARCHAR* arguments are specified by placing *VARCHAR* after the argument name. In the following example, `function myfunct(arg1, arg2 $, arg3, arg4 $);` `arg1` and `arg3` are numeric arguments, and `arg2` and `arg4` are character arguments.

expression
 specifies the value that is returned from the function.

Optional Arguments

VARARGS
 specifies that the function supports a variable number of arguments. If you specify **VARARGS**, then the last argument in the function must be an array.

Restriction You must specify a numeric variable with the **VARARGS** argument.

See [“Example 5: Using Variable Arguments with an Array” on page 737](#)

\$

specifies that the function returns a character value. If neither \$ or VARCHAR are specified, then the function returns a numeric value.

VARCHAR

specifies that the function returns a VARCHAR value. If neither VARCHAR or \$ are specified, then the function returns a numeric value.

length

specifies the length of a character or VARCHAR value. If no length is specified for a VARCHAR value, the maximum VARCHAR length allowed is used.

Defaults Character 32

VARCHAR(*)

KIND='string'

GROUP='string'

specifies a collection of items that have specific attributes and is limited to 32 characters.

LABEL='string-2'

specifies a label of up to 256 characters, including blanks.

Details

The FUNCTION statement is a special case of the subroutine declaration that returns a value. You do not use a CALL statement to call a function. The definition of a function begins with the FUNCTION statement and ends with an ENDSUB statement.

LABEL Statement

Specifies a label of up to 256 characters.

Syntax

LABEL *variable*=*'label'*;

Required Arguments

variable

names the variable that you want to label.

'label'

specifies a label of up to 256 characters, including blanks.

Example

Here are some examples of the LABEL statement:

```
label date='Maturity Date';
label bignum='Very very large numeric value';
```

LENGTH Statement

Specifies the number of bytes for storing character and numeric variables, or the number of characters for storing VARCHAR variables.

Syntax

LENGTH *variable-specification(s)* <DEFAULT= *n*>;

Required Argument

variable-specification

is a required argument and has this form:

variable(s) <\$ *length* | *length* | VARCHAR(*length*) | VARCHAR(*)>

variable

specifies one or more variables that are to be assigned a length

\$

specifies that the preceding variables are character variables of type CHAR.

length

specifies a numeric constant for storing variable values. For numeric and character variables, this constant is the maximum number of bytes stored in the variable.

Range

For character variables, 1 to 32767 bytes. For numeric variables, 3 to 8 bytes. The minimum length that you can specify for a numeric variable depends on the floating-point format used by your system. Because most systems use the IEEE floating-point format, the minimum is 3 bytes.

Restriction

In CAS, numeric variables shorter than 8 bytes are treated as 8 bytes.

VARCHAR

specifies that the preceding variables are of type VARCHAR.

length	
	specifies the maximum number of characters stored for VARCHAR variables.
	For example, <code>length foo varchar(100);</code> means store up to 100 characters in the <i>foo</i> VARCHAR variable.
Default	*
Range	1 to 536,870,911 UTF-8 characters for VARCHAR variables.
Restriction	When assigning a character constant to a VARCHAR variable, the character constant is limited to 32767 bytes.
*	specifies to support the maximum length allowed: 536,870,911 UTF-8 characters. For example, <code>length foo varchar(*);</code> means set the maximum length for the <i>foo</i> VARCHAR variable.

Optional Argument

DEFAULT = n	changes the default number of bytes that SAS uses to store the values of any newly created numeric variables.
Default	8 bytes
Range	3 to 8 bytes

LISTFUNC Statement

Causes the source code for a function to be written to the SAS listing.

Syntax

LISTFUNC *function-name* </NODEPENDENTS >;

Required Argument

<i>function-name</i>	specifies the name of the function for which source code is written to the SAS listing.
Note:	All dependent functions are returned in addition to the specified function by default.

Optional Argument

/NODEPENDENTS

specifies that only the functions specified by *function-name* be returned and not any dependent functions.

Alias `/NODEPS`

LISTSUBR Statement

Causes the source code for a subroutine to be written to the SAS listing.

Syntax

LISTSUBR *subroutine-name*;

Required Argument

subroutine-name

specifies the name of the subroutine for which source code is written to the SAS listing.

OUTARGS Statement

Specifies arguments in an argument list that you want a subroutine to update.

Restriction: Many SAS analytical procedures perform analytical differentiation on FCMP functions. If you plan to use the function in this way, do not use the OUTARGS statement. In most cases, use the OUTARGS statement only with the SUBROUTINE statement.

Examples: [“Example 2: Creating and Saving Functions with PROC FCMP” on page 733](#)
[“Example 8: Standardizing Each Row of a Data Set” on page 741](#)
[“Example 6: Using the SUBROUTINE Statement with a CALL Statement” on page 738](#)

Syntax

OUTARGS *out-argument-1* < , *out-argument-2*, ... >;

Required Argument

out-argument

specifies arguments from the argument list that you want the subroutine to update.

Tip If an array is listed in the OUTARGS statement within a routine, then the array is passed "by reference." Otherwise, it is passed "by value."

Example See ["SUBROUTINE Statement" on page 697](#) for an example of how to use the OUTARGS statement in a subroutine

STATIC Statement

Retains a variable's value from a previous call until the variable is reassigned.

Syntax

STATIC *variables* , <*initial-value(s)* >;

Required Argument

variables

specifies variable names, variable lists, or array names whose values you want to retain.

Optional Argument

initial-values

specifies an initial value, numeric or character, for one or more of the preceding elements.

Details

The STATIC statement can be used to initialize variables.

Local variables in a function or subroutine are usually not retained between calls to the function or subroutine. If there is an expensive initialization required, a STATIC variable can be used to perform the initialization. STATIC variables are not allocated on the stack, so they can be used for large local arrays to avoid reallocations and overflows on the stack.

Examples

Example 1: STATIC with a Numeric Variable

Here is a numeric static example:

```
proc fcmp;
  function fdef1(in);
    static x1 1;
    if x1 = 1 then do;
      x1 = 2;
    return(in);
    end;

    return (in*2);
  endsub;

  ans = fdef1( 1);
  put "Answer should be 1" ans=;
  ans = fdef1( 1);
  put "Answer should be 2" ans=;

run;
```

Example 2: STATIC with a Character Variable

Here is a character static example:

```
proc fcmp;
  function char_func( in $) $;
  length c1 $ 32;
  static c1 "Elephant";
  if c1 = "Elephant" then
    do;
      c1 = in || c1;
    return (c1);
    end;
  return( in);
endsub;
length ans $ 32;
ans = char_func( "Big ");
put "Answer should be >>Big Elephant<<" ans=;
ans = char_func( "Big ");
put "Answer should be >>Big<<" ans=;

run;
quit;
```

Example 3: STATIC with an Array

Here is an array static example:

```
proc fcmp ;
  function array_func( in ) ;
  array a[5] ;
  array foo[5];
  static a first 1;
```

```

    put a[1]= foo[1]=;
    If first then do;
        do i=1 to dim(a);
            a[i]=i;
            foo[i]=i;
        end;
        first =0;
    end;
else do;
    do i=1 to dim(a);
        a[i]=a[i]+1;
    end;
end;
put a[5];
return( in);

endsub;

ans = array_func( 4);

/* should increase by 1 */
ans = array_func( 4);

run;

```

STRUCT Statement

Declares (creates) structure types that are defined in C-Language packages.

Syntax

STRUCT *structure-name variable*;

Required Arguments

structure-name

specifies the name of a structure that is defined in a C-language package and declared in PROC FCMP.

variable

specifies the variable that you want to declare as this structure type.

Example

Here is an example of the STRUCT statement.

```

struct DATESTR matdate;
matdate.month=3;

```

```
matdate.day=22;
matdate.year=2009;
```

SUBROUTINE Statement

Declares (creates) an independent computational block of code that you can call using a CALL statement.

Examples:

[“Example 8: Standardizing Each Row of a Data Set” on page 741](#)

[“Example 2: Creating and Saving Functions with PROC FCMP” on page 733](#)

Syntax

```
SUBROUTINE subroutine-name (argument-1 < , argument-2 , ... >) < VARARGS >
< KIND | GROUP = 'string' >;
    OUTARGS out-argument-1 < , out-argument-2 , ... >;
    ... more-program-statements ...
ENDSUB ;
```

Required Arguments

subroutine-name

specifies the name of a subroutine.

argument

specifies one or more arguments for the subroutine. Character arguments are specified by placing a dollar sign (\$) after the argument name. VARCHAR arguments are specified by placing *VARCHAR* after the argument name. In the following example, `subroutine mysub(arg1, arg2 $, arg3, arg4 $);` `arg1` and `arg3` are numeric arguments, and `arg2` and `arg4` are character arguments.

OUTARGS

specifies arguments from the argument list that the subroutine should update.

out-argument

specifies arguments from the argument list that you want the subroutine to update.

Optional Arguments

VARARGS

specifies that the subroutine supports a variable number of arguments. If you specify **VARARGS**, then the last argument in the subroutine must be an array.

GROUP='string'

KIND='string'

specifies a collection of items that have specific attributes and is limited to 32 characters.

Details

The SUBROUTINE statement enables you to declare (create) an independent computational block of code that you can call with a CALL statement. The definition of a subroutine begins with the SUBROUTINE statement and ends with an ENDSUB statement. You can use the OUTARGS statement in a SUBROUTINE statement to specify arguments from the argument list that the subroutine should update.

Usage: FCMP Procedure

PROC FCMP and DATA Step Differences

Overview of PROC FCMP and DATA Step Differences

PROC FCMP was originally developed as a programming language for several SAS/STAT, SAS/ETS, and SAS/OR procedures. Because the implementation is not identical to the DATA step, differences exist between the two languages. The following section describes some of the differences between PROC FCMP and the DATA step.

Differences between PROC FCMP and the DATA Step

ABORT Statement

The ABORT statement in PROC FCMP does not accept arguments.

The ABORT statement is not valid within functions or subroutines in PROC FCMP. It is valid only in the main body of the procedure.

Arrays

PROC FCMP uses parentheses after a name to represent a function call. When referencing an array, the recommended practice is to use square brackets [] or

braces { }. For an array named ARR, the code would be `ARR[i]` or `ARR{i}`. PROC FCMP limits the number of dimensions for an array to six.

For more information about the differences in the ARRAY statement for PROC FCMP, see [“Details” on page 685](#).

Data Set Input and Output

PROC FCMP supports the DATA and OUTPUT statements for creating and writing to an output data set. The PROC FCMP statement supports the DATA= and OUT= options for specifying data input and output data sets. It does not support the SET, MERGE, UPDATE, or MODIFY statements for data set input. Data is typically transferred into and out of PROC FCMP routines by using parameters. If a large amount of data needs to be transferred, you can pass arrays to a PROC FCMP routine.

DATA Step Debugger

When you use the DATA step debugger, PROC FCMP routines perform like any other routine. That is, it is not possible to step into the function when debugging. Instead, use a PUT statement within the routine.

DO Statement

The following type of DO statement is supported by PROC FCMP:

```
do i=1, 2, 3;
```

The DO statement in PROC FCMP does not support character loop control variables. You can execute the following code in the DATA step, but not in PROC FCMP:

```
do i='a', 'b', 'c';
```

The DO statement does not support a character index variable. Therefore, the following code is not supported in PROC FCMP:

```
do i='one', 'two', 'three';
```

File Input and Output

PROC FCMP supports the PUT and FILE statements, but the FILE statement is limited to LOG and PRINT destinations. There are no INFILE or INPUT statements in PROC FCMP.

IF Expressions

An IF expression enables IF-THEN/ELSE conditions to be evaluated within an expression. IF expressions are supported by PROC FCMP but not by the DATA step. You can simplify some expressions with IF expressions by not having to split the expression among IF-THEN/ELSE statements. For example, the following two pieces of code are equivalent, but the IF expression (the first example) is not as complex:

```
x=if y < 100 then 1 else 0;
```

```

if y < 100 then
  x=1;
else
  x=0;

```

The alternative to IF expressions is expressions. This means that parentheses are used to group operations instead of DO/END blocks.

PUT Statement

The syntax of the PUT statement is similar in PROC FCMP and in the DATA step, but their operations can be different. In PROC FCMP, the PUT statement is typically used for program debugging. In the DATA step, the PUT statement is used as a report or file creation tool, as well as a debugging tool. The following list describes other differences:

- The PUT statement in PROC FCMP writes output to the SAS Output Window by default. The PUT statement in the DATA step writes output to the SAS log by default.
- The PUT statement in PROC FCMP does not support line pointers, format modifiers, column output, factored lists, iteration factors, overprinting, the `_INFILE_` option, or the special character `$`. It does not support features that are provided by the FILE statement options, such as `DLM=` and `DSD`.
- The PUT statement in PROC FCMP supports evaluating an expression and writing the result by placing the expression in parentheses. The DATA step, however, does not support the evaluation of expressions in a PUT statement. In the following example for PROC FCMP, the expressions `x/100` and `sqrt(y)/2` are evaluated and the results are written to the SAS log:

```
put (x/100) (sqrt(y)/2);
```

Because parentheses are used for expression evaluation in PROC FCMP, they cannot be used for variable or format lists as in the DATA step.

- The PUT statement in PROC FCMP does not support subscripted array names unless they are enclosed in parentheses. For example, the statement `put (A[i])`; writes the *i*-th element of the array *A*, but the statement `put A[i]`; results in an error message.
- An array name can be used in a PUT statement without subscripts. Therefore, the following statements are valid:
 - `put A=`; (when *A* is an array), writes all of the elements of array *A* with each value labeled with the name of the element variable.
 - `put (A) *=`; writes the same output as `put A=`;
 - `put A`; writes all of the elements of array *A*.
- The PUT statement in PROC FCMP follows the output of each item with a space, which is similar to list mode output in the DATA step. Detailed control over column and line position are supported to a lesser extent than in the DATA step.
- The PUT statement in PROC FCMP supports the print item `_PDV_` and prints a formatted listing of all of the variables in the routine's program data vector. The

statement `put _PDV_;` prints a much more readable listing of the variables than is printed by the statement `put _ALL_;`.

PUT Function

The syntax of the PUT function is similar in PROC FCMP and in the DATA step, but their results can be different. In PROC FCMP, the minimum of the string length and format width is used to determine the output formatted string width. In the DATA step, only the specified format width is used to determine the output width of the formatted string. In situations where the string length is less than the format length, PROC FCMP attempts to return the largest amount of valid information possible. In those same situations where the string length is less than the format length, the DATA step returns information in the format style. The following example demonstrates the differences between the PUT function in PROC FCMP and the DATA step:

Example Code 21.1 *Variable Length is Less Than Format Length*

```
proc fcmp;
/*Create variables with different lengths*/
length s2 $2;
length s3 $3;
length s4 $4;
length s5 $5;
length s6 $6;
length s7 $7;
length s8 $8;
length s9 $9;
length s10 $10;

/*Apply the same length=10 format to the different length variables*/
s2 = put('01Mar2022'd, mmddyy10.);
s3 = put('01Mar2022'd, mmddyy10.);
s4 = put('01Mar2022'd, mmddyy10.);
s5 = put('01Mar2022'd, mmddyy10.);
s6 = put('01Mar2022'd, mmddyy10.);
s7 = put('01Mar2022'd, mmddyy10.);
s8 = put('01Mar2022'd, mmddyy10.);
s9 = put('01Mar2022'd, mmddyy10.);
s10 = put('01Mar2022'd, mmddyy10.);

/*Display the results*/
put 's2=' s2;
put 's3=' s3;
put 's4=' s4;
put 's5=' s5;
put 's6=' s6;
put 's7=' s7;
put 's8=' s8;
put 's9=' s9;
put 's10=' s10;
run; quit;

/*Repeat the same steps for the DATA step*/
```

```

data _null_;
length s2 $2;
length s3 $3;
length s4 $4;
length s5 $5;
length s6 $6;
length s7 $7;
length s8 $8;
length s9 $9;
length s10 $10;

s2 = put('01Mar2022'd, mmddyy10.);
s3 = put('01Mar2022'd, mmddyy10.);
s4 = put('01Mar2022'd, mmddyy10.);
s5 = put('01Mar2022'd, mmddyy10.);
s6 = put('01Mar2022'd, mmddyy10.);
s7 = put('01Mar2022'd, mmddyy10.);
s8 = put('01Mar2022'd, mmddyy10.);
s9 = put('01Mar2022'd, mmddyy10.);
s10 = put('01Mar2022'd, mmddyy10.);

put 's2=' s2;
put 's3=' s3;
put 's4=' s4;
put 's5=' s5;
put 's6=' s6;
put 's7=' s7;
put 's8=' s8;
put 's9=' s9;
put 's10=' s10;
run;

```

Output 21.1 PROC FCMP Returns These Results:

```

s2= 03
s3= 03
s4= 0301
s5= 03/01
s6= 030122
s7= 030122
s8= 03/01/22
s9= 03/01/22
s10= 03/01/2022

```

Example Code 21.2 The DATA Step Returns These Results:

```

s2= 03
s3= 03/
s4= 03/0
s5= 03/01
s6= 03/01/
s7= 03/01/2
s8= 03/01/20
s9= 03/01/202
s10= 03/01/2022

```

The results show that when the string length is less than the format length, PROC FCMP prioritizes putting as much valid information in the result string as possible. The DATA step prioritizes the format style. For example, when the string length is set to 4 for the variable `s4` and the format length is 10, PROC FCMP returns the month and day `0301`, whereas the DATA step returns `03/0`. If the string length is equal to or greater than the format length, PROC FCMP and the DATA step return identical results.

WHEN and OTHERWISE Statements

The WHEN and OTHERWISE statements allow more than one target statement. That is, DO/END groups are not necessary for multiple WHEN statements. Here is an example:

```
SELECT;
    WHEN (expression-1
        )
        statement-1
    ;
    statement-2
    ;
    WHEN (expression-2
        )
        statement-3
    ;
    statement-4
    ;
END;
```

Operators

PROC FCMP operators are valid anywhere expressions are valid. For example, if-then expressions, do-loop expressions, assignment statements, and so on are all valid locations for operators.

IS <NOT> MISSING and IS <NOT> NULL are used to test for missing or null values. The IS operator is valid in PROC FCMP anywhere expressions are valid. Here are some examples:

```
if my-var IS MISSING then do;
    expression-1;
end;

do WHILE my-var IS NOT NULL;

    my-test = my-var IS MISSING;
```

PROC FCMP supports the CONTAINS operator. *target-string* <NOT> CONTAINS *test-string* tests if a specified test string is a part of a target string. The

CONTAINS operator is valid in PROC FCMP anywhere expressions are valid. Here is an example:

```
proc fcmp;
  length s $10;
  s = "abc";
  x = s contains "b";
  y = NOT(s contains "b");
  if s contains "b" then
    PUT "Contains Match";
  else
    PUT "Contains does not match";
  put x= y=;
run; quit;
```

The SAS log returns these results:

```
Contains Match
x=1 y=0
```

PROC FCMP supports the BETWEEN operator. When used, *value* <NOT> BETWEEN *range-1* AND *range-2* tests if a value is in the range of values specified. The BETWEEN operator is valid in PROC FCMP anywhere expressions are valid. Here is an example:

```
proc fcmp;
  x = 10;
  s= 'cat';
  r1 = x between 1 and 20;
  put r1=;
  r2 = x between 90 and 100;
  put r2=;
  r3= upcase(s) between 'CAMEL' and 'DOG';
  put r3=;
  if (x not between 80 and 70) then put 'out of range';
run; quit;
```

The SAS log returns these results:

```
r1=1
r2=0
r3=1
out of range
```

PROC FCMP supports the LIKE operator. The LIKE operator compares a test string to a target string using a pattern-matching specification. The LIKE operator is valid in PROC FCMP anywhere expressions are valid. For more details on pattern-matching, see [“LIKE Operator” in SAS SQL Procedure User's Guide](#). Here is an example:

```
proc fcmp;
  length s $ 100;
  s = 'hello there';
  r = s not like '%dog%';
  put r=;

  if s like 'hello%' then
    put "Match";
```

```

        else put "No Match";
    run;
quit;

```

The SAS log returns these results:

```

r=1
Match

```

PROC FCMP supports the SOUNDS-LIKE operator. The SOUNDS-LIKE operator is based on the SOUNDEX algorithm for identifying words that sound alike. The SOUNDEX algorithm is English-biased and is less useful for languages other than English. For more information about the SOUNDEX algorithm, see [Retrieving Values with the SOUNDS-LIKE Operator](#). The SOUNDS-LIKE operator is valid in PROC FCMP anywhere expressions are valid. The syntax for the SOUNDS-LIKE operator is shown in the following table and example:

Table 21.1 SOUNDS-LIKE Operator Syntax

Operator	Syntax
SOUNDS-LIKE	=*
NOT SOUNDS-LIKE	^=*
NOT SOUNDS-LIKE	~=*
NOT SOUNDS-LIKE	NE=*

```

proc fcmp;
length x $ 100 y $ 100;
x= Johnson;
y= Johnsen;
test = x =* y;
put test=;
run;
quit;

```

The SAS log returns this result:

```

test=1

```

Additional Features in PROC FCMP

PROC REPORT and Compute Blocks

PROC REPORT uses the DATA step to evaluate compute blocks. Because the DATA step can call PROC FCMP routines, you can also call these routines from PROC REPORT compute blocks.

Computing Implicit Values of a Function

PROC FCMP uses a SOLVE function for computing implicit values of a function.

PROC FCMP and Microsoft Excel

Many Microsoft Excel functions, not typically available in SAS, are implemented in PROC FCMP. You can find these functions in the sashelp.slkwxl data set. You can find more information about Excel functions in SAS by searching for them in [SAS Communities](#).

You can also view these functions by using the following SAS code:

```
proc fcmp inlib=sashelp.slkwxl listall;
run;
```

The following example uses the ODD_SLK function:

```
options cmplib=sashelp.slkwxl;
data _null_;
  num      =4.2;
  odd_num=odd_slk(num);
  put 'Odd number nearest to' num ' is ' odd_num;
run;
```

```
Odd number nearest to 4.2 is 5
```

Working with Arrays

Passing Arrays

By default, PROC FCMP passes arrays "by value" between routines. However, if an array is listed in the OUTARGS statement within the routine, the array is passed "by reference."

This means that a modification to the formal parameter by the function modifies the array that is passed. Passing arrays by reference helps to efficiently pass large amounts of data between the function and the calling environment because the data does not need to be copied. The syntax for specifying a formal array has the following form:

```
function
name(numeric-array-parameter[*],
character-array-parameter[*] $,
varchar-array-parameter[*] varchar);
```

You can pass DATA step temporary arrays to PROC FCMP routines.

Resizing Arrays

You can resize arrays in PROC FCMP routines by calling the built-in CALL routine DYNAMIC_ARRAY. The syntax for this CALL routine has the following form:

```
call dynamic_array(array, new-dim1-size <, new-dim2-size, ...>);
```

SAS passes to the DYNAMIC_ARRAY CALL routine both the array that is to be resized and a new size for each dimension of the array. A dynamic array enables the routine to allocate the amount of memory that is needed, instead of having to create an array that is large enough to handle all possible cases.

Support for dynamic arrays is limited to PROC FCMP routines. When an array is resized, the array is available only in the routine that resized it. It is not possible to resize a DATA step array or to return a PROC FCMP dynamic array to a DATA step.

Using Macros with PROC FCMP Routines

You can use the %SYSFUNC and %SYSCALL macros to call routines that you create with PROC FCMP. All SAS CALL routines are accessible with %SYSCALL except LABEL, VNAME, SYMPUT, and EXECUTE. %SYSFUNC and %SYSCALL macros support SAS function names up to 32 characters.

Variable Scope in PROC FCMP Routines

The Concept of Variable Scope

A critical part of keeping routines and programs independent of one another is variable scope. A variable's scope is the section of code where a variable's value can be used. In the case of PROC FCMP routines, variables that are declared outside a routine are not accessible inside a routine. Variables that are declared

inside a routine is not accessible outside the routine. Variables that are created within a routine are called local variables because their scope is “local” to the routine.

Functions use local variables as scratch variables during computations, and the variables are not available when the function returns. When a function is called, space for local variables is pushed on the call stack. When the function returns, the space used by local variables is removed from the call stack.

When Local Variables in Different Routines Have the Same Name

The concept of variable scope can be confusing when local variables in different routines have the same name. When this occurs, each local variable is distinct. In the following example, the DATA step and CALL routines subA and subB contain a local variable named x. Each x is distinct from the other x variables. When the program executes, the DATA step calls subA and subA calls subB. Each environment writes the value of x to the log. The log output shows how each x is distinct from the others.

```
proc fcmp outlib=work.funcs.math;
  subroutine subA();
    x=5;
    call subB();
    put 'In subA: ' x;
  endsub;

  subroutine subB();
    x='subB';
    put 'In subB: ' x;
  endsub;
run;

options cmplib=work.funcs;
data _null_;
  x=99;
  call subA();
  put 'In DATA step: ' x;
run;
```

Example Code 21.3 Local Variables in Different Routines That Have the Same Name

```
In subB:  x=subB
In subA:  x=5
In DATA step: x=99
```

Recursion

PROC FCMP routines can be recursive. Recursion is a problem-solving technique that reduces a problem to a smaller one that is simpler to solve and then combines the results of the simpler solution to form a complete solution. A recursive function is a function that calls itself, either directly or indirectly.

Each time a routine is called, space for the local variables is pushed on the call stack. The space on the call stack ensures independence of local variables for each call. When the routine returns, the space allocated on the call stack is removed, freeing the space used by local variables. Recursion relies on the call stack to store progress toward a complete solution.

When a routine calls itself, both the calling routine and the routine that is being called must have their own set of local variables for intermediate results. If the calling routine was able to modify the local variables of the routine that is being called, it would be difficult to program a recursive solution. A call stack ensures the independence of local variables for each call.

In the following example, the ALLPERMK routine in PROC FCMP has two arguments, n and k , and writes all $C(n, k) = n! / (n - k)!$ combinations that contain exactly k out of the n elements. The elements are represented as binary values (0, 1). The function ALLPERMK calls the recursive function PERMK to traverse the entire solution space and output only the items that match a particular filter:

```
proc fcmp outlib=work.funcs.math;
  subroutine allpermk(n, k);
    array scratch[1] / nosymbols;
    call dynamic_array(scratch, n);
    call permk(n, k, scratch, 1,0);
  endsub;

  subroutine permk(n, k, scratch[*], m, i);
    outargs scratch;
    if m-1=n then do;
      if i=k then
        put scratch[*];
      end;
    else do;
      scratch[m]=1;
      call permk(n, k, scratch, m+1, i+1);
      scratch[m]=0;
      call permk(n, k, scratch, m+1, i);
    end;
  endsub;
run;
quit;

options cmplib=work.funcs;
data _null_;
  call allpermk(5,3);
```

```
run;
```

Example Code 21.4 *Recursion Example Results*

```
1 1 1 0 0
1 1 0 1 0
1 1 0 0 1
1 0 1 1 0
1 0 1 0 1
1 0 0 1 1
0 1 1 1 0
0 1 1 0 1
0 1 0 1 1
0 0 1 1 1
```

This program uses the /NOSYMBOLS option in the ARRAY statement to create an array without a variable for each array element. A /NOSYMBOLS array can be accessed only with an array reference, `scratch[m ]`, and is equivalent to a DATA step `_temporary_` array. A /NOSYMBOLS array uses less memory than a regular array because no space is allocated for variables. ALLPERMK also uses PROC FCMP dynamic arrays.

Directory Traversal

Overview of Directory Traversal

Implementing functionality that enables functions to traverse a directory hierarchy is difficult if you use the DATA step or macros. With the DATA step and macro code recursion or pseudo-recursion is not easy to code. This section describes how to develop a routine named DIR_ENTRIES that fills an array with the full pathname of all of the files in a directory hierarchy. This example shows the similarity between PROC FCMP and DATA step syntax and underscores how PROC FCMP routines simplify a program and produce independent, reusable code. DIR_ENTRIES uses as input the following parameters:

- a starting directory
- a result array to fill with pathnames
- an output parameter that is the number of pathnames placed in the result array
- an output parameter that indicates whether the complete result set was truncated because the result array was not large enough

The flow of control for DIR_ENTRIES is as follows:

- 1 Open the starting directory.
- 2 For each entry in the directory, do one of the following tasks:
 - If the entry is a directory, call DIR_ENTRIES to fill the result array with the subdirectory's pathnames.

- Otherwise, the entry is a file, and you must add the file's path to the result array.
- 3 Close the starting directory.

Directory Traversal Example

Opening and Closing a Directory

Opening and closing a directory are handled by the CALL routines DIROPEN and DIRCLOSE. DIROPEN accepts a directory path and has the following flow of control:

- 1 Create a fileref for the path by using the FILENAME function.
- 2 If the FILENAME function fails, write an error message to the log and then return.
- 3 Otherwise, use the DOPEN function to open the directory and retrieve a directory ID.
- 4 Clear the directory fileref.
- 5 Return the directory ID.

The DIRCLOSE CALL routine is passed a directory ID, which is passed to DCLOSE. DIRCLOSE sets the passed directory ID to missing so that an error occurs if a program tries to use the directory ID after the directory has been closed. The following code implements the DIROPEN and DIRCLOSE CALL routines:

```
proc fcmp outlib=work.funcs.dir;
  function diropen(dir $);
    length dir $ 256 fref $ 8;
    rc=filename(fref, dir);
    if rc=0 then do;
      did=dopen(fref);
      rc=filename(fref);
    end;
    else do;
      msg=sysmsg();
      put msg '(DIROPEN(' dir= ')';
      did=.;
    end;
    return(did);
  endsub;

  subroutine dirclose(did);
    outargs did;
    rc=dclose(did);
    did=.;
  endsub;
```

Gathering Filenames

File paths are collected by the DIR_ENTRIES CALL routine. DIR_ENTRIES uses the following arguments:

- a starting directory
- a result array to fill
- an output parameter to fill with the number of entries in the result array
- an output parameter to set to 0 if all pathnames fit in the result array; or an output parameter to set to 1 if some of the pathnames do not fit into the array

The body of DIR_ENTRIES is almost identical to the code that is used to implement this functionality in a DATA step. Also, DIR_ENTRIES is a CALL routine that is easily reused in several programs.

DIR_ENTRIES calls DIROPEN to open a directory and retrieve a directory ID. The routine then calls DNUM to retrieve the number of entries in the directory. For each entry in the directory, DREAD is called to retrieve the name of the entry. Now that the entry name is available, the routine calls MOPEN to determine whether the entry is a file or a directory.

If the entry is a file, then MOPEN returns a positive value. In this case, the full path to the file is added to the result array. If the result array is full, the truncation output argument is set to 1.

If the entry is a directory, then MOPEN returns a value that is less than or equal to 0. In this case, DIR_ENTRIES gathers the pathnames for the entries in this subdirectory. It gathers the pathnames by recursively calling DIR_ENTRIES and passing the subdirectory's path as the starting path. When DIR_ENTRIES returns, the result array contains the paths of the subdirectory's entries.

```
subroutine dir_entries(dir $, files[*] $, n, trunc);
  outargs files, n, trunc;
  length dir entry $ 256;

  if trunc then return;

  did=diropen(dir);
  if did <= 0 then return;

  dnum=dnum(did);
  do i=1 to dnum;
    entry=dread(did, i);
    /* If this entry is a file, then add to array, */
    /* else entry is a directory, recurse.          */
    fid=mopen(did, entry);
    entry=trim(dir) || '\ ' || entry;
    if fid > 0 then do;
      rc=fclose(fid);
      if n < dim(files) then do;
        trunc=0;
        n=n + 1;
        files[n]=entry;
      end;
    end;
```

```

        else do;
            trunc=1;
            call dirclose(did);
            return;
        end;
    end;
else
    call dir_entries(entry, files, n, trunc);
end;

    call dirclose(did);
    return;
endsub;

```

Calling DIR_ENTRIES from a DATA Step

You invoke DIR_ENTRIES like any other DATA step CALL routine. Declare an array with enough entries to hold all the files that might be found. Then call the DIR_ENTRIES routine. When the routine returns, the result array is looped over and each entry in the array is written to the SAS log.

```

options cmplib=work.funcs;
data _null_;
    array files[1000] $ 256 _temporary_;
    dnum=0;
    trunc=0;
    call dir_entries("c:\logs", files, dnum, trunc);
    if trunc then put 'ERROR: Not enough result array entries. Increase
array
size.';
    do i=1 to dnum;
        put files[i];
    end;
run;

```

Example Code 21.5 Results from Calling DIR_ENTRIES from a DATA Step

```

c:\logs\2004\qtr1.log
c:\logs\2004\qtr2.log
c:\logs\2004\qtr3.log
c:\logs\2004\qtr4.log
c:\logs\2005\qtr1.log
c:\logs\2005\qtr2.log
c:\logs\2005\qtr3.log
c:\logs\2005\qtr4.log
c:\logs\2006\qtr1.log
c:\logs\2006\qtr2.log

```

This example shows the similarity between PROC FCMP syntax and the DATA step. For example, numeric expressions and flow of control statements are identical. The abstraction of DIROPEN into a PROC FCMP function simplifies DIR_ENTRIES. All of the PROC FCMP routines that are created can be reused by other DATA steps without any need to modify the routines to work in a new context.

Identifying the Location of Compiled Functions and Subroutines: The CMPLIB= System Option

Overview of the CMPLIB= System Option

The SAS system option CMPLIB= specifies where to look for previously compiled functions and subroutines. All procedures (including FCMP) that support the use of FCMP functions and subroutines use this system option.

Instead of specifying the LIBRARY= option on every procedure statement that supports functions and subroutines, you can use the CMPLIB= system option to set libraries that can be used by all procedures.

The _DISPLAYLOC_ option writes to the log the name of the data set from where SAS loaded a function. The _NO_DISPLAYLOC_ option prevents the data set name from being written to the log.

Syntax of the CMPLIB= System Option

The syntax for the CMPLIB= option has the following form:

```

OPTIONS CMPLIB=library
OPTIONS CMPLIB=(library-1 <, library-2, ...>)
OPTIONS CMPLIB=list-1 <, list-2, ...
OPTIONS CMPLIB=_DISPLAYLOC_
OPTIONS CMPLIB=_NO_DISPLAYLOC_

```

The following descriptions refer to the preceding syntax:

OPTIONS

identifies the statement as an OPTIONS statement.

library

specifies that the previously compiled libraries be linked into the program.

list

specifies a list of libraries.

DISPLAYLOC

when using PROC FCMP, specifies to display in the SAS log the data set from where SAS loaded the function.

Default _NO_DISPLAYLOC_

_NO_DISPLAYLOC_

when using PROC FCMP, specifies to not display in the SAS log the data set from where SAS loaded the function, and removes any library specifications as CMPLIB= option values.

Default _NO_DISPLAYLOC_

Tip When you specify CMPLIB=library-specification without the _DISPLAYLOC_ option, SAS does not display the data set name in the SAS log.

Example 1: Setting the CMPLIB= System Option

The following example shows how to set the CMPLIB= system option.

```
options cmplib=work.funcs;
options cmplib=(work.funcs work.functions mycat.funcs);
options cmplib=(work.func1 - work.func10);
```

Example 2: Compiling and Using Functions

In the following example, PROC FCMP compiles the SIMPLE function and stores it in the work.Models data set. Then the CMPLIB= system option is set, and the function is called by PROC MODEL.

The output from this example spans several pages. The output is divided into five parts.

```
proc fcmp outlib=work.models.yval;
  function simple(a, b, x);
    y=a+b*x;
    return(y);
  endsub;
run;

options cmplib=work.models nodate ls=80;

data a;
  input y @@;
  x=_n_;
  datalines;
08 06 08 10 08 10
;

proc model data=a;
  y=simple(a, b, x);
  fit y / outest=est1 out=out1;
quit;
```

Output 21.2 Compiling and Using Functions: Part 1**The SAS System****The MODEL Procedure**

Model Summary	
Model Variables	1
Parameters	2
Equations	1
Number of Statements	1

Model Variables	y
Parameters	a b
Equations	y

The Equation to Estimate is	
y =	F(a, b)

NOTE: At OLS Iteration 1 CONVERGE=0.001 Criteria Met.

Output 21.3 *Compiling and Using Functions: Part 2***The SAS System****The MODEL Procedure
OLS Estimation Summary**

Data Set Options	
DATA=	A
OUT=	OUT1
OUTEST=	EST1

Minimization Summary	
Parameters Estimated	2
Method	Gauss
Iterations	1

Final Convergence Criteria	
R	0
PPC	0
RPC(a)	64685.48
Object	0.984333
Trace(S)	1.67619
Objective Value	1.11746

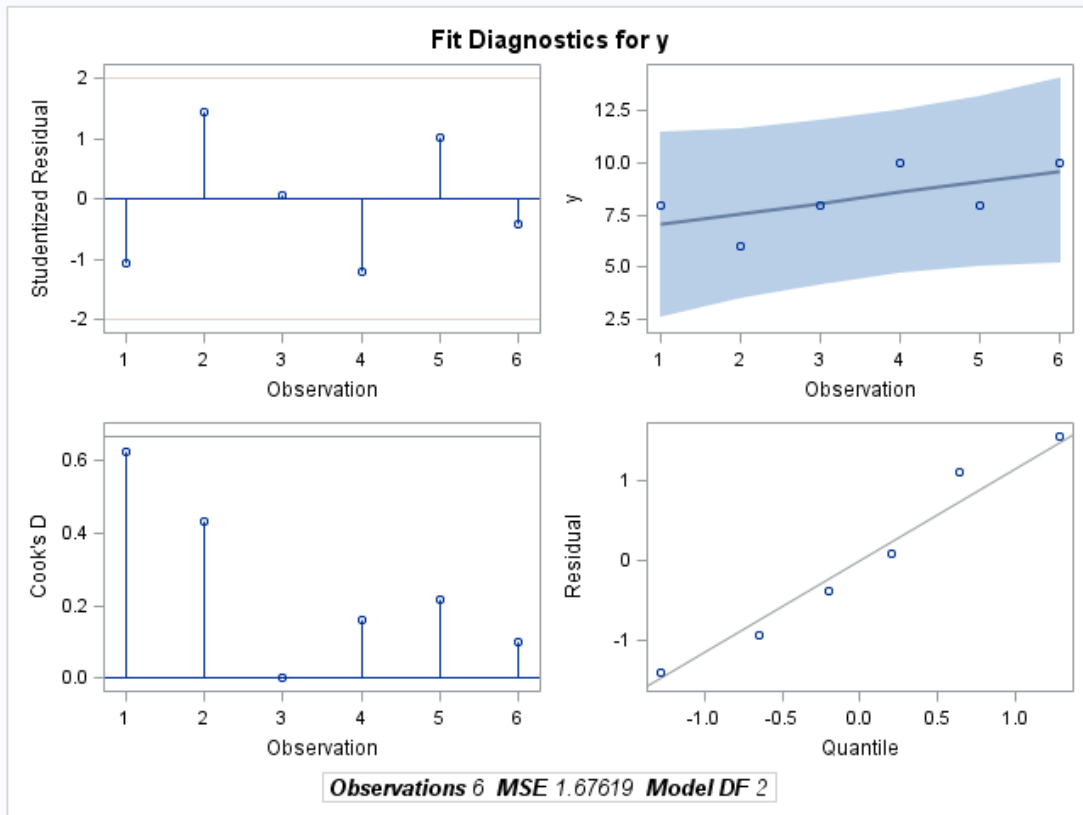
Observations Processed	
Read	6
Solved	6

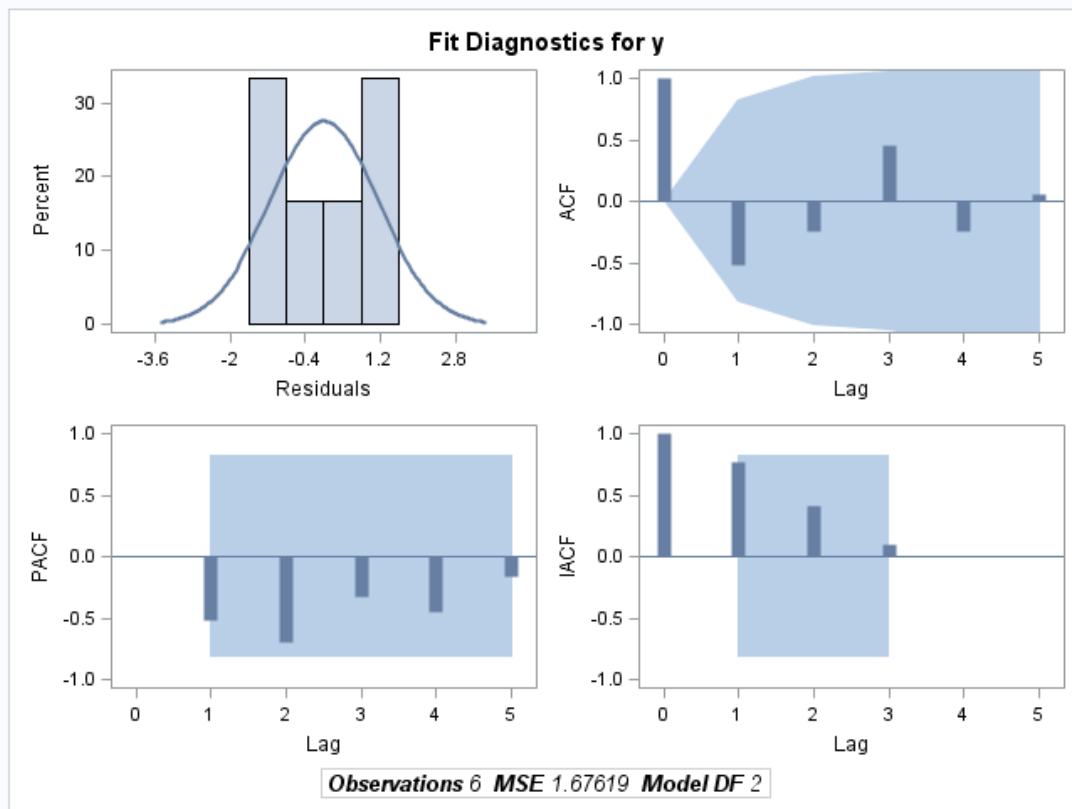
Output 21.4 Compiling and Using Functions: Part 3**The SAS System****The MODEL Procedure**

Nonlinear OLS Summary of Residual Errors							
Equation	DF Model	DF Error	SSE	MSE	Root MSE	R-Square	Adj R-Sq
y	2	4	6.7048	1.6762	1.2947	0.4084	0.2605

Nonlinear OLS Parameter Estimates				
Parameter	Estimate	Approx Std Err	t Value	Approx Pr > t
a	6.533333	1.2053	5.42	0.0056
b	0.514286	0.3095	1.66	0.1719

Number of Observations		Statistics for System	
Used	6	Objective	1.1175
Missing	0	Objective*N	6.7048

Output 21.5 Compiling and Using Functions: Part 4**The SAS System****The MODEL Procedure**

Output 21.6 Compiling and Using Functions: Part 5

For information about PROC MODEL, see the *SAS/ETS User's Guide*.

Example 3: Identifying the Data Set Name from Where SAS Loaded a Function

The following example uses the `_DISPLAYLOC_` and `_NO_DISPLAYLOC_` options. When you use the `_DISPLAYLOC_` option, SAS writes to the log the name of the data set from where SAS loaded a function. With the `_NO_DISPLAYLOC_` option, the name of the data set is not written to the log.

```
proc fcmp outlib=work.myfuncs1.pkg;
  function myfunc();
    return(1);
  endsub;
run;

proc fcmp outlib=work.myfuncs2.pkg;
  function myfunc();
    return(2);
  endsub;
run;

proc fcmp outlib=work.myfuncs3.pkg;
  function myfunc();
```

```

        return(3);
    endsub;
run;

option CMPLIB=(myfuncs1-myfuncs3 _DISPLAYLOC_);

proc fcmp;

    a = myfunc();
    put a=;
run;

/*- turning _DISPLAYLOC_ off -*/
option CMPLIB=(myfuncs1-myfuncs3);

proc fcmp;

    a = myfunc();
    put a=;
run;

option CMPLIB=(myfuncs1 myfuncs2 _DISPLAYLOC_);

proc fcmp;

    a = myfunc();
    put a=;
run;

option CMPLIB=_DISPLAYLOC_;

proc fcmp inlib=work.myfuncs1;
    a = myfunc();
    put a=;
run;

option CMPLIB=_NO_DISPLAYLOC_;

proc fcmp inlib=work.myfuncs1;
    a = myfunc();
    put a=;
run;

```

The following results show a partial SAS log. The `_DISPLAYLOC_` and `_NO_DISPLAYLOC` options produce different results:

```

116 option CMPLIB=(myfuncs1-myfuncs3 _DISPLAYLOC_);
117
118 proc fcmp;
119
120     a = myfunc();
121     put a=;
122 run;

NOTE: Function 'myfunc' loaded from WORK.myfuncs3.PKG.
NOTE: PROCEDURE FCMP used (Total process time):
      real time          0.06 seconds
      cpu time           0.04 seconds

123
124 /*- turning _DISPLAYLOC_ off -*/
125 option CMPLIB=(myfuncs1-myfuncs3);
126
127
128 proc fcmp;
129
130     a = myfunc();
131     put a=;
132 run;

NOTE: PROCEDURE FCMP used (Total process time):
      real time          0.06 seconds
      cpu time           0.06 seconds

133
134 option CMPLIB=(myfuncs1 myfuncs2 _DISPLAYLOC_);
135
136 proc fcmp;
137
138     a = myfunc();
139     put a=;
140 run;

NOTE: Function 'myfunc' loaded from WORK.myfuncs2.PKG.
NOTE: PROCEDURE FCMP used (Total process time):
      real time          0.05 seconds
      cpu time           0.04 seconds

141
142 option CMPLIB=_DISPLAYLOC_;
143
144 proc fcmp inlib=work.myfuncs1;
145     a = myfunc();
146     put a=;
147 run;

NOTE: Function 'myfunc' loaded from work.myfuncs1.PKG.
NOTE: PROCEDURE FCMP used (Total process time):
      real time          0.04 seconds
      cpu time           0.03 seconds

```



```

149 option CMPLIB=_NO_DISPLAYLOC_;
150
151 proc fcmp inlib=work.myfuncs1;
152     a = myfunc();
153     put a=;
154 run;

```

```

NOTE: PROCEDURE FCMP used (Total process time):
      real time          0.04 seconds
      cpu time           0.03 seconds

```

Using PROC FCMP Component Objects

Hash Object and Hash Iterator Object

Hashing is available in user-defined subroutines through the FCMP procedure. Hashing enables you to extend the scope of your programs so that you can solve larger problems without sacrificing simplicity. Embedding a hash object in PROC FCMP functions and subroutines can improve performance and streamline existing programs.

For more information about hash and hash iterator component objects, see [“Using the PROC FCMP Hash Object and PROC FCMP Hash Iterator Object” in SAS Component Objects: Reference](#).

Dictionaries

Dictionaries create references to numeric and character data, and they also give you fast in-memory hashing to arrays, other dictionaries, and PROC FCMP hash objects.

For information about how to use dictionaries in PROC FCMP, and to review examples, see [“Using FCMP Dictionary Objects” in SAS Component Objects: Reference](#) and [Dictionaries: Referencing a New PROC FCMP Data Type](#).

PROC FCMP and ASTORE

CMP supports ASTORE (Analytic Store) scoring models. Data movement is reduced by performing the score and the computations together within the same TKCMP program. PROC FCMP supports ASTORE models on the SAS client.

What is an Analytic Store?

An analytic store is a binary file that contains the state from a predictive analytic procedure. This state from a predictive analytic procedure, such as a random forest, is created using the results from the training phase of model development. A key feature of an analytic store is that it is easily transported from one host to another. An analytic store is a compact and universal file form. The store names the only SAS component that can restore the state of the computation (memory) that can restore the post training memory. It is also called a warm restart for scoring.

What is a State?

State is a copy of all the memory items that are relevant to the next task, for example, scoring. `SAVESTATE` takes a snapshot of this information:

- Public information (that is, information common to all analytic engines). Examples of public information include the list of input variables, the list of output variables, the formats, and other elements.
- Private information (that is, information specific to that particular analysis, for example, random forests). Examples of private information include the number of trees, the trees themselves, the scores, and other types.

What is Scoring?

Historic data is where the outcomes are known. You train your model with this historic data, and then you score the new data by using the input variables. Scoring new data is predicting the outcome with new data, using the model that is built using the historic data.

What Does PROC ASTORE Do?

PROC ASTORE scores an input data set and produces an output data set using the analytic store that you specify. It is an interactive procedure in which each statement runs immediately. PROC ASTORE produces these outputs:

- Types of DS2 scoring code that can run locally using the DS2 procedure
- DS2 scoring code that can run in SAS Viya.

PROC ASTORE can also move analytic stores between the client and the server and it can provide descriptive information about the analytic store. The syntax is shown below:

SCORE

Scores the model.

DESCRIBE

Specifies the name of the analytic store and produces DS2 basic scoring code.

DOWNLOAD

Retrieves from the CAS session the specified analytic store and stores it in the local file system

UPLOAD

Moves the specified analytic store from the local file system into a data table in CAS.

Note: See [The ASTORE Procedure](#) for more information about PROC ASTORE.

Example: Using ASTORE in PROC FCMP

This example demonstrates using ASTORE in PROC FCMP. It also demonstrates TKCMP scoring of ASTORE models.

Note: The code for declaring a CMP object is the same for all platforms:

```
declare object myscore(astore);
```

Note: In a SAS client, the score() method takes a single parameter file path:

■ Windows:

```
call myscore.score("C:\models\_va_model208");
```

■ UNIX:

```
call myscore.score("/userid/models/_va_model208");
```

Note: In CAS, the score() method requires two arguments, the caslib and the CAS table name:

```
call myscore.score('CASUSER', '_va_model208');
```

Note: The score object also supports the describe() method, which prints the input/output variables from the ASTORE to the log. The method takes no arguments:

```
call myscore.describe();
```

This code runs a previously created SVM CAS action in FCMP.

Example Code 21.2 *ASTORE.SAS*

```

title 'CMP ASTORE Test 1';

libname mydata "C:\project\my_data";
libname mymodel "C:\project\my_models";

/* Test a simple object model using the describe method */
/* Note: No input dataset is specified. */
proc fcmp;
  declare object myscore(astore);
  call myscore.score("C:\project\my_models\_va_model208");

  /* Can use Proc FCMP to describe the local model */
  call myscore.describe();
run;quit;

/* Run the ASTORE model over an input dataset. */
/* Write the results to an output dataset. */
proc fcmp data=mydata.hmeq out=astore_fcmp_out;
  declare object myscore(astore);
  call myscore.score("C:\project\my_models\_va_model208");
run;quit;

proc print data=astore_fcmp_out(obs=10);
run;

title 'CMP ASTORE Test 2';

/* Using ASTORE scoring from a FCMP function */
proc fcmp outlib=work.score.funcs;

/* Return the probability value */
function astore(clage, clno, debtinc, delinq, ninq, value);
  declare object myscore(astore);
  call myscore.score("C:\project\my_models\_va_model208");
  return(_P_);
endsub;

/* Pass input values as arguments into the function */
/* Return all probability values from the score */
subroutine astore2(clage, clno, debtinc, delinq, ninq, value,
                  _P_, P__EVENT_0, P__EVENT_1, I__EVENT_ $, _WARN_ $);
  outargs _P_, P__EVENT_0, P__EVENT_1, I__EVENT_, _WARN_;

  declare object myscore(astore);
  call myscore.score("C:\project\my_models\_va_model208");
endsub;
run;
quit;

/* Call these functions from Proc FCMP */

title 'CMP ASTORE Test 3';

proc fcmp data=mydata.hmeq inlib=work.score out=astore_fcmp_out2;

```

```

ds_p2 = astore(clage, clno, debtinc, delinq, ninq, value);
run;
quit;
proc print data=astore_fcmp_out2(obs=10);
run;

title 'CMP ASTORE Test 4';

proc fcmp data=mydata.hmeq(obs=10) inlib=work.score
out=astore_fcmp_out3;
  _P=.;
  P__EVENT_0=.;
  P__EVENT_1=.;
  I__EVENT_="";
  _WARN_="";

call astore2(clage, clno, debtinc, delinq, ninq, value,
             _P_, P__EVENT_0, P__EVENT_1, I__EVENT_, _WARN_);
run;
quit;
proc print data=astore_fcmp_out3(obs=10);
run;

title 'CMP ASTORE Test 5';

/* You can call the function from the data step */
/* We can perform ASTORE scoring using the data step */
options cmplib=work.score;
data astore_ds_out;
set mydata.hmeq;
ds_p = astore(clage, clno, debtinc, delinq, ninq, value);
run;
proc print data=astore_ds_out(obs=10);
run;

```

PROC ASTORE transports data from CAS into a local analytic store (local file).
Once in that form, the data is used by CMP and the new ASTORE object.

Example Code 21.3 ASTORE_CAS.SAS

```

title 'CMP ASTORE Test 1';

/* CASPORT=0 means generate a port number for session connection */
/*options casuser=<userid> cashost="<machine>" casport=<port>;*/
options casuser=<userid> cashost="<machine>" casport=<port>;

/* This is how to start a CAS session */
/* CAS sessions live for the life of the SAS session, unless redefined or closed. */
/* Unix: cas mysession AUTHINFO='/userid/.authinfo';*/
/* Windows */
cas mysession authinfo='/userid/.authinfo';
libname sascas1 cas sessref="mysession" caslib="CASUSER";

libname mydata "C:\project\my_data";
libname mymodel "C:\project\my_models";

```

```

/* Our input data for SVM; Load into CAS */
data sascas1.hmeq;
set mydata.hmeq;
run;

/* Run action from VDMML */
proc cas;
    action builtins.loadactionset / actionSet='tkaasvm';
    action tkaasvm.svmtrain result=r /
c=1.0,
code={comment=false,fmtWdth=15,lineSize=200},
includeMissing=false,
maxiter=25,
noscale=false,
savestate={caslib="CASUSER",name="_va_model208"},
table={caslib="CASUSER",compOnDemand="false",compPgm="_va_calculated_208_1=round('BAD'n,1
.0)};
if (('va_calculated_208_1'n = 0.0))then do;
    _va_calculated_208_11= 0.0;
end;
else do;
    _va_calculated_208_11= 1.0;
end;
;
_EVENT=_va_calculated_208_11;
_va_FILTER=(NOT(MISSING('va_calculated_208_1'n))
AND NOT(MISSING('CLAGE'n)) AND NOT(MISSING('CLNO'n)) AND NOT(MISSING('DEBTINC'n))
AND NOT(MISSING('DELINQ'n)) AND NOT(MISSING('NINQ'n)) AND NOT(MISSING('VALUE'n)));
_va_calculated_208_14=NOT(('va_FILTER'n = 0.0));";compVars={"_va_calculated_208_1","
_va_calculated_208_11","_EVENT","_va_FILTER","_va_calculated_208_14"},name="HMEQ",
onDemand="false",where="NOT('_va_calculated_208_14'n = 0)"}",
emtarget={name="_EVENT_",options={levelType="BINARY"}},
tolerance=1.0E-6,
var={"CLAGE","CLNO","DEBTINC","DELINQ","NINQ","VALUE"}
outputTables={names={nobs="NObs",modelinfo="ModelInfo"}, replace="TRUE"};
run;

quit;

/* Use Proc ASTORE to describe the model in CAS */
/* Download the model locally */
proc astore;
    describe rstore=sascas1._va_model208;
    download rstore=sascas1._va_model208;
    store="C:\project\my_models\_va_model208";
    describe store="C:\project\my_models\_va_model208";
run;
quit;

options pagesize=max;
proc cas;
    loadactionset "table";
    table.fetch
    format=false
    maxRows=1
    sasTypes=TRUE

```

```

table = {
  compOnDemand=TRUE
  caslib="CASUSER"
  name="hmeq"
  compPgm=
"declare object myscore( astore );
call myscore.score('CASUSER', '_va_model208');"
  singlePass=TRUE
  compVars={"_P_", "P__EVENT_0" , "P__EVENT_1" , "I__EVENT_" , "_WARN_"}
};
run;
quit;

```

The SETOPTION Method

The CMP ASTORE method, SETOPTION, passes options to ASTORE objects that take different option values. Here is code using PROC FCMP:

```

routineCode = "
  declare object model_glmstore( astore );
  call model_glmstore.setoption('alpha', 0.05);
  call model_glmstore.setoption('COMPUTE_CONFIDENCE_LIMIT', 1);
  call model_glmstore.score('CASUSER', 'glmstore');
"
;
run;
quit;

```

Here is code using the FCMPACT action:

```

routineCode = "
declare object model_glmstore( astore );
call model_glmstore.setoption('alpha', 0.05);
call model_glmstore.setoption('COMPUTE_CONFIDENCE_LIMIT', 1);
call model_glmstore.score('CASUSER', 'glmstore');
"
;
run;
quit;

```

Using Python Functions in PROC FCMP

PROC FCMP supports submitting and executing functions written in Python by using the Python object. A complete guide for using the Python object is located in the [Using Python Objects](#) chapter in the [SAS Component Objects: Reference](#).

Returning a Variable by Name Using VNAME Arguments

PROC FCMP supports the use of VNAME arguments in PROC FCMP functions and subroutines. By default, when a PROC FCMP function or subroutine is called, the variables are passed by value. PROC FCMP VNAME arguments enable users to pass variables by name to a function or subroutine like the [VNAME Function](#). To create a PROC FCMP VNAME argument, specify `vname` after the argument name.

IMPORTANT PROC FCMP VNAME arguments cannot be used in the OUTARGS list.

IMPORTANT PROC FCMP VNAME arguments must be variable names. Specifying an expression or an array subscript is not supported for PROC FCMP VNAME arguments.

In the following example, a PROC FCMP function, FOO, and subroutine, FOO_SUB, are defined using PROC FCMP VNAME arguments. An additional function, FOO_VNAME, is defined to compare using the VNAME function instead of VNAME arguments. The input variable is `callarg = "hello"`. When the functions and subroutine are called, the name of the variable, `callarg` is returned instead of the value, `"hello"`.

Example Code 21.4 Using PROC FCMP VNAME Arguments

```
proc fcmp;

  function foo(defarg vname) $;
  length s $40;
  s = "" || trim(defarg) || "";
  return(s);
endfunc;

  function foo_vname(defarg $) $;
  length s $40;
  s = "" || trim(defarg) || "";
  return(s);
endfunc;

  subroutine foo_sub(retarg $, defarg vname);
  outargs retarg;
  length s $40;
  s = "" || trim(defarg) || "";
  returnarg = s;
endsub;

length x $20;
```



```

length y $20;
length z $20;
length callarg $20;
callarg = "hello";

x = foo(callarg);
y = foo_vname(vname(callarg));
call foo_sub(z, callarg);

put x= y= z=;

if (x = y) then
  put 'Success on function!';
else
  put 'Failure on function!';

if (z = y) then
  put 'Success on subroutine!';
else
  put 'Failure on subroutine!';

run;
quit;

```

The SAS log returns these results:

```

x='callarg' y='callarg' z='callarg'
Success on function!
Success on subroutine!

```

Examples: FCMP Procedure

Example 1: Creating a Function and Calling the Function from a DATA Step

Features: PROC FCMP statement option
OUTLIB=
DATA step

Details

This example shows how to compute a study day during a drug trial by creating a function in PROC FCMP and using that function in a DATA step.

Program

```
proc fcmp outlib=work.funcs.trial;
    function study_day(intervention_date, event_date);
        n=event_date - intervention_date;
        if n >= 0 then
            n=n + 1;
        return(n);
    endsub;
options cmplib=work.funcs;
data _null_;
    start='15Feb2010'd;
    today='27Mar2010'd;
    sd=study_day(start, today);
    put sd=;
run;
```

Program Description

Specify the name of an output package to which the compiled function and CALL routine are written. The package is stored in the data set work.Funcs.

```
proc fcmp outlib=work.funcs.trial;
```

Create a function called STUDY_DAY. STUDY_DAY is created in a package called Trial, and contains two numeric input arguments.

```
function study_day(intervention_date, event_date);
```

Use a DATA step IF statement to calculate EVENT_DATE. Use DATA step syntax to compute the difference between EVENT_DATE and INTERVENTION_DATE. The days before INTERVENTION_DATE begin at -1 and become smaller. The days after and including INTERVENTION_DATE begin at 1 and become larger. (This function never returns 0 for a study date.)

```
    n=event_date - intervention_date;
    if n >= 0 then
        n=n + 1;
    return(n);
endsub;
```

Use the **CMPLIB=** system option to specify a SAS data set that contains the compiler subroutine to include during program compilation.

```
options cmplib=work.funcs;
```

Create a DATA step to produce a value for the function STUDY_DAY. The function uses a start date and today's date to compute the value. STUDY_DAY is called from the DATA step. When the DATA step encounters a call to STUDY_DAY, it does not find this function in its traditional library of functions. It searches each of the data sets that are specified in the CMPLIB system option for a package that contains STUDY_DAY. In this case, it finds STUDY_DAY in work.Funcs.Trial.

```
data _null_;
    start='15Feb2010'd;
    today='27Mar2010'd;
    sd=study_day(start, today);
```

Write the output to the SAS log.

```
put sd=;
```

Execute the SAS program.

```
run;
```

Log

Example Code 21.6 Results from Creating and Calling a Function from a DATA Step

```
sd=41
```

Example 2: Creating and Saving Functions with PROC FCMP

Features:	PROC FCMP statement option OUTLIB= OUTARGS statement
-----------	--

Details

This example shows how to use PROC FCMP to create and save the functions that are used in the example.

Program

```
proc fcmp outlib=work.exsubs.pkt1;

  subroutine calc_years(maturity, current_date, years);
    outargs years;
    years=(maturity - current_date) / 365.25;
  endsub;

  function garkhprc(type$, buysell$, amount,E, t, S, rd, rf, sig);
    if buysell="Buy" then sign=1.;
    else do;
      if buysell="Sell" then sign=-1.;
      else sign=.;
    end;

    if type="Call" then
      garkhprc=sign * amount * garkhptprc(E, t, S, rd, rf, sig);
    else do;
      if type="Put" then
        garkhprc=sign * amount * garkhptprc(E, t, S, rd, rf, sig);
      else garkhprc=.;
    end;

  return(garkhprc);

run;

endfunc;
```

Program Description

Specify the entry where the function package information is saved. The package is a three-level name.

```
proc fcmp outlib=work.exsubs.pkt1;
```

Create a function to calculate years to maturity. A generic function called CALC_YEARS is declared to calculate years to maturity from date variables that are stored as the number of days. The OUTARGS statement specifies the variable that is updated by CALC_YEARS.

```
  subroutine calc_years(maturity, current_date, years);
    outargs years;
    years=(maturity - current_date) / 365.25;
  endsub;
```

Create a function for Garman-Kohlhagen pricing for FX options. A function called GARKHPRC is declared, which calculates Garman-Kohlhagen pricing for FX options. The function uses the SAS functions GARKHCLPRC and GARKHPTPRC.

```
  function garkhprc(type$, buysell$, amount,E, t, S, rd, rf, sig);
    if buysell="Buy" then sign=1.;
    else do;
      if buysell="Sell" then sign=-1.;
      else sign=.;
    end;
```

```

end;

if type="Call" then
    garkhprc=sign * amount * garkhptprc(E, t, S, rd, rf, sig);
else do;
    if type="Put" then
        garkhprc=sign * amount * garkhptprc(E, t, S, rd, rf, sig);
    else garkhprc=.;
    end;
end;

```

The RETURN statement returns the value of the GARKHPRC function.

```

return(garkhprc);

```

Execute the FCMP procedure. The RUN statement executes the FCMP procedure.

```

run;

```

Close the function. The endfunc statement closes the function.

```

endfunc;

```

Log

Example Code 21.7 Location of Functions That Are Saved

```

NOTE: Function garkhprc saved to work.exsubs.pkt1.
NOTE: Function calc_years saved to work.exsubs.pkt1.

```

Example 3: Using Numeric Data in the FUNCTION Statement

Details

The following example uses numeric data as input to the FUNCTION statement of PROC FCMP.

Program

```

proc fcmp;
    function inverse(in);
        if in=0 then inv=.;
        else inv=1/in;
        return(inv);
    endfunction;
run;

```

```

        endfunc;
run;

```

Example 4: Using Character Data with the FUNCTION Statement

Details

The following example uses character data as input to the FUNCTION statement of PROC FCMP. The output from FUNCTION TEST is assigned a length of 12 bytes.

Program

```

options cmplib=work.funcs;

proc fcmp outlib=work.funcs.math;
  function test(x $) $ 12;
    if x='yes' then
      return('si si si');
    else
      return('no');
    endfunc;
run;

data _null_;
  spanish=test('yes');
  put spanish=;
run;

```

Log

Example Code 21.8 Results from Using Character Data with the FUNCTION Statement in PROC FCMP

```
spanish=si si si
```

Example 5: Using Variable Arguments with an Array

Details

The following example shows an array that accepts variable arguments. The example implies that the summation function can be called as follows:

```
sum=summation(1,2,3,4,5);
```

Note: When calling this function from a DATA step, you must provide the *VARARGS* as an array.

Program

```
options cmplib=work.funcs;

proc fcmp outlib=work.funcs.temp;
function summation (b[*]) varargs;
    total=0;
    do i=1 to dim(b);
        total=total + b[i];
    end;
    return(total);
endfunc;
sum=summation(1, 2, 3, 4, 5);
put sum=;
run;
```

Example 6: Using the SUBROUTINE Statement with a CALL Statement

Details

Here is an example of the SUBROUTINE statement. The SUBROUTINE statement creates an independent computational block of code that can be used with a CALL statement.

Program

```
proc fcmp outlib=work.funcs.temp;
  subroutine inverse(in,inv) group="generic";
    outargs inv;
    if in=0 then inv=.;
    else inv=1/in;
  endsub;

  options cmplib=work.funcs;
  data _null_;
    x=5;
    call inverse(x, y);
    put x= y=;
  run;
```

Log

Example Code 21.9 Results from Using the SUBROUTINE Statement in PROC FCMP

```
x=5 y=0.2
```

Example 7: Using Graph Template Language (GTL) with User-Defined Functions

Features: PROC FCMP functions

OSCILLATE
 OSCILLATEBOUND
 Other procedures
 PROC TEMPLATE
 PROC SGRENDER

Details

The following example shows how to use functions in a GTL EVAL function. It shows how to define functions that define new curve types (oscillate and oscillateBound). These functions can be used in a GTL EVAL function to compute new columns that are presented with a seriesplot and bandplot.

Program

```
proc fcmp outlib=work.funcs.curves;
  function oscillate(x,amplitude,frequency);
    if amplitude le 0 then amp=1; else amp=amplitude;
    if frequency le 0 then freq=1; else freq=frequency;
    y=sin(freq*x)*constant("e")**(-amp*x);
    return (y);
  endfunc;

  function oscillateBound(x,amplitude);
    if amplitude le 0 then amp=1; else amp=amplitude;
    y=constant("e")**(-amp*x);
    return(y);
  endfunc;
run;

options cmplib=work.funcs;

data range;
  do time=0 to 2 by .01;
    output;
  end;
run;

proc template ;
  define statgraph damping;
    dynamic X AMP FREQ;
    begingraph;
    entrytitle "Damped Harmonic Oscillation";
    layout overlay / yaxisopts=(label="Displacement");
    if (exists(X) and exists(AMP) and exists(FREQ))
      bandplot x=X limitlower=eval(-oscillateBound(X,AMP))
        limitupper=eval(oscillateBound(X,AMP));
      seriesplot x=X y=eval(oscillate(X,AMP,FREQ));
    endif;
  end;
end;
```

```

        endlayout;
    endgraph;
end;
run;

proc sgrender data=range template=damping;
    dynamic x="Time" amp=10 freq=50 ;
run;

```

Program Description

Create the OSCILLATE function.

```

proc fcmp outlib=work.funcs.curves;
    function oscillate(x,amplitude,frequency);
        if amplitude le 0 then amp=1; else amp=amplitude;
        if frequency le 0 then freq=1; else freq=frequency;
        y=sin(freq*x)*constant("e")**(-amp*x);
        return (y);
    endfunc;
run;

```

Create the OSCILLATEBOUND function.

```

function oscillateBound(x,amplitude);
    if amplitude le 0 then amp=1; else amp=amplitude;
    y=constant("e")**(-amp*x);
    return(y);
endfunc;
run;

```

Create a data set called Range that is used by PROC SGRENDER.

```

options cmplib=work.funcs;

data range;
    do time=0 to 2 by .01;
        output;
    end;
run;

```

Use the TEMPLATE procedure to customize the appearance of your SAS output.

```

proc template ;
    define statgraph damping;
        dynamic X AMP FREQ;
        begingraph;
            entrytitle "Damped Harmonic Oscillation";
            layout overlay / yaxisopts=(label="Displacement");
            if (exists(X) and exists(AMP) and exists(FREQ))
                bandplot x=X limitlower=eval(-oscillateBound(X,AMP))
                    limitupper=eval(oscillateBound(X,AMP));
                seriesplot x=X y=eval(oscillate(X,AMP,FREQ));
            endif;
        endlayout;
    endgraph;
end;

```

```
run;
```

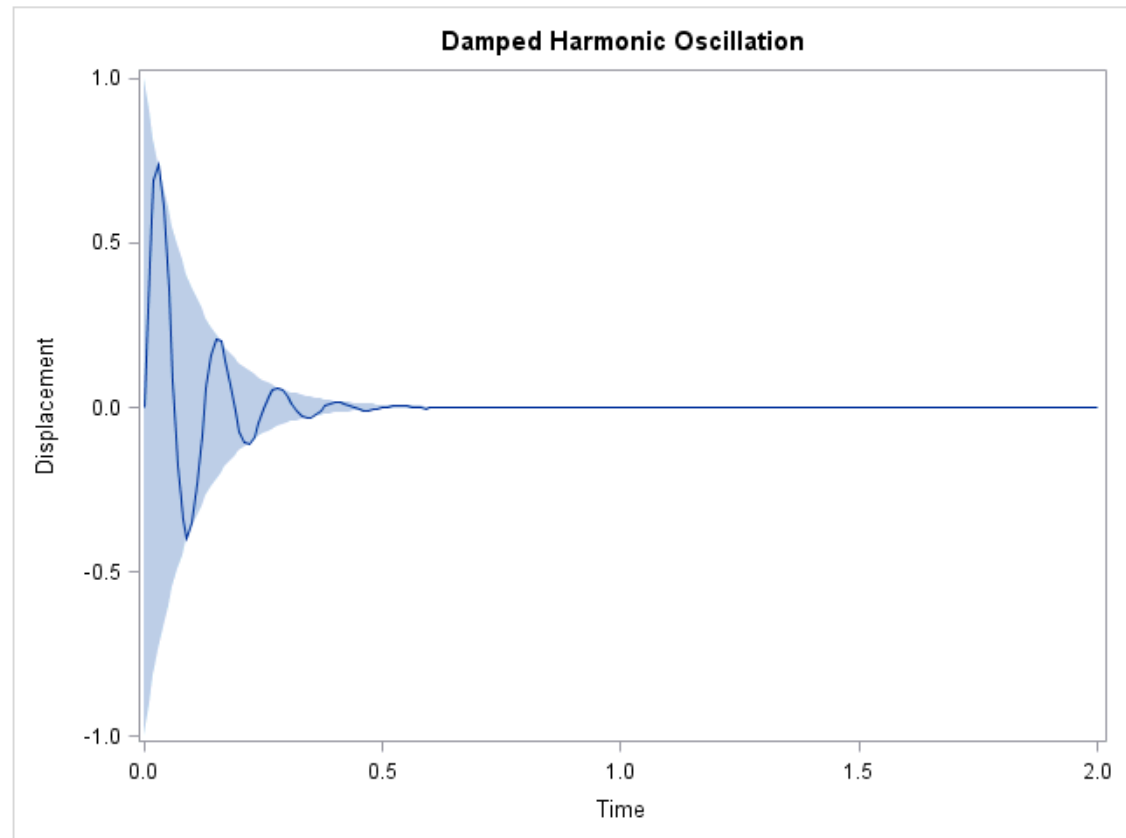
Use the **SGRENDER** procedure to identify the data set that contains the input variables and to assign a statgraph template for the output.

```
proc sgrender data=range template=damping;
  dynamic x="Time" amp=10 freq=50 ;
run;
```

Output: Using GTL with Functions

Output 21.7 Results from Using GTL with User-Defined Functions

The SAS System



Example 8: Standardizing Each Row of a Data Set

Features:

PROC FCMP functions
 RUN_MACRO
 RUN_SASFILE
 READ_ARRAY
 WRITE_ARRAY

Details

This example shows how to standardize each row of a data set.

Program

```
data numbers;
  drop i j;
  array a[5];
  do j=1 to 5;
    do i=1 to 5;
      a[i] = rand('Uniform') * (i+123.234);
    end;
  output;
end;
run;

%macro standardize;
  %let dsname=%sysfunc(dequote(&dsname));
  %let colname=%sysfunc(dequote(&colname));
  proc standard data=&dsname mean=&MEAN std=&STD out=_out;
    var &colname;
  run;
  data &dsname;
    set _out;
  run;
%mend standardize;

proc fcmp outlib=work.ds.functions;
  subroutine standardize(x[*], mean, std);
    outargs x;

    rc=write_array('work._TMP_', x, 'x1');
    dsname='work._TMP_';
    colname='x1';
    rc=run_macro('standardize', dsname, colname, mean, std);
    array x2[1]_temporary_;
    rc=read_array('work._TMP_', x2);
    if dim(x2)=dim(x) then do;
      do i=1 to dim(x);
        x[i]=x2[i];
      end;
    end;
  endsub;
run;

options cmlib=(work.ds);
data numbers2;
  set numbers;
  array a[5];
```

```

array t[5]_temporary_;
do i=1 to 5;
    t[i]=a[i];
end;
call standardize(t, 0, 1);
do i=1 to 5;
    a[i]=t[i];
end;
output;
run;

proc print data=work.numbers2;
run;

```

Program Description

Create a data set that contains five rows of random numbers.

```

data numbers;
    drop i j;
    array a[5];
    do j=1 to 5;
        do i=1 to 5;
            a[i] = rand('Uniform') * (i+123.234);
        end;
        output;
    end;
run;

```

Create a macro to standardize a data set with a given value for mean and std.

```

%macro standardize;
    %let dsname=%sysfunc(dequote(&dsname));
    %let colname=%sysfunc(dequote(&colname));
    proc standard data=&dsname mean=&MEAN std=&STD out=_out;
        var &colname;
    run;
    data &dsname;
        set _out;
    run;
%mend standardize;

```

Use the FCMP function to call WRITE_ARRAY, which writes the data to a data set. Call RUN_MACRO to standardize the data in the data set. Call WRITE_ARRAY to write data to a data set. Call READ_ARRAY to read the standardized data back into the array.

```

proc fcmp outlib=work.ds.functions;
    subroutine standardize(x[*], mean, std);
        outargs x;

        rc=write_array('work._TMP_', x, 'x1');
        dsname='work._TMP_';
        colname='x1';
        rc=run_macro('standardize', dsname, colname, mean, std);
        array x2[1]_temporary_;
    endsub;
endproc;

```

```

        rc=read_array('work._TMP_', x2);
    if dim(x2)=dim(x) then do;
        do i=1 to dim(x);
            x[i]=x2[i];
        end;
    end;
endsub;
run;

```

Execute the function for each row in the DATA step.

```

options cmplib=(work.ds);
data numbers2;
    set numbers;
    array a[5];
    array t[5]_temporary_;
    do i=1 to 5;
        t[i]=a[i];
    end;
    call standardize(t, 0, 1);
    do i=1 to 5;
        a[i]=t[i];
    end;
    output;
run;

```

Write the output.

```

proc print data=work.numbers2;
run;

```

Output: Standardizing Rows in a Data Set

Output 21.8 *Results from Standardizing Each Row of a Data Set*

The SAS System

Obs	a1	a2	a3	a4	a5
1	45.088	93.3237	104.908	35.152	23.5725
2	90.552	9.7548	92.696	89.987	97.9810
3	60.596	22.7409	19.284	50.079	58.9264
4	106.778	49.1589	22.885	20.641	30.1756
5	34.812	71.3746	44.248	101.808	79.3731

References

- SAS Institute Inc. 2013. *SAS Component Objects: Reference*. Cary, NC: SAS Institute Inc.
- Henrick, A., D. Erdman, and S. Christian. 2013. "Hashing in PROC FCMP to Enhance Your Productivity." *Proceedings of the SAS Global Forum 2013 Conference*, Cary, NC. SAS Institute Inc., 1–15. Available at <http://support.sas.com/resources/papers/proceedings13/129-2013.pdf>.

FCMP Special Functions and Call Routines

<i>Overview of Special Functions and CALL Routines</i>	748
<i>Functions and CALL Routines by Category</i>	748
<i>Dictionary</i>	750
CALL ADDMATRIX Routine	750
CALL CHOL Routine	752
CALL DET Routine	753
CALL DYNAMIC_ARRAY Routine	754
CALL ELEMULT Routine	756
CALL EXPMATRIX Routine	757
CALL FILLMATRIX Routine	758
CALL IDENTITY Routine	759
CALL INV Routine	760
CALL MULT Routine	761
CALL POWER Routine	762
CALL SETNULL Routine	763
CALL STRUCTINDEX Routine	764
CALL SUBTRACTMATRIX Routine	765
CALL TRANSPOSE Routine	766
CALL ZEROMATRIX Routine	767
INVCDF Function	768
ISNULL Function	772
LIMMOMENT Function	773
NDIMS Function	777
READ_ARRAY Function	778
RUN_MACRO Function	780
RUN_SASFILE Function	784
SOLVE Function	786
WRITE_ARRAY Function	790

Overview of Special Functions and CALL Routines

The FCMP procedure provides a small set of special use functions. You can call these functions from user-defined FCMP functions but you cannot call these functions directly from the DATA step. To use these functions in a DATA step, you must wrap the special function inside another user-defined FCMP function.

Note: You can call special functions directly in a procedure, but not in the DATA step.

Functions and CALL Routines by Category

Table 22.1 Categories of FCMP Functions and CALL Routines

Category	Description
Calling SAS Code from within Functions	Two functions are available that enable you to call SAS code from within functions. The RUN_MACRO function executes a predefined SAS macro. The RUN_SASFILE function executes SAS code from a fileref that you specify.
C Helper	Several helper functions are provided with the package to handle C-language constructs in PROC FCMP. Most C-language constructs must be defined in a package that is created by PROC PROTO before the constructs can be referenced or used by PROC FCMP. The ISNULL function and the SETNULL and STRUCTINDEX CALL routines have been added to extend the SAS language to handle C-language constructs that do not naturally fit into the SAS language.
Arrays	PROC FCMP provides the READ_ARRAY function to read arrays, and the WRITE_ARRAY function to write arrays to a data set. This functionality enables PROC FCMP array data to be processed by SAS programs, macros, and procedures.

Category	Description
Matrix Operations	The FCMP procedure provides you with a number of CALL routines for performing simple matrix operations on declared arrays. These CALL routines are automatically provided by the FCMP procedure. With the exception of ZEROMATRIX, FILLMATRIX, and IDENTITY, the CALL routines listed below do not support matrices or arrays that contain missing values.
Special Purpose	The FCMP procedure provides two special purpose functions: INVCDF and LIMMOMENT. The INVCDF function computes the quantile from any distribution for which you have defined a cumulative distribution function (CDF). The LIMMOMENT function computes the limited moment of any distribution for which you have defined a cumulative distribution function (CDF).

Category	Language Elements	Description
Array	CALL DYNAMIC_ARRAY Routine (p. 754)	Enables an array that is declared within a function to change size in an efficient manner.
	NDIMS Function (p. 777)	Returns the number of dimensions in an array.
	READ_ARRAY Function (p. 778)	Reads data from a SAS data set into a PROC FCMP array variable.
	WRITE_ARRAY Function (p. 790)	Writes data from a PROC FCMP array variable to a data set that can then be used by SAS programs, macros, and procedures.
C Helper	CALL SETNULL Routine (p. 763)	Sets a pointer element of a structure to null.
	CALL STRUCTINDEX Routine (p. 764)	Enables you to access each structure element in an array of structures.
	ISNULL Function (p. 772)	Determines whether a pointer element of a structure is null.
Calling SAS Code from within Functions	RUN_MACRO Function (p. 780)	Executes a predefined SAS macro.
	RUN_SASFILE Function (p. 784)	Executes SAS code in a fileref that you specify.
Compute Implicit Values	SOLVE Function (p. 786)	Computes implicit values of a function using the Gauss-Newton method.
Matrix Operations	CALL ADDMATRIX Routine (p. 750)	Performs an elementwise addition of two matrices or a matrix and a scalar.
	CALL CHOL Routine (p. 752)	Calculates the Cholesky decomposition for a given symmetric matrix.

Category	Language Elements	Description
	CALL DET Routine (p. 753)	Calculates the determinant of a specified matrix that should be square.
	CALL ELEMULT Routine (p. 756)	Performs an elementwise multiplication of two matrices.
	CALL EXPMATRIX Routine (p. 757)	Returns a matrix e^{tA} given the input matrix A and a multiplier t .
	CALL FILLMATRIX Routine (p. 758)	Replaces all of the element values of the input matrix with the specified value.
	CALL IDENTITY Routine (p. 759)	Converts the input matrix to an identity matrix.
	CALL INV Routine (p. 760)	Calculates a matrix that is the inverse of the provided input matrix that should be a square, non-singular matrix.
	CALL MULT Routine (p. 761)	Calculates the multiplicative product of two input matrices.
	CALL POWER Routine (p. 762)	Raises a square matrix to a given scalar value.
	CALL SUBTRACTMATRIX Routine (p. 765)	Performs an element-wide subtraction of two matrices or a matrix and a scalar.
	CALL TRANSPOSE Routine (p. 766)	Returns the transpose of a matrix.
	CALL ZEROMATRIX Routine (p. 767)	Replaces all of the element values of the numeric input matrix with 0.
Special Purpose Functions	INVCDF Function (p. 768)	Computes the quantile from any distribution for which you have defined a cumulative distribution function (CDF).
	LIMMOMENT Function (p. 773)	Computes the limited moment of any distribution for which you have defined a cumulative distribution function (CDF).

Dictionary

CALL ADDMATRIX Routine

Performs an elementwise addition of two matrices or a matrix and a scalar.

Category: Matrix Operations

Requirement: All input and output matrices must have the same dimensions.

Syntax

CALL ADDMATRIX(*X*, *Y*, *Z*);

Required Arguments

X

specifies an input matrix with dimensions $m \times n$ (that is, $X[m, n]$) or a scalar.

Y

specifies an input matrix with dimensions $m \times n$ (that is, $Y[m, n]$) or a scalar.

Z

specifies an output matrix with dimensions $m \times n$ (that is, $Z[m, n]$), such that

$$Z = X + Y$$

Example

The following example uses the ADDMATRIX CALL routine:

```
options pageno=1 nodate;

proc fcmp;
  array mat1[3,2] (0.3, -0.78, -0.82, 0.54, 1.74, 1.2);
  array mat2[3,2] (0.2, 0.38, -0.12, 0.98, 2, 5.2);
  array result[3,2];
  call addmatrix(mat1, mat2, result);
  call addmatrix(2, mat1, result);
  put result=;
quit;
```

Output 22.1 Results from the ADDMATRIX CALL Routine

```

                                The SAS System                                1
                                The FCMP Procedure

result[1, 1]=2.3 result[1, 2]=1.22 result[2, 1]=1.18 result[2, 2]=2.54
result[3, 1]=3.74 result[3, 2]=3.2
```

CALL CHOL Routine

Calculates the Cholesky decomposition for a given symmetric matrix.

Category: Matrix Operations

Alias: CHOLESKY_DECOMP

Requirement: Both input and output matrices must be square and have the same dimensions. X must be symmetric positive-definite, and Y a lower triangle matrix.

Syntax

CALL CHOL(X, Y <, *validate*>);

Required Arguments

X

specifies a symmetric positive-definite input matrix with dimensions $m \times m$ (that is, $X[m, m]$).

Y

specifies an output matrix with dimensions $m \times m$ (that is, $Y[m, m]$). This variable contains the Cholesky decomposition, such that

$$Z = YY^*$$

where Y is a lower triangular matrix with strictly positive diagonal entries and Y^* denotes the conjugate transpose of Y.

Note: If X is not symmetric positive-definite, then Y is filled with missing values.

Optional Argument

validate

specifies an optional argument that can increase the processing speed by avoiding error checking. The argument can take the following values:

- 0 the matrix X checks for symmetry. This is the default if the *validate* argument is omitted.
- 1 the matrix is assumed to be symmetric.

Example

The following example uses the CHOL CALL routine:

```
options pageno=1 nodate;

proc fcmp;
  array xx[3,3] 2 2 3 2 4 2 3 2 6;
  array yy[3,3];
  call chol(xx, yy, 0);
  do i=1 to 3;
    put yy[i, 1] yy[i, 2] yy[i, 3];
  end;
run;
```

Output 22.2 Results from PROC FCMP and the CHOL CALL Routine

The SAS System		1
The FCMP Procedure		
1.4142135624	0	0
1.4142135624	1.4142135624	0
2.1213203436	-0.707106781	1

CALL DET Routine

Calculates the determinant of a specified matrix that should be square.

Category: Matrix Operations

Requirement: The input matrix X must be square.

Syntax

CALL DET(*X*, *a*);

Required Arguments

X

specifies an input matrix with dimensions $m \times n$ (that is, $X[m, n]$).

a

specifies the returned determinate value, such that

$$a = |X|$$

Details

The determinant, the product of the eigenvalues, is a single numeric value. If the determinant of a matrix is zero, then that matrix is singular (that is, it does not have an inverse). The method performs an LU decomposition and collects the product of the diagonals (Forsythe, Malcolm, and Moler 1967). For more information, see the *SAS/IML User's Guide*.

Example

The following example uses the DET CALL routine:

```
options pageno=1 nodate;

proc fcmp;
  array mat1[3,3] (.03, -0.78, -0.82, 0.54, 1.74,
                  1.2, -1.3, 0.25, 1.49);
  call det(mat1, result);
  put result=;
quit;
```

Output 22.3 Results from the DET CALL Routine

```

                                The SAS System                                1
                                The FCMP Procedure
result=-0.052374
```

CALL DYNAMIC_ARRAY Routine

Enables an array that is declared within a function to change size in an efficient manner.

Category: Array

Syntax

CALL DYNAMIC_ARRAY(*array-name*, *new-dimension1-size* <, *new-dimension2-size*, ...>);

Required Arguments

array-name

specifies the name of a temporary array.

new-dimension-size

specifies a new size for the temporary array.

Details

Arrays that are declared in functions and CALL routines can be resized, as well as arrays that are declared with the /NOSYMBOLS option. No other array can be resized.

The DYNAMIC_ARRAY CALL routine attempts to dynamically resize the array to match the dimensions of the target that you provide. This means that the array must be dynamic. That is, the array must be declared either in a function or subroutine, or declared with the /NOSYMBOLS option.

The DYNAMIC_ARRAY CALL routine is passed the array to be resized and a new size for each dimension of the array. In the ALLPERMK routine, a scratch array that is the size of the number of elements being permuted is needed. When the function is created, this value is not known because it is passed in as parameter *n*. A dynamic array enables the routine to allocate the amount of memory that is needed, instead of having to create an array that is large enough to handle all possible cases.

When using dynamic arrays, support is limited to PROC FCMP routines. When an array is resized, the resized array is available only within the routine that resized it. It is not possible to resize a DATA step array or to return a PROC FCMP dynamic array to the DATA step.

Example

The following example creates a temporary array named TEMP. The size of the array area depends on parameters that are passed to the function.

```
proc fcmp;
  function avedev_wacky(data[*]);

    length=dim(data);
    array temp[1] /nosymbols;
    call dynamic_array(temp, length);

    mean=0;
    do i=1 to length;
      mean += data[i];
      if i>1 then temp[i]=data[i-1];
      else temp[i]=0;
    end;
    mean=mean/length;

    avedev=0;
    do i=1 to length;
      avedev += abs((data[i]) - temp[i] /2 - mean);
    end;
```

```

    avedev=avedev/length;

    return(avedev);
endsub;

array data[10];

do i = 1 to 10;

    data[i] = i;
end;

avedev = avedev_wacky(data);
run;

```

CALL ELEMULT Routine

Performs an elementwise multiplication of two matrices.

Category: Matrix Operations

Requirement: All input and output matrices must have the same dimensions.

Syntax

CALL ELEMULT(*X*, *Y*, *Z*);

Required Arguments

X

specifies an input matrix with dimensions $m \times n$ (that is, $X[m, n]$).

Y

specifies an input matrix with dimensions $m \times n$ (that is, $Y[m, n]$).

Z

specifies an output matrix with dimensions $m \times n$ (that is, $Z[m, n]$).

Example

The following example uses the ELEMULT CALL routine:

```

options pageno=1 nodate;

proc fcmp;
    array mat1[3,2] (0.3, -0.78, -0.82, 0.54, 1.74, 1.2);
    array mat2[3,2] (0.2, 0.38, -0.12, 0.98, 2, 5.2);
    array result[3,2];
    call elemmult(mat1, mat2, result);

```

```

      call elemmult(2.5, mat1, result);
      put result=;
      quit;

```

Output 22.4 Results from the ELEMULT CALL Routine

```

                                The SAS System                                1
                                The FCMP Procedure
result[1, 1]=0.75 result[1, 2]=-1.95 result[2, 1]=-2.05 result[2, 2]=1.35
result[3, 1]=4.35 result[3, 2]=3

```

CALL EXPMATRIX Routine

Returns a matrix e^{tA} given the input matrix A and a multiplier t .

Category: Matrix Operations

Requirement: Both input and output matrices must be square and have the same dimensions. t can be any scalar value.

Syntax

CALL EXPMATRIX(X , t , Y);

Required Arguments

X

specifies an input matrix with dimensions $m \times m$ (that is, $X[m, m]$).

t

specifies a double scalar value.

Y

specifies an output matrix with dimensions $m \times m$ (that is, $Y[m, m]$), such that

$$Y = e^{tX}$$

Details

The EXPMATRIX CALL routine uses a Padé approximation algorithm as presented in Golub and van Loan (1989), p. 558. Note that this module does not exponentiate each entry of a matrix. For more information, see the EXPMATRIX documentation in the *SAS/IML User's Guide*.

Example

The following example uses the EXPMATRIX CALL routine:

```
options pageno=1 nodate;

proc fcmp;
  array mat1[3,3] (0.3, -0.78, -0.82, 0.54, 1.74,
                  1.2, -1.3, 0.25, 1.49);
  array result[3,3];
  call expmatrix(mat1, 3, result);
  put result=;
quit;
```

Output 22.5 Results from the EXPMATRIX CALL Routine

```

                                The SAS System                                1
                                -----
                                The FCMP Procedure

result[1, 1]=365.58043585 result[1, 2]=-589.6358476 result[1, 3]=-897.1034008
result[2, 1]=-507.0874798 result[2, 2]=838.64570481 result[2, 3]=1267.3598426
result[3, 1]=-551.588816 result[3, 2]=858.97629382 result[3, 3]=1324.8187125
```

CALL FILLMATRIX Routine

Replaces all of the element values of the input matrix with the specified value.

Category: Matrix Operations

Note: You can use the FILLMATRIX CALL routine with multidimensional numeric arrays.

Syntax

CALL FILLMATRIX(*X*, *Y*);

Required Arguments

X
specifies an input numeric matrix.

Y
specifies the numeric value that fills the matrix.

Example

The following example uses the FILLMATRIX CALL routine.

```
options pageno=1 nodate ls=80 ps=64;

proc fcmp;
  array mat1[3, 2] (0.3, -0.78, -0.82, 0.54, 1.74, 1.2);
  call fillmatrix(mat1, 99);
  put mat1=;
quit;
```

Output 22.6 Results from the FILLMATRIX CALL Routine

```

                                The SAS System                                1
                                The FCMP Procedure

mat1[1, 1]=99 mat1[1, 2]=99 mat1[2, 1]=99 mat1[2, 2]=99 mat1[3, 1]=99
mat1[3, 2]=99
```

CALL IDENTITY Routine

Converts the input matrix to an identity matrix.

Category: Matrix Operations

Requirement: The input matrix must be square.

Note: Diagonal element values of the matrix are set to 1, and the rest of the values are set to 0.

Syntax

CALL IDENTITY(*X*);

Required Argument

X

specifies an input matrix with dimensions $m \times m$ (that is, $X[m, m]$).

Example

The following example uses the IDENTITY CALL routine:

```
options pageno=1 nodate;
```

```

proc fcmp;
  array mat1[3,3] (0.3, -0.78, -0.82, 0.54, 1.74, 1.2,
                  -1.3, 0.25, 1.49);
  call identity(mat1);
  put mat1=;
quit;

```

Output 22.7 Results from the IDENTITY CALL Routine

```

                                The SAS System                                1
                                The FCMP Procedure
mat1[1, 1]=1 mat1[1, 2]=0 mat1[1, 3]=0 mat1[2, 1]=0 mat1[2, 2]=1 mat1[2, 3]=0
mat1[3, 1]=0 mat1[3, 2]=0 mat1[3, 3]=1

```

CALL INV Routine

Calculates a matrix that is the inverse of the provided input matrix that should be a square, non-singular matrix.

Category: Matrix Operations

Requirement: Both the input and output matrices must be square and have the same dimensions.

Syntax

CALL INV(*X*, *Y*);

Required Arguments

X

specifies an input matrix with dimensions $m \times m$ (that is, $X[m, m]$).

Y

specifies an output matrix with dimensions $m \times m$ (that is, $Y[m, m]$), such that

$$Y[m, m] = X'[m, m]$$

where ' denotes inverse

$$X \times Y = Y \times X = I$$

and I is the identity matrix.

Example

The following example uses the INV CALL routine:

```
options pageno=1 nodate;

proc fcmp;
  array mat1[3,3] (0.3, -0.78, -0.82, 0.54, 1.74,
                  1.2, -1.3, 0.25, 1.49);
  array result[3,3];
  call inv(mat1, result);
  put result=;
quit;
```

Output 22.8 Results from the INV CALL Routine

```

                                The SAS System                                1
                                The FCMP Procedure

result[1, 1]=4.0460407887 result[1, 2]=1.6892917399 result[1, 3]=0.8661767509
result[2, 1]=-4.173108283 result[2, 2]=-1.092427483 result[2, 3]=-1.416802558
result[3, 1]=4.230288655 result[3, 2]=1.6571719011 result[3, 3]=1.6645841716
```

CALL MULT Routine

Calculates the multiplicative product of two input matrices.

Category: Matrix Operations

Requirement: The number of columns for the first input matrix must be the same as the number of rows for the second matrix.

Syntax

CALL MULT(*X*, *Y*, *Z*);

Required Arguments

X

specifies an input matrix with dimensions $m \times n$ (that is, $X[m, n]$).

Y

specifies an input matrix with dimensions $n \times p$ (that is, $Y[n, p]$).

Z

specifies an output matrix with dimensions $m \times p$ (that is, $Z[m, p]$), such that

$$Z[m, p] = X[m, n] \times Y[n, p]$$

Example

The following example uses the MULT CALL routine:

```
options pageno=1 nodate;

proc fcmp;
  array mat1[2,3] (0.3, -0.78, -0.82, 0.54, 1.74, 1.2);
  array mat2[3,2] (1, 0, 0, 1, 1, 0);
  array result[2,2];
  call mult(mat1, mat2, result);
  put result=;
quit;
```

Output 22.9 Results from the MULT CALL Routine

```

                                The SAS System                                1
                                The FCMP Procedure
result[1, 1]=-0.52 result[1, 2]=-0.78 result[2, 1]=1.74 result[2, 2]=1.74
```

CALL POWER Routine

Raises a square matrix to a given scalar value.

Category: Matrix Operations

Restriction: Large scalar values should be avoided because the POWER CALL routine's internal use of the matrix multiplication routine might cause numerical precision problems.

Requirement: Both input and output matrices must be square and have the same dimensions.

Syntax

CALL POWER(*X*, *a*, *Y*);

Required Arguments

X

specifies an input matrix with dimensions $m \times m$ (that is, $X[m, m]$).

a

specifies an integer scalar value (power).

Y

specifies an output matrix with dimensions $m \times m$ (that is, $Y[m, m]$), such that

$$Y = X^a$$

Details

If the scalar is not an integer, it is truncated to an integer. If the scalar is less than 0, then it is changed to 0. For more information, see the *SAS/IML User's Guide*.

Example

The following example uses the POWER CALL routine:

```
options pageno=1 nodate;

proc fcmp;
  array mat1[3,3] (0.3, -0.78, -0.82, 0.54, 1.74,
                  1.2, -1.3, 0.25, 1.49);
  array result[3,3];
  call power(mat1, 3, result);
  put result=;
quit;
```

Output 22.10 Results from the POWER CALL Routine

```

                                The SAS System                                1

                                The FCMP Procedure

result[1, 1]=2.375432 result[1, 2]=-4.299482 result[1, 3]=-6.339638
result[2, 1]=-3.031224 result[2, 2]=6.272988 result[2, 3]=8.979036
result[3, 1]=-4.33592 result[3, 2]=5.775695 result[3, 3]=9.326529
```

CALL SETNULL Routine

Sets a pointer element of a structure to null.

Category: C Helper

Syntax

CALL SETNULL(*pointer-element*);

Required Argument

pointer-element

is a pointer to a structure.

Example

The following example assumes a LINKLIST structure that is defined using PROC PROTO. The CALL SETNULL routine can be used to set the NEXT element to null:

```
struct linklist list;
call setnull(list.next);
```

CALL STRUCTINDEX Routine

Enables you to access each structure element in an array of structures.

Category: C Helper

Syntax

CALL STRUCTINDEX(*structure-array*, *index*, *structure-element*);

Required Arguments

structure-array

specifies an array.

index

is a 1-based index as used in most SAS arrays.

structure-element

points to an element in the array.

Example

In the first part of this example, the following structures and function are defined by using PROC PROTO.

```
proc proto package=sasuser.mylib.str2;
struct point{
  short s;
  int i;
  long l;
  double d;
};

struct point_array {
  int length;
  struct point p[2];
  char name[32];
};
run;
```

In the second part of this example, the PROC FCMP code segment shows how to use the STRUCTINDEX CALL routine to retrieve and set each point structure element of an array called P in the POINT_ARRAY structure:

```
options pageno=1 nodate ls=80 ps=64;

proc fcmp libname=sasuser.mylib;
  struct point_array pntarray;
  struct point pnt;

  pntarray.length=2;
  pntarray.name="My funny structure";

  /* Get each element using the STRUCTINDEX CALL routine and set
  values. */
  do i=1 to 2;
    call structindex(pntarray.p, i, pnt);
    put "Before setting the" i "element: " pnt=;
    pnt.s=1;
    pnt.i=2;
    pnt.l=3;
    pnt.d=4.5;
    put "After setting the" i "element: " pnt=;
  end;
run;
```

Output 22.11 Results of Setting the Point Structure Elements of an Array

The SAS System	1
The FCMP Procedure	
Before setting the 1 element: pnt {s=0, i=0, l=0, d=0}	
After setting the 1 element: pnt {s=1, i=2, l=3, d=4.5}	
Before setting the 2 element: pnt {s=0, i=0, l=0, d=0}	
After setting the 2 element: pnt {s=1, i=2, l=3, d=4.5}	

CALL SUBTRACTMATRIX Routine

Performs an element-wide subtraction of two matrices or a matrix and a scalar.

Category: Matrix Operations

Requirement: All input and output matrices must have the same dimensions.

Syntax

CALL SUBTRACTMATRIX(*X*, *Y*, *Z*);

Required Arguments

X

specifies an input matrix with dimensions $m \times n$ (that is, $X[m, n]$) or a scalar.

Y

specifies an input matrix with dimensions $m \times n$ (that is, $Y[m, n]$) or a scalar.

Z

specifies an output matrix with dimensions $m \times n$ (that is, $Z[m, n]$), such that

$$Z = X - Y$$

Example

The following example uses the SUBTRACTMATRIX CALL routine:

```
options pageno=1 nodate;

proc fcmp;
  array mat1[3,2] (0.3, -0.78, -0.82, 0.54, 1.74, 1.2);
  array mat2[3,2] (0.2, 0.38, -0.12, 0.98, 2, 5.2);
  array result[3,2];
  call subtractmatrix(mat1, mat2, result);
  call subtractmatrix(2, mat1, result);
  put result=;
quit;
```

Output 22.12 Results from the SUBTRACTMATRIX CALL Routine

```

                                The SAS System                                1
                                The FCMP Procedure

result[1, 1]=1.7 result[1, 2]=2.78 result[2, 1]=2.82 result[2, 2]=1.46
result[3, 1]=0.26 result[3, 2]=0.8
```

CALL TRANSPOSE Routine

Returns the transpose of a matrix.

Category: Matrix Operations

Syntax

CALL TRANSPOSE(*X*, *Y*);

Required Arguments

X

specifies an input matrix with dimensions $m \times n$ (that is, $X[m, n]$).

Y

specifies an output matrix with dimensions $n \times m$ (that is, $Y[n, m]$)

Details

$$Y = X'$$

Note that the number of rows for the input matrix should be equal to the number of columns of the output matrix, and the number of rows for the output matrix should be equal to the number of columns of the input matrix.

Example

The following example uses the TRANSPOSE CALL routine:

```
options pageno=1 nodate;

proc fcmp;
  array mat1[3,2] (0.3, -0.78, -0.82, 0.54, 1.74, 1.2);
  array result[2,3];
  call transpose(mat1, result);
  put result=;
quit;
```

Output 22.13 Results from the TRANSPOSE CALL Routine

```

                                The SAS System                                1
                                The FCMP Procedure
result[1, 1]=0.3 result[1, 2]=-0.82 result[1, 3]=1.74 result[2, 1]=-0.78
result[2, 2]=0.54 result[2, 3]=1.2
```

CALL ZEROMATRIX Routine

Replaces all of the element values of the numeric input matrix with 0.

Category: Matrix Operations

Note: You can use the ZEROMATRIX CALL routine with multi-dimensional numeric arrays.

Syntax

CALL ZEROMATRIX(*X*);

Required Argument

X
specifies a numeric input matrix.

Example

The following example uses the ZEROMATRIX CALL routine:

```
options pageno=1 nodate;

proc fcmp;
  array mat1[3,2] (0.3, -0.78, -0.82, 0.54, 1.74, 1.2);
  call zeromatrix(mat1);
  put mat1=;
quit;
```

Output 22.14 Results from the ZEROMATRIX CALL Routine

```

                                The SAS System                                1
                                -----
                                The FCMP Procedure
                                -----
mat1[1, 1]=0 mat1[1, 2]=0 mat1[2, 1]=0 mat1[2, 2]=0 mat1[3, 1]=0 mat1[3,
2]=0
```

INVCDF Function

Computes the quantile from any distribution for which you have defined a cumulative distribution function (CDF).

Category: Special Purpose Functions

Notes: INVCDF is a special purpose function that is automatically provided by the FCMP procedure for your convenience.

If you specify a probability p for a distribution with CDF denoted by $F(x; \langle \text{parameters} \rangle)$, then the INVCDF function returns a quantile q that satisfies $F(q; \langle \text{parameters} \rangle) = p$. In other words, $q = F^{-1}(p)$.

Syntax

quantile=**INVCDF**('CDF-function-name', options-array, cumulative-probability, parameter-1 <, parameter-2, ...>);

Required Arguments

quantile

specifies the quantile that is returned from the INVCDF function.

'CDF-function-name'

specifies the name of the CDF function. Enclose *CDF-function-name* in quotation marks.

Requirement *CDF-function-name* must be a function defined using the FCMP procedure. It must have a signature as follows:

```
function <CDF-function-name> (x, parameter-1 <, parameter-2, ...>);
endsub;
```

Note It is recommended that the CDF be a continuous function. For discrete CDF, the INVCDF function might not be able to compute the quantile.

options-array

specifies an array of options to use with the INVCDF function. *Options-array* is used to control and monitor the process of inverting the CDF. *Options-array* can be a missing value (.), or it can have up to four of the following elements in the following order:

initial-value

specifies the initial guess for the quantile at which the inversion process starts. This is useful when you have an idea of the approximate value for quantile (for example, from the empirical estimate of the CDF).

Default 0.1

desired-accuracy

specifies the desired relative accuracy of the quantile. You can specify any value in the range (0,0.1). If you specify a smaller value, the result is a more accurate estimate of the quantile, but it might take longer to invert the CDF.

Default 1.0e-8

domain-type

specifies the domain for the CDF function. A missing value or a value of 0 indicates a nonnegative support, that is $[0, \infty)$. Any other value indicates a support over the entire real line, that is $(-\infty, \infty)$.

Default 0

return-code

specifies the return status. If *options-array* is of dimension 4 or more, then the fourth element contains the return status. *Return-code* can have one of the following values:

<=0

indicates success. If negative, then the absolute value is the number of times the CDF function was evaluated in order to compute the quantile. A larger absolute value indicates longer convergence time.

1

indicates that the quantile could not be computed.

cumulative-probability

specifies the cumulative probability value for which the quantile is desired.

Range [0,1)

parameter

specifies the parameters of the distribution at which the quantile is desired. You must specify exactly the same number of parameters as required by the specified CDF function, and they should appear exactly in the same order as required by the specified CDF function.

Details

The INVCDF function finds the quantile for the specified cumulative probability from a distribution whose cumulative distribution function is specified by the *CDF-function-name* argument. In other words, it inverts the CDF function such that the following expression is true:

$$\text{cumulative-probability} = \text{CDF-function-name}(\text{quantile}, \langle \text{parameters} \rangle)$$

If ε denotes the desired accuracy of the quantile for cumulative probability p , then INVCDF attempts to compute the quantile q such that $|p - F(q)| < \varepsilon p$, where $F(x)$ denotes the CDF evaluated at x .

You can control the inversion process with various options. Here is an example of an options array:

```
array opts[4] initial epsilon support (1.5 1.0e-6 0);
```

These values refer to the preceding line of code:

```
initial(initial-value)=1.5
```

```
epsilon(desired-accuracy)=1.0e-6
```

```
support(domain-type)=0
```

You can examine the return status of the function by checking `opts[4]`.

Comparisons

You can regard this function as a generic extension of the QUANTILE function, which computes quantiles only from specific distributions. The INVCDF function enables you to compute quantiles from any continuous distribution as long as you can programmatically define that distribution's CDF function. Unlike the QUANTILE function, this function cannot be used directly in a DATA step. It only can be used inside the definition of an FCMP function or subroutine. However, this is not a limitation because you can invoke the FCMP function that uses it from a DATA step. See the following example.

Example: Generating a Random Sample from an Exponential Distribution

The following example demonstrates how you can generate a random sample from any parametric distribution in a DATA step using the INVCDF function. The example uses the exponential distribution for illustration, but it can be extended to any distribution for which you can programmatically define a CDF function. The following statements define an FCMP function EXP_QUANTILE that uses the INVCDF function and the CDF function to compute a quantile from the exponential distribution.

```
proc fcmp library=work.mycdf outlib=work.myquantile.functions;
  function exp_quantile(cdf, theta, rc);
    outargs rc;
    array opts[4] / nosym(0.1 1.0e-8.);
    q=invcdf("exp_cdf", opts, cdf, theta);
    rc=opts[4]; /* return code */
    return(q);
  endsub;
quit;
```

The preceding code assumes that you have stored the definition of the EXP_CDF function in an FCMP library called Work.Mycdf using a PROC FCMP step as follows:

```
proc fcmp outlib=work.mycdf.functions;
  function exp_cdf(x, theta);
    return(1.0 - exp(-x/Theta));
  endsub;
quit;
```

Now you can invoke the EXP_QUANTILE function from a DATA step to generate a random sample from the exponential distribution with a scale parameter (theta) that has a value of 50. Note that the locations of the EXP_CDF and EXP_QUANTILE functions need to be specified with the appropriate value for the CMPLIB= option before you execute the DATA step:

```
options cmplib=(work.mycdf work.myquantile);

data exp_sample(keep=q);
  n=0;k=0;
  do k=1 to 500;
    if (n=100) then leave;
```

```

        rcode=. ;
        q=exp_quantile(rand('UNIFORM'), 50, rcode);
        if (rcode <= 0) then do;
            n=n+1;
            output;
        end;
    end;
end;
run;

```

ISNULL Function

Determines whether a pointer element of a structure is null.

Category: C Helper

Syntax

numeric-variable = **ISNULL** (*pointer-element*);

Required Arguments

numeric-variable

specifies a numeric value.

pointer-element

specifies a variable that contains the address of another variable.

Examples

Example 1: Generating a Linked List

In the following example, the LINKLIST structure and GET_LIST function are defined by using PROC PROTO. The GET_LIST function is an external C routine that generates a linked list with as many elements as requested.

```

struct linklist{
    double value;
    struct linklist * next;
};

struct linklist * get_list(int);

```

Example 2: Using the ISNULL C Helper Function in a Loop

The following code segment shows that the ISNULL C helper function loops over the linked list that is created by GET_LIST and writes out the elements.

```

proc proto package=sasuser.mylib.str2;

```

```

struct linklist{
    double value;
    struct linklist * next;
};

struct linklist * get_list(int);

externc get_list;
struct linklist * get_list(int len){
    int i;
    struct linklist * list=0;
    list=(struct linklist*)
        malloc(len*sizeof(struct linklist));
    for (i=0;i<len-1;i++){
        list[i].value=i;
        list[i].next=&list[i+1];
    }
    list[i].value=i;
    list[i].next=0;
    return list;
}
externcend;
run;

options pageno=1 nodate ls=80 ps=64;
proc fcmp libname=sasuser.mylib;
    struct linklist list;
    list=get_list(3);
    put list.value=;

    do while (^isnull(list.next));
        list=list.next;
        put list.value=;
    end;
run;

```

Output 22.15 Results from Using the ISNULL C Helper Function

The SAS System	1
The FCMP Procedure	
list.value=0	
list.value=1	
list.value=2	

LIMMOMENT Function

Computes the limited moment of any distribution for which you have defined a cumulative distribution function (CDF).

Category: Special Purpose Functions

Notes: LIMMOMENT is a special purpose function that is automatically provided by the FCMP procedure for your convenience.

If you specify order k and upper limit u , then the LIMMOMENT function computes the limited moment as $E[\min(X,u)^k]$, where E denotes an expectation taken over the distribution of a random variable X that is defined by the specified CDF function.

Syntax

```
imom=LIMMOMENT('CDF-function-name', options-array, order, limit,  
parameter-1 <, parameter-2, ...>);
```

Required Arguments

imom

specifies the limited moment that is returned from the LIMMOMENT function.

'CDF-function-name'

specifies the name of the CDF function. Enclose *CDF-function-name* in quotation marks.

Requirement *CDF-function-name* must be a function defined using the FCMP procedure. It must have a signature as follows:

```
function <CDF-function-name> (x, parameter-1 <, parameter-2, ...>);  
endsub;
```

Note It is recommended that the CDF be a continuous function. For discrete CDF, the LIMMOMENT function might not be able to compute the limited moment.

options-array

specifies an array of options to use with the LIMMOMENT function. *Options-array* is used to control and monitor the process of numerical integration used to compute the limited amount. *Options-array* can be a missing value (.), or it can have up to four of the following elements in the following order

desired-accuracy

specifies the desired accuracy of the numerical integration. You can specify any value in the range (0,0.1). If you specify a smaller value, the result is a more accurate estimate of the moment, but it takes longer to compute the desired-accuracy.

Default 1.0e-8

initial-step-size

specifies the step size that is used initially by the numerical integration process. An increase in the value results in a linear decrease in the number of times the integrand is evaluated. Typically, using the default value of 1 produces good results.

Default 1

maximum-iterations

specifies the maximum number of iterations that are used to refine the integration result in order to achieve the desired accuracy. An increase in this value results in an exponential increase in the number of times the integrand is evaluated.

Default 8

return-code

specifies the return status. If *options-array* is of dimension 4 or more, then the fourth element contains the return status. *Return-code* can have one of the following values:

<=0

indicates success. If negative, then the absolute value is the number of times the integrand function was evaluated in order to compute the limited moment. A larger absolute value indicates longer convergence time.

1

indicates that the limited moment could not be computed.

order

specifies the order of the desired limited moment.

Range [1,10]

limit

specifies the upper limit that is used to compute the desired limited moment.

Requirement The value of *limit* must be greater than 0.

parameter

specifies the parameters of the distribution at which the limited moment is desired. You must specify exactly the same number of parameters as required by the specified CDF function, and they should appear exactly in the same order as required by the specified CDF function.

Details

Let a random variable X have a probability distribution with probability density function $f(x;\theta)$ and cumulative distribution function $F(x;\theta)$, where θ denotes the parameters of the distribution. For a specified upper limit u , the k^{th} -order limited moment of this distribution is defined as follows:

$$E[(X \wedge u)^k] = \int_0^u x^k f(x) dx + u^k \int_u^\infty f(x) dx = \int_0^u x^k f(x) dx + u^k (1 - F(u))$$

The LIMMOMENT function uses the following alternate expression:

$$E[(X \wedge u)^k] = k \int_0^u x^{k-1} [1 - F(x)] dx$$

Because the expression needs only $F(x)$, you need to specify only the CDF function for the distribution. Limited moments are often used in insurance applications to compute the maximum amount expected to be paid if the policy limit is set at a certain value.

You can control the numerical integration process with various options. Here is an example of an options array:

```
array opts[4] epsilon initial maxiter (1.0e-5 1 6);
```

These values refer to the preceding line of code.

```
epsilon(desired-accuracy)=1.0e-5
```

```
initial(initial-step)=1
```

```
maxiter(maximum-iterations)=6
```

You can examine the return status of the function by checking `opts[4]`.

Example: Computing a Limited Moment for a Lognormal Distribution

This example demonstrates how you can compute the limited moment for any parametric distribution in a DATA step using the LIMMOMENT function. The example uses the lognormal distribution for illustration, but it can be extended to any distribution for which you can programmatically define a CDF function. The following statements define an FCMP function LOGN_LIMMOMENT that uses the LIMMOMENT function and the CDF function to compute limited moments from the lognormal distribution:

```
proc fcmp library=work.mycdf outlib=work.mylimmom.functions;
  function logn_limmoment(order, limit, mu, sigma, rc);
    outargs rc;
    array opts[4] / nosym (1.0e-8 . . .);

    m=limmoment("logn_cdf", opts, order, limit, mu, sigma);
    rc=opts[4]; /* return code */

    return(m);
  endsub;
quit;
```

The preceding code assumes that you have stored the definition of the LOGN_CDF function in an FCMP library called `Work.Mycdf` using a PROC FCMP step as follows:

```
proc fcmp outlib=work.mycdf.functions;
  function logn_cdf(x, Mu, Sigma);
    if (x >= constant('MACEPS')) then do;
      z=(log(x) - Mu)/Sigma;
      return(CDF('NORMAL',z));
    end;
    return (0);
  endsub;
quit;
```

You can now invoke the LOGN_LIMMOMENT function from a DATA step as shown below. Note that the location of the LOGN_CDF and LOGN_LIMMOMENT functions must be specified with an appropriate value for the CMPLIB= option before you execute the DATA step:

```
options cmplib=(work.mycdf work.mylimmom);

data _null_;
  do order=1 to 3;
    rcode=.;
    m=logn_limmoment(order, 100, 5, 0.5, rcode);
    if (rcode > 0) then
      put "ERROR: Limited moment could not be computed.";
    else
      put 'Moment of order ' order ' with limit 100 = ' m;
  end;
run;
```

NDIMS Function

Returns the number of dimensions in an array.

Category: Array

Syntax

rc = **NDIMS**(*array-name*)

Required Arguments

array-name

specifies the name of the array.

rc

specifies the variable to return the number of dimensions to.

Details

The NDIMS function returns the number of dimensions of a multidimensional array, or returns 1 for a one-dimensional array.

Example

```
proc fcmp;
  array my_onearray[3] 1 2 3;
```

```

        array my_multiarray[2,3] x11 x12 x13 x21 x22 x23;
        n_one=ndims(my_onearray);
        n_multi=ndims(my_multiarray);
        put n_one= n_multi=;
run;
quit;

```

READ_ARRAY Function

Reads data from a SAS data set into a PROC FCMP array variable.

Category: Array

Syntax

```
rc = READ_ARRAY(data_set_name, array_variable < , 'column_name_1',
'column_name_2' , ... > );
```

Required Arguments

rc

is 0 if the function is able to successfully read the data set.

data_set_name

specifies the name of the data set from which the array data is read.

Data_set_name must be a character literal or variable that contains the member name (libname.memname) of the data set to be read from.

array_variable

specifies the PROC FCMP array variable into which the data is read.

Array_variable must be a local temporary array variable because the function might need to grow or shrink its size to accommodate the size of the data set.

Optional Argument

column_name

specifies optional names for the specific columns of the data set that are read.

If specified, *column_name* must be a literal string enclosed in quotation marks.

Column_name cannot be a PROC FCMP variable. If column names are not specified, PROC FCMP reads all of the columns in the data set.

Details

When SAS translates between an array and a data set, the array is indexed as [row,column].

Arrays that are declared in functions and CALL routines can be resized, as well as arrays that are declared with the /NOSYMBOLS option. No other arrays can be resized.

The READ_ARRAY function attempts to dynamically resize the array to match the dimensions of the input data set. This means that the array must be dynamic. That is, the array must be declared either in a function or CALL routine or declared with the /NOSYMBOLS option.

Example

This example creates and reads a SAS data set into an FCMP array variable.

```
options nodate pageno=1;

data account;
  input acct price cost;
  datalines;
1 2 3
4 5 6
;
run;

proc fcmp;
  array x[2,3] / nosymbols;
  rc=read_array('account',x);
  put x=;
run;

proc fcmp;
  array x[2,2] / nosymbols;
  rc=read_array('account', x, 'price', 'acct');
  put x=;
run;
```

Output 22.16 Results from the READ_ARRAY Function

The SAS System		1
The FCMP Procedure		
x[1, 1]=1 x[1, 2]=2 x[1, 3]=3 x[2, 1]=4 x[2, 2]=5 x[2, 3]=6		

The SAS System		2
The FCMP Procedure		
x[1, 1]=2 x[1, 2]=1 x[2, 1]=5 x[2, 2]=4		

RUN_MACRO Function

Executes a predefined SAS macro.

Category: Calling SAS Code from within Functions

Note: The behavior of this function is similar to executing %macro_name; in SAS.

Syntax

```
rc = RUN_MACRO ('macro_name' <, variable_1, variable_2, ...>);
```

Required Arguments

rc

is 0 if the function is able to submit the macro. The return code indicates only that the macro call was attempted. The macro itself should set the value of a SAS macro variable that corresponds to a PROC FCMP variable to determine whether the macro executed as expected.

macro_name

specifies the name of the macro to be run.

Requirement *Macro_name* must be a string enclosed in quotation marks or a character variable that contains the macro to be executed.

Optional Argument

variable

specifies optional PROC FCMP variables, which are set by macro variables of the same name. These arguments must be PROC FCMP double or character variables.

Before SAS executes the macro, SAS macro variables are defined with the same name and value as the PROC FCMP variables. After SAS executes the macro, the macro variable values are copied back to the corresponding PROC FCMP variables.

Examples

Example 1: Executing a Predefined Macro with PROC FCMP

This example creates a macro called %TESTMACRO, and then uses the macro within PROC FCMP to subtract two numbers.

```
/* Create a macro called %TESTMACRO. */
```

```

%macro testmacro;
    %let p=%sysevalf(&a - &b);
%mend testmacro;

/* Use %TESTMACRO within a function in PROC FCMP to subtract two
numbers. */
proc fcmp outlib=sasuser.ds.functions;
    function subtract_macro(a, b);
        rc=run_macro('testmacro', a, b, p);
        if rc eq 0 then return(p);
        else return(.);
    endsub;
run;

/* Make a call from the DATA step. */
option cmplib=(sasuser.ds);

data _null_;
    a=5.3;
    b=0.7;
    p=.;
    p=subtract_macro(a, b);
    put p=;
run;

```

Example Code 22.1 Results from Executing a Predefined Macro with PROC FCMP

```
p=4.6
```

Example 2: Comparing Results from the RUN_MACRO and the DOSUBL Functions

This example creates the %TESTMACRO macro, and compares how the DOSUBL function invokes the same %TESTMACRO as RUN_MACRO. In the RUN_MACRO invocation, all variables are passed as arguments so that they can be set. In the DOSUBL invocation, all macro variables are imported to the code to be executed, then exported on completion. The macro variables &a and &b are made available to %TESTMACRO and the macro variable &p is made available to the caller of DOSUBL. For more information, see [“DOSUBL Function” in SAS Functions and CALL Routines: Reference](#).

```

/* Create a macro called %TESTMACRO. */
%macro testmacro;
    %let p=%sysevalf(&a - &b);
%mend testmacro;

/* Use %TESTMACRO within a function in PROC FCMP to subtract two
numbers. */
proc fcmp outlib=sasuser.ds.functions;
    function subtract_macro(a, b);
        rc=run_macro('testmacro', a, b, p);
        if rc eq 0 then return(p);
        else return(.);
    endsub;
run;

```

```

        /* Make a call from the DATA step. */
option cmplib=(sasuser.ds);

data _null_;
    a=5.3;
    b=0.7;
    p=.;
    p=subtract_macro(a, b);
    put p= '(RUN_MACRO function and a DATA step)';
run;

%global a b p;
%put p=&p;
    /* The value should not yet be known. */
    %let a=5.3;
    %let b=0.7;
data _null_;
    rc=dosubl('%testmacro');
    run;
    %put p=&p (DOSUBL function);

```

Example Code 22.2 Results from the RUN_MACRO and the DOSUBL Functions

```

p=4.6 (RUN_MACRO function and a DATA step)
p=4.6 (DOSUBL function)

```

Example 3: Executing a DATA Step within a DATA Step

This example shows how to execute a DATA step from within another DATA step. The program consists of the following sections:

- The first section of the program creates a macro called APPEND_DS. This macro can write to a data set or append a data set to another data set.
- The second section of the program calls the macro from function writeDataset in PROC FCMP.
- The third section of the program creates the SALARIES data set and divides the data set into four separate data sets depending on the value of the variable Department.
- The fourth section of the program writes the results to the output window.

```

        /* Create a macro called APPEND_DS. */
%macro append_ds;
    /* Character values that are passed to RUN_MACRO are put
    */
    /* into their corresponding macro variables inside of quotation
    */
    /* marks. The quotation marks are part of the macro variable value.
    */
    /* The DEQUOTE function is called to remove the quotation marks.
    */
    %let dsname=%sysfunc(dequote(&dsname));
    data &dsname;
        %if %sysfunc(exist(&dsname)) %then %do;
            modify &dsname;

```

```

        %end;
        Name=&Name;
        WageCategory=&WageCategory;
        WageRate=&WageRate;
        output;
        stop;
    run;
%mend append_ds;

/* Call the APPEND_DS macro from function writeDataset in PROC
FCMP. */
proc fcmp outlib=sasuser.ds.functions;
    function writeDataset (DsName $, Name $, WageCategory $, WageRate);
        rc=run_macro('append_ds', DsName, Name, WageCategory, WageRate);
        return(rc);
    endsub;
run;

/* Use the DATA step to separate the salaries data set into four
separate */
/* departmental data sets (NAD, DDG, PPD, and
STD). */
data salaries;
    input Department $ Name $ WageCategory $ WageRate;
    datalines;
BAD Carol Salaried 20000
BAD Beth Salaried 5000
BAD Linda Salaried 7000
BAD Thomas Salaried 9000
BAD Lynne Hourly 230
DDG Jason Hourly 200
DDG Paul Salaried 4000
PPD Kevin Salaried 5500
PPD Amber Hourly 150
PPD Tina Salaried 13000
STD Helen Hourly 200
STD Jim Salaried 8000
;
run;

options cmplib=(sasuser.ds) pageno=1 nodate;
data _null_;
    set salaries;
    by Department;
    length dsName $ 64;
    retain dsName;
    if first.Department then do;
        dsName='work.' || trim(left(Department));
    end;
    rc=writeDataset(dsName, Name, WageCategory, wageRate);
run;

proc print data=work.BAD; run;
proc print data=work.DDG; run;
proc print data=work.PPD; run;
proc print data=work.STD; run;

```

Output 22.17 Results for Calling a DATA Step within a DATA Step

The SAS System				1
Obs	Name	Wage Category	Wage Rate	
1	Carol	Salaried	20000	
2	Beth	Salaried	5000	
3	Linda	Salaried	7000	
4	Thomas	Salaried	9000	
5	Lynne	Hourly	230	

The SAS System				2
Obs	Name	Wage Category	Wage Rate	
1	Jason	Hourly	200	
2	Paul	Salaried	4000	

The SAS System				3
Obs	Name	Wage Category	Wage Rate	
1	Kevin	Salaried	5500	
2	Amber	Hourly	150	
3	Tina	Salaried	13000	

The SAS System				4
Obs	Name	Wage Category	Wage Rate	
1	Helen	Hourly	200	
2	Jim	Salaried	8000	

RUN_SASFILE Function

Executes SAS code in a fileref that you specify.

Category: Calling SAS Code from within Functions

Syntax

```
rc = RUN_SASFILE ('fileref_name' <, variable-1, variable-2, ...>);
```

Required Arguments

rc

is 0 if the function is able to submit a request to execute the code that processes the SAS file. The return code indicates only that the call was attempted.

fileref_name

specifies the name of the SAS fileref that points to the SAS code.

Requirement *Fileref_name* must be a string enclosed in quotation marks or a character variable that contains the name of the SAS fileref.

Optional Argument

variable

specifies optional PROC FCMP variables that are set by macro variables of the same name. These arguments must be PROC FCMP double or character variables.

Before SAS executes the code that references the SAS file, the SAS macro variables are defined with the same name and value as the PROC FCMP variables. After execution, these macro variable values are copied back to the corresponding PROC FCMP variables.

Example

The following example is similar to the first example for RUN_MACRO except that RUN_SASFILE uses a SAS file instead of a predefined macro. This example assumes that test.sas(a, b, c) is located in the current directory.

```
/* test.sas(a,b,c) */
data _null_;
  call symput('p', &a * &b);
run;

/* Set a SAS fileref to point to the data set. */
filename myfileref "test.sas";

/* Set up a function in PROC FCMP and call it from the DATA step. */
proc fcmp outlib=sasuser.ds.functions;
  function subtract_sasfile(a, b);
    rc=run_sasfile('myfileref', a, b,
p);
    if rc=0 then return(p);
    else return(.);
  endsub;
run;
```

```

options cmplib=(sasuser.ds);
data _null_;
  a=5.3;
  b=0.7;
  p=.;
  p=subtract_sasfile(a, b);
  put p=;
run;

```

SOLVE Function

Computes implicit values of a function using the Gauss-Newton method.

Category: Compute Implicit Values

Note: This special purpose function is automatically provided by the FCMP procedure for convenience.

Syntax

answer = **SOLVE**('function-name', *options-array*, *expected-value*,
argument-1 <, *argument-2*, ...>);

Required Arguments

answer

specifies the value that is returned from the SOLVE function.

'function-name'

specifies the name of the function. Enclose *function-name* in quotation marks.

options-array

specifies an array of options to use with the SOLVE function. *Options-array* is used to control and monitor the root-finding process. *Options-array* can be a missing value (.), or it can have up to five of the following elements in the following order

initial-value

specifies the starting value for the implied value. The default for the first call is 0.001. If the same line of code is executed again, then *options-array* uses the previously found implied value.

absolute-criterion

specifies a value for convergence. The absolute value of the difference between the expected value and the predicted value must be less than the value of *absolute-criterion* for convergence.

Default 1.0e-12

relative-criterion

specifies a value for convergence. If the change in the computed implied value is less than the value of *relative-criterion*, then convergence is assumed.

Default 1.0e-6

maximum-iterations

specifies the maximum number of iterations to use to find the solution.

Default 100

solve-status

can be one of the following values:

- 0 successful.
- 1 could not decrease the error.
- 2 could not compute a change vector.
- 3 maximum number of iterations exceeded.
- 4 initial objective function is missing.

expected-value

specifies the expected value of the function of interest.

argument

specifies the arguments to pass to the function that is being minimized.

Details

The SOLVE function finds the value of the specified argument that makes the expression of the following form equal to zero.

```
expected-value - function-name
(argument-1, argument-2, ...)
```

You specify the argument of interest with a missing value (.), which appears in place of the argument in the parameter list that is shown above. If the SOLVE function finds the value, then the value that is returned for this function is the implied value.

Here is an example of an options array:

```
array opts[5] initial abconv relconv maxiter (.5 .001 1.0e-6 100);
```

These values refer to the preceding line of code.

- **initial** (initial-value)=.5
- **abconv** (absolute-criterion)=.001
- **relconv** (relative-criterion)=1.0e-6
- **maxiter** (maximum-iterations)=100

The solve status is the fifth element in the array. You can display this value by specifying `opts[5]` in the output list.

Examples

Example 1: Computing a Square Root Value

The following SOLVE function example computes a value of x that satisfies the equation $y=1/\sqrt{x}$. Note that you must first define functions and subroutines before you can use them in the SOLVE function. In this example, the function INVERSESQRT is first defined and then used in the SOLVE function.

```
options pageno=1 nodate ls=80 ps=64;

proc fcmp;
  /* define the function */
  function inversesqrt(x);
    return(1/sqrt(x));
  endsub;

  y=20;
  x=solve("inversesqrt", {., y, .});
  put x;
run;
```

Output 22.18 Results from Computing a Square Root Value

The SAS System	1
The FCMP Procedure	
0.0025	

Example 2: Calculating the Garman-Kohlhagen Implied Volatility

In this example, the subroutine GKIMPVOL calculates the Garman-Kohlhagen implied volatility for FX options by using the SOLVE function with the GARKHPRC function.

In this example, note the following:

- The options_array is SOLVOPTS, which requires an initial value.
- The expected value is the price of the FX option.
- The missing argument in the subroutine is the volatility (sigma).

```
proc fcmp;
  function garkhprc(type$, buysell$, amount, E, t, S, rd, rf, sig)
    kind=pricing label='FX option pricing';

    if buysell='Buy' then sign=1.;
    else do;
      if buysell='Sell' then sign=-1.;
      else sign=.;
    end;
end;
```

```

if type='Call' then
  garkhprc=sign*amount*(E+t+S+rd+rf+sig);
else do;
  if type='Put' then
    garkhprc=sign*amount*(E+t+S+rd+rf+sig);
  else garkhprc=.;
end;
return(garkhprc);
endsub;

subroutine gkimpvol(n, premium[*], typeflag[*], amt_lc[*],
                  strike[*], matdate[*], valudate, xrate,
                  rd, rf, sigma);
outargs sigma;

array solvopts[1] initial (0.20);
sigma=0;
do i=1 to n;
  maturity=(matdate[i] - valudate) / 365.25;
  stk_opt=1./strike[i];
  amt_opt=amt_lc[i] * strike[i];
  price=premium[i] * amt_lc[i];

  if typeflag[i] eq 0 then type="Call";
  if typeflag[i] eq 1 then type="Put";

  /* solve for volatility */
  sigma=sigma + solve("GARKHPRC", solvopts, price,
                    type, "Buy", amt_opt, stk_opt,
                    maturity, xrate, rd, rf, .);
end;
sigma=sigma / n;
endsub;
run;

```

Example 3: Calculating the Black-Scholes Implied Volatility

This SOLVE function example defines the function BLKSCH by using the built-in SAS function BLKSHCLPRC. The SOLVE function uses the BLKSCH function to calculate the Black-Scholes implied volatility of an option.

In this example, note the following:

- The options_array is OPTS.
- The missing argument in the function is the volatility (VOLTY).
- PUT statements are used to write the implied volatility (BSVOLTY), the initial value, and the solve status.

```

options pageno=1 nodate ls=80 ps=64;

proc fcmp;
  opt_price=5;
  strike=50;
  today='20jul2010'd;
  exp='21oct2010'd;
  eq_price=50;

```

```

    intrate=.05;
    time=exp - today;
    array opts[5] initial abconv relconv maxiter status
        (.5 .001 1.0e-6 100 -1);
    function blksch(strike, time, eq_price, intrate, volty);
        return(blkshclprc(strike, time/365.25,
            eq_price, intrate, volty));
    endsub;
    bsvolty=solve("blksch", opts, opt_price, strike,
        time, eq_price, intrate, .);

    put 'Option Implied Volatility:' bsvolty
        'Initial value: ' opts[1]
        'Solve status: ' opts[5];
run;

```

Output 22.19 Results of Calculating the Black-Scholes Implied Volatility

The SAS System	1
The FCMP Procedure	
Option Implied Volatility: 0.4687011859 Initial value: 0.5 Solve status: 0	

Note: SAS functions and external C functions cannot be used directly in the SOLVE function. They must be enclosed in a PROC FCMP function. In this example, the built-in SAS function BLKSHCLPRC is enclosed in the PROC FCMP function BLKSCH, and then BLKSCH is called in the SOLVE function.

WRITE_ARRAY Function

Writes data from a PROC FCMP array variable to a data set that can then be used by SAS programs, macros, and procedures.

Category: Array

Note: When SAS translates between an array and a data set, the array is indexed as [row, column].

Syntax

```
rc = WRITE_ARRAY(data_set_name, array_variable <, 'column_name_1',
'column_name_2', ...>);
```

Required Arguments

rc

is 0 if the function is able to successfully write the data set.

data_set_name
 specifies the name of the data set to which the array data is written.
Data_set_name must be a character literal or variable that contains the member name (libname.memname) of the data set to be created.

array_variable
 specifies the PROC FCMP array or matrix variable whose contents are written to *data_set_name*.

Optional Argument

column_name
 specifies optional names for the columns of the data set that are created.
 If specified, *column_name* must be a literal string enclosed in quotation marks. *Column_name* cannot be a PROC FCMP variable. If column names are not specified, the column name is the array name with a numeric suffix.

Examples

Example 1: Using the WRITE_ARRAY Function with a PROC FCMP Array Variable

This example uses the ARRAY statement and the WRITE_ARRAY function with PROC FCMP to write output to a data set.

```
options nodate pageno=1 ls=80 ps=64;

proc fcmp;
  array x[4,5] (11 12 13 14 15 21 22 23 24 25 31 32 33 34 35 41 42 43
44 45);
  rc=write_array('work.numbers', x);
run;

proc print data=work.numbers;
run;
```

Output 22.20 Results from Using the WRITE_ARRAY Function

The SAS System						1
Obs	x1	x2	x3	x4	x5	
1	11	12	13	14	15	
2	21	22	23	24	25	
3	31	32	33	34	35	
4	41	42	43	44	45	

Example 2: Using the WRITE_ARRAY Function to Specify Column Names

This example uses the optional *colN* variable to write column names to the data set.

```
options pageno=1 nodate ps=64 ls=80;

proc fcmp;
  array x[2,3] (1 2 3 4 5 6);
  rc=write_array('numbers2', x, 'col1', 'col2', 'col3');
run;

proc print data=numbers2;
run;
```

Output 22.21 Results from Using the WRITE_ARRAY Function to Specify Column Names

The SAS System				1
Obs	col1	col2	col3	
1	1	2	3	
2	4	5	6	

FEDSQL Procedure

Overview: FEDSQL Procedure	793
What Does the FEDSQL Procedure Do?	794
Concepts: FEDSQL Procedure	795
Benefits of FedSQL	795
Data Source Support	796
Syntax: FEDSQL Procedure	797
PROC FEDSQL Statement	797
QUIT Statement	806
Usage: FEDSQL Procedure	806
Connecting to SAS and DBMS Data Sources	806
Connecting to the CAS Server	808
FedSQL Functionality with SAS and DBMS Data Sources	809
FedSQL Functionality in SAS Cloud Analytic Services	810
Applying FedSQL Table Options	810
Macro Variables	811
Passwords	812
Encryption	813
FedSQL Data Type Support for SAS Data Sets	813
FedSQL Data Type Support for CAS Tables	815
PROC FEDSQL and PROC SQL: A Comparison	816
Examples: FEDSQL Procedure	816
Example 1: Creating a SAS Data Set	816
Example 2: Creating a New Table from Existing Tables	818
Example 3: Querying Data Using a Correlated Subquery	820
Example 4: Creating and Using an Index to Perform a Join	822
Example 5: Using a DS2 Package in an Expression	823
Example 6: Querying Data in CAS	826
Example 7: Explicitly Loading and Joining Tables in CAS	828
Example 8: Joining Tables from Multiple CAS Libraries	833

Overview: FEDSQL Procedure

What Does the FEDSQL Procedure Do?

The FEDSQL procedure enables you to submit SAS FedSQL language statements. Using FedSQL statements, you can create, query, and join data. You can submit FedSQL statements to SAS and to third-party DBMS. Or, you can submit FedSQL statements that query data in CAS.

The FedSQL language is the SAS implementation of the ANSI SQL:1999 core standard. It provides support for extended data types, such as DECIMAL, INTEGER, and VARCHAR, and other ANSI 1999 core compliance features and proprietary extensions.

FedSQL provides threaded data access technology that brings a scalable, high-performance way to access, manage, and share relational data in multiple data sources. When possible, FedSQL queries are optimized with multi-threaded algorithms to resolve large-scale operations.

FedSQL is a vendor-neutral SQL dialect. For applications, FedSQL provides a common SQL syntax across all of the data sources that it supports. Applications can submit the same FedSQL queries to all FedSQL data sources instead of having to submit queries in the SQL dialect that is specific to the data source.

Note: Unlike most other SAS procedures, the FEDSQL procedure does not use the CAS engine to connect to CAS. In order to submit FedSQL requests to the CAS server, you must specify a caslib that was assigned using the CASLIB statement. For more information, see [“Connecting to the CAS Server” on page 808](#).

The FedSQL statements that are supported in CAS are a subset of the statements that are available for SAS and DBMS data sources. For more information, see:

- [“FedSQL Functionality with SAS and DBMS Data Sources” on page 809](#)
- [“FedSQL Functionality in SAS Cloud Analytic Services” on page 810](#)

Concepts: FEDSQL Procedure

Benefits of FedSQL

FedSQL provides the following features that are not provided in the SAS SQL procedure.

- FedSQL conforms to the SQL 1999 ANSI standard. This conformance allows it to process queries in its own language as well as the native languages of other DBMSs that conform to the ANSI 1999 standard.
- FedSQL supports many more data types than previous SAS SQL implementations. Traditional DBMS access through SAS/ACCESS LIBNAME engines translates target DBMS data types to and from SAS legacy data types, which are SAS numeric and SAS character. When FedSQL connects to a DBMS, the language matches or translates the target data source's definition to the [FedSQL data types](#), as appropriate, which allows greater precision.

When FedSQL connects to a DBMS from CAS, the language translates the data source's definition to native CAS data types.

- FedSQL supports federated queries. A federated query enables you to connect to multiple data sources, access and perform operations against that data, and return a single result set. The output data can be created in any of the connected data sources. This enables you to store data in the data source that most closely meets the needs of your application.

(FedSQL output tables in CAS are in-memory tables. You can use other CAS functionality to persist data to caslib data sources.)

- FedSQL provides implicit SQL pass-through for ANSI-compliant functionality. There is no need to submit requests in the native SQL of each data source to get the benefits of native processing. FedSQL implicit pass-through improves query response time and enhances security. For more information, see [“FedSQL Implicit Pass-Through Facility” in SAS FedSQL Language Reference](#).
- For those who choose it, the FedSQL explicit pass-through facility enables you to connect to a data source and send native SQL statements to that data source for execution.

CAS connections accept native SQL syntax through the use of the CONNECTION TO component of the FedSQL SELECT statement's FROM clause.

Data Source Support

The following data sources are supported with the FedSQL procedure.

- Amazon Redshift
- CAS tables
- DB2 in the UNIX operating environment
- Google BigQuery
- Greenplum
- Hadoop (Hive)
- Impala
- JDBC-compliant databases
- Microsoft SQL Server
- MongoDB
- MySQL
- Netezza
- ODBC-compliant databases
- Oracle
- PostgreSQL
- Salesforce
- SAP
- SAP HANA
- SAP IQ
- SAS data sets
- SAS Scalable Performance Data Engine data sets
- SAS Scalable Performance Data Server tables
- SingleStore
- Snowflake
- Spark
- Teradata
- Vertica
- Yellowbrick

You must have appropriate SAS/ACCESS software configured in order to access DBMS data sources. You must have appropriate SAS data connector software configured to load data on the CAS server. For information to establish a data source connection with PROC FEDSQL, see:

- “Connecting to SAS and DBMS Data Sources”
- “Connecting to the CAS Server”

Syntax: FEDSQL Procedure

```
PROC FEDSQL <connection-options><processing-options>;
...FedSQL statements
QUIT;
```

Statement	Task	Example
PROC FEDSQL	Specify that the subsequent input is FedSQL language statements.	Ex. 1, Ex. 2, Ex. 3, Ex. 4, Ex. 5, Ex. 6, Ex. 7, Ex. 8
QUIT	Stop the execution of the FEDSQL procedure.	Ex. 1, Ex. 2, Ex. 3, Ex. 4, Ex. 5, Ex. 6, Ex. 7, Ex. 8

PROC FEDSQL Statement

Specifies that the subsequent input is FedSQL language statements.

Requirement: Follow the PROC FEDSQL statement with one or more FedSQL language statements. See [SAS FedSQL Language Reference](#) for information about FedSQL statements that are supported in a SAS library. See “[FedSQL Functionality in SAS Cloud Analytic Services](#)” on page 810 for information about FedSQL statements that are supported in a CAS session.

Interactions: When creating tables with PROC FEDSQL, note that by default, you cannot overwrite an existing table. You must destroy the table with the DROP TABLE statement first. If you specify the DROP TABLE statement in the same procedure execution as the CREATE TABLE statement, you must specify the FORCE statement option. In CAS, you can overwrite an existing table by specifying the REPLACE= table option in the CREATE TABLE statement. For more information, see “[REPLACE= Table Option](#)” in [SAS Viya Platform: FedSQL Programming for SAS Cloud Analytic Services](#).

By default when accessing SAS or DBMS data sources, the FEDSQL procedure processes nonexistent values in CHAR(n) and DOUBLE columns as SAS missing values. To request that nonexistent values are processed as ANSI SQL null values, specify the ANSIMODE option. ANSIMODE is not supported in CAS.

Supports:	Threaded processing
Note:	As of September 2023, Microsoft Azure Active Directory (Azure AD) was renamed to Microsoft Entra ID.
Examples:	“Example 1: Creating a SAS Data Set” on page 816 “Example 2: Creating a New Table from Existing Tables” on page 818 “Example 3: Querying Data Using a Correlated Subquery” on page 820 “Example 4: Creating and Using an Index to Perform a Join” on page 822 “Example 5: Using a DS2 Package in an Expression” on page 823 “Example 6: Querying Data in CAS” on page 826 “Example 7: Explicitly Loading and Joining Tables in CAS” on page 828 “Example 8: Joining Tables from Multiple CAS Libraries” on page 833

Syntax

PROC FEDSQL *<connection-options><processing-options>;*

Summary of Optional Arguments

Connection

LIBS=*libref* | (*libref1libref2...librefn*)

restricts the data source connection to the specified libref or librefs, and the default library.

SESSREF=*session-reference*

specifies to run the FedSQL statements in a CAS session. The CAS session is identified by its session name.

SESSUUID=*"session-uuid"*

specifies to run the FedSQL statements in a CAS session. The CAS session is identified by its universally unique identifier (UUID).

Execution Control

CNTL=(*instruction(s)*)

specifies optional control instructions for FedSQL query planning and query execution in CAS.

General Processing

ANSIMODE

specifies that nonexistent values in CHAR and DOUBLE columns are processed as ANSI SQL null values.

AUTOCOMMIT

NOAUTOCOMMIT

for use with DBMS that support transactions only, specifies whether updates are automatically saved to a table after a default number of rows are updated, or whether rollback is available.

ERRORSTOP

NOERRORSTOP

specifies whether the procedure stops executing if it encounters an error.

LABEL**NOLABEL**

specifies whether to use the column label or the column name as the column heading.

MEMSIZE=*n* | *n*M | *n*G

specifies a limit for the amount of memory that is used for an underlying query (such as a SELECT statement), so that allocated memory is available to support other PROC FEDSQL operations.

NOPRINT

suppresses the normal display of results.

NUMBER

specifies to include a column named Row, which is the row (observation) number of the data as the rows are retrieved.

STIMER

specifies to write a subset of system performance statistics, such as time-elapsed statistics, to the SAS log.

XCODE=ERROR | WARNING | IGNORE

controls the behavior of the SAS session when an NLS transcoding failure occurs.

Query Planning**_METHOD**

prints a brief text description of the FedSQL query plan.

_POSTOPTPLAN

prints an XML tree illustrating the FedSQL query plan.

EXEC**NOEXEC**

specifies whether a statement should be executed after its syntax is checked for accuracy.

Optional Arguments**_METHOD**

prints a brief text description of the FedSQL query plan. A FedSQL query is broken into stages. Each stage of execution requires a standalone SQL query. This option generates a brief text description of the nodes and stages in the query plan. The information is written to the SAS log.

Notes **_METHOD** is supported on the CAS server only.

The **_METHOD** option does not return the stage query or the number of threads used by a FedSQL request. This information can be obtained with the **showStages CNTL=** option. For more information, see [“SHOWSTAGES” on page 802](#).

See FedSQL provides several ways to return information about query planning and query execution in CAS. For a comparison of the methods,

see [“Using FedSQL Query Plan Options: Examples” in SAS Viya Platform: FedSQL Programming for SAS Cloud Analytic Services](#)

_POSTOPTPLAN

prints an XML tree illustrating the FedSQL query plan. A FedSQL query is broken into stages. Each stage of execution requires a standalone SQL query. This option generates an XML tree that illustrates each stage of the FedSQL query plan and the results from each execution stage. The information is written to the SAS log.

Notes _POSTOPTPLAN is supported on the CAS server only.

The XML tree can be very long. You might want to use the _METHOD option instead.

ANSIMODE

specifies that nonexistent values in CHAR and DOUBLE columns are processed as ANSI SQL null values. By default, PROC FEDSQL processes nonexistent values in CHAR, DOUBLE, and VARCHAR columns as missing values. This is how SAS processes nonexistent values. The ANSIMODE option specifies to process nonexistent values in CHAR, DOUBLE, and VARCHAR columns as ANSI SQL null values. It is important to understand the differences, or data can be lost.

Default SAS mode

Restriction ANSIMODE is not supported on the CAS server.

Note All other data types use ANSI NULL semantics all of the time.

See [“How FedSQL Processes Nulls and SAS Missing Values” in SAS FedSQL Language Reference](#)

AUTOCOMMIT | NOAUTOCOMMIT

for use with DBMS that support transactions only, specifies whether updates are automatically saved to a table after a default number of rows are updated, or whether rollback is available.

Default AUTOCOMMIT

Restrictions For data sources that do not support transactions, specifying NOAUTOCOMMIT returns an error. The data sources that support transactions include Greenplum, Microsoft SQL Server, MySQL, databases that are compliant with ODBC, Oracle, SAP HANA, Teradata, and Vertica.

NOAUTOCOMMIT is not supported on the CAS server.

CNTL=(instruction(s))

specifies optional control instructions for FedSQL query planning and query execution in CAS. Multiple instructions are allowed inside of the parentheses. Separate each instruction with a space. The following instructions are supported:

DISABLEPASSTHROUGH

disables implicit FedSQL pass-through in CAS.

FedSQL attempts to use implicit pass-through for all SQL data sources by default. In order for a FedSQL request to be eligible for implicit pass-through in CAS, all tables must exist in the same caslib and the tables cannot have already been loaded into the CAS session. For other requirements, see [“FedSQL Implicit Pass-Through Facility in CAS” in SAS Viya Platform: FedSQL Programming for SAS Cloud Analytic Services](#). This option can save processing time for FedSQL requests that contain functions that are specific to SAS, or whose tables have already been loaded into the CAS session.

DYNAMICCARDINALITY

instructs the FedSQL query planner to perform cardinality estimations before selecting a query plan.

The FedSQL query planner does not perform cardinality estimations before selecting a query plan by default. Cardinality estimation can improve the accuracy of selectivity estimates for join conditions and WHERE clause predicates, which in turn can lead to better join-order decisions and faster query execution times for some queries. However, dynamic cardinality does not help all queries and adds overhead to the query planning process.

See [“Using the Dynamic Cardinality Instruction” in SAS Viya Platform: FedSQL Programming for SAS Cloud Analytic Services](#)

OPTIMIZEVARBINARYPRECISION

optimizes VARBINARY precision by using a precision that is appropriate to the actual data, instead of the precision declared for the VARBINARY columns.

The precision is optimized both when reading from a table and when creating a new table from a result set. The greatest benefits from this option are achieved when the declared precision is far larger than the precision of the actual data. Then, this option can improve performance, reduce memory footprint, and create VARBINARY columns in new tables with the needed size rather than propagating the precision from the source table.

OPTIMIZEVARCHARPRECISION

optimizes VARCHAR precision by using a precision that is appropriate to the actual data, instead of the precision declared for the VARCHAR columns.

The precision is optimized both when reading from a table and when creating a new table from a result set. The greatest benefits from this option are achieved when the declared precision is far larger than the precision of the actual data. Then, this option can improve performance, reduce memory footprint, and create VARCHAR columns in new tables with the needed size rather than propagating the precision from the source table.

PRESERVEJOINORDER

joins tables in the specified order instead of an order chosen by the FedSQL query optimizer.

REQUIREFULLPASSTHROUGH

stops processing the FedSQL request when implicit pass-through of the full query cannot be achieved.

SHOWSTAGES

writes query execution details to the SAS log.

This includes the information returned by the `_METHOD` option. In addition, `showStages` prints the stage query, the number of SQL threads used by each stage, and the following query execution details:

- Time for each intermediate stage
- Number of output rows from each intermediate stage
- Elapsed time for each stage and for the entire action
- Whether a table was replicated to all workers, or auto-partitioned.

The information is gathered when the query is executed and is written to the client log. Do not specify `NOEXEC` with `showStages`. Execution details cannot be printed if the query is not executed.

See [FedSQL provides several ways to return information about query planning and query execution in CAS. For a comparison of the methods, see “Displaying the FedSQL Query Plan for a Request” in *SAS Viya Platform: FedSQL Programming for SAS Cloud Analytic Services*.](#)

Note `CNTL=` is supported on the CAS server only.

See [“Optimize FedSQL Performance by Modifying the FedSQL Query Plan” in *SAS Viya Platform: FedSQL Programming for SAS Cloud Analytic Services*](#)

ERRORSTOP | NOERRORSTOP

specifies whether the procedure stops executing if it encounters an error. `ERRORSTOP` instructs the procedure to stop executing the statements but to continue checking the syntax after it has encountered an error. `NOERRORSTOP` instructs the procedure to execute the statements and to continue checking the syntax after an error occurs.

Default `NOERRORSTOP` in an interactive SAS session; `ERRORSTOP` in a batch or noninteractive session

Restriction `ERRORSTOP` has an effect only when SAS is running in the batch or noninteractive execution mode.

Tip `NOERRORSTOP` is useful if you want a batch job to continue executing SQL procedure statements after an error is encountered.

EXEC | NOEXEC

specifies whether a statement should be executed after its syntax is checked for accuracy. `EXEC` specifies to execute the statement. `NOEXEC` specifies not to execute the statement.

Default `EXEC`

Tip NOEXEC is useful if you want to check the syntax of your FedSQL statements without executing the statements.

LABEL | NOLABEL

specifies whether to use the column label or the column name as the column heading. If a column does not have a label, the procedure uses the column's name as the column heading.

Default LABEL

Interaction When a column alias is used, the column alias supersedes the label or column name as the column heading.

LIBS=libref | (libref1libref2...librefn)

restricts the data source connection to the specified libref or librefs, and the default library. LIBS= is useful when multiple LIBNAME statements are defined in a SAS session, to limit the scope of the FedSQL request to the LIBNAME statement(s) to which the request applies. It can also avoid duplicate catalog errors when connecting to data sources that support native catalogs, such as Netezza.

When more than one libref is specified, enclose the librefs within parentheses. Separate each libref with a space.

Alias LIBNAMES=

Restriction LIBS= is not supported on the CAS server.

Interactions Specifying LIBS= does not preclude the need to specify librefs in your FedSQL statements, even when only one libref is specified in the LIBS= option. If you do not specify a libref with your table names, the default library is used. For more information, see [“Connecting to SAS and DBMS Data Sources” on page 806](#).

If both LIBS= and SESSREF= (or SESSUUID=) are specified in the procedure statement, SESSREF= is applied and LIBS= is ignored.

If you specify LIBS= on the procedure statement more than once, only the libref(s) in the last instance of the procedure option are applied.

Tip If you want to know how LIBS= affects library assignments, set the MSGLEVEL=i system option before running a PROC FEDSQL request with LIBS=. The option will produce Include and Ignore messages for each of the LIBNAME statements that are processed in the procedure request.

Example The following PROC FEDSQL procedure statement specifies to create a data source connection that uses only two of four librefs that are assigned in the SAS session and the default library:

```
proc fedsql libs=(mylib3 mylib4);
```

MEMSIZE=n | nM | nG

specifies a limit for the amount of memory that is used for an underlying query (such as a SELECT statement), so that allocated memory is available to support

other PROC FEDSQL operations. Specify the memory limit in multiples of 1 (bytes); 1,048,576 (megabytes); or 1,073,741,824 (gigabytes). For example, the value 23M specifies 24,117,248 bytes of memory. The value 16G specifies 17,179,869,184 bytes of memory.

Default The procedure optimizes the setting based on the amount of memory on the host.

Note On the CAS server, MEMSIZE= specifies the memory for a single CAS worker.

Tip Generally, specifying a memory limit is not necessary unless FedSQL reports a memory problem error.

NOPRINT

suppresses the normal display of results.

Interaction NOPRINT affects the value of the SQLOBS automatic macro variable, which contains the number of rows that are executed by a statement.

NUMBER

specifies to include a column named Row, which is the row (observation) number of the data as the rows are retrieved.

Default No row numbers.

SESSREF=*session-reference*

specifies to run the FedSQL statements in a CAS session. The CAS session is identified by its session name.

Requirements A CAS server must be configured for your system.

The CAS session must have been previously established by using the CAS statement. If the specified CAS session does not exist, the procedure terminates.

Interactions Use SESSREF= or SESSUUID= to connect to the CAS session. If both options are specified, the last option in the procedure statement is applied.

If both SESSREF= and LIBS= are specified on the procedure statement, SESSREF= is applied and the other option is ignored.

See [“Connecting to the CAS Server” on page 808](#)

[“FedSQL Functionality in SAS Cloud Analytic Services” on page 810](#)

[“Example 7: Explicitly Loading and Joining Tables in CAS” on page 828](#)

SESSUUID=*“session-uuid”*

specifies to run the FedSQL statements in a CAS session. The CAS session is identified by its universally unique identifier (UUID).

Requirements	A CAS server must be configured for your system. The CAS session must have been previously established by using the CAS statement. The CAS statement generates a UUID value. If the specified CAS session does not exist, the procedure terminates.
Interactions	Use SESSREF= or SESSUUID= to connect to the CAS session. If both options are specified, the last option in the procedure statement is applied. If both SESSUUID= and LIBS= are specified in the procedure statement, SESSUUID= is applied and the other option is ignored.
See	“Connecting to the CAS Server” on page 808 “FedSQL Functionality in SAS Cloud Analytic Services” on page 810
Example	Here is a PROC FEDSQL procedure statement that specifies SESSUUID=: <pre>proc fedsql sessuuid="76904741-fb09-554d-a8de-6cbce2a0e0e5";</pre> The UUID value can be enclosed in single or double quotation marks.

STIMER

specifies to write a subset of system performance statistics, such as time-elapsed statistics, to the SAS log. When STIMER is in effect, the procedure writes to the SAS log a list of computer resources used for each step and the entire SAS session.

Default	No performance statistics are written to the SAS log.
Interaction	If the SAS system option FULLSTIMER is in effect, the complete list of computer resources is written to the SAS log.

XCODE=ERROR | WARNING | IGNORE

controls the behavior of the SAS session when an NLS transcoding failure occurs. Transcoding failures can occur during row input or output operations, or during string assignment. Transcoding is the process of converting character data from one encoding to another encoding.

ERROR

specifies that a run-time error occurs, which causes row processing to halt. An error message is written to the SAS log. This is the default behavior.

WARNING

specifies that the incompatible character is set to a substitution character. A warning message is written to the SAS log.

IGNORE

specifies that the incompatible character is set to a substitution character. No messages are written to the SAS log.

Default	ERROR
---------	-------

QUIT Statement

Stops the execution of the FEDSQL procedure.

Interaction: PROC FEDSQL does not recognize step boundaries. If you submit a DATA step or another procedure step without first specifying the QUIT statement, the FedSQL language issues a syntax error and PROC FEDSQL continues processing. The QUIT statement is required to stop the FEDSQL procedure.

Syntax

QUIT;

Details

When the FEDSQL procedure reaches the QUIT statement, all resources allocated by the procedure are released. You can no longer execute FedSQL language statements without invoking the procedure again. However, the connection to the data source server is not lost, because that connection was made through the LIBNAME statement. As a result, any subsequent invocation of the procedure that uses the same libref executes almost instantaneously because the LIBNAME engine is already connected to the server.

Usage: FEDSQL Procedure

Connecting to SAS and DBMS Data Sources

You connect PROC FEDSQL to a SAS or DBMS data source by specifying a libref.

- 1 First, submit LIBNAME statements that specify the SAS and SAS/ACCESS engines for the data sources that you want to connect to. For information to define a LIBNAME statement for a given data source, see these resources:

SAS data sets

[SAS Global Statements: Reference](#)

Relational DBMS data sources

[SAS/ACCESS for Relational Databases: Reference](#)

MongoDB and Salesforce

[SAS/ACCESS for Nonrelational Databases: Reference](#)

SPD Engine data sets

[SAS Scalable Performance Data Engine: Reference](#)

SPD Server tables

[SAS Scalable Performance Data Server: User's Guide](#)

Note: The library path in the LIBNAME statement must be known to the SAS Compute Server.

Note: Do not use a CAS engine libref with PROC FEDSQL. See [“Connecting to the CAS Server” on page 808](#) for instructions to direct FedSQL statements to the CAS server.

- 2 In your FedSQL language statements, use a name in the form *libref.table-name* to refer to tables. The libref tells the FedSQL language where to create or locate a table.
- 3 Then, submit PROC FEDSQL.

This example illustrates how PROC FEDSQL accesses a data source by using the attributes of a previously assigned libref.

```
libname mylib 'library-path';

proc fedsql;
  create table mylib.mytable (col1 char(5), col2 double);
  insert into mylib.mytable values ('one', 1);
  insert into mylib.mytable values ('two', 2);
quit;
```

In this example, the LIBNAME statement assigns libref Mylib and then specifies the physical location for a SAS data set. The FedSQL statements use a name in the form *libref.table-name* to reference the table.

When you submit a program with the FEDSQL procedure, the procedure builds a data source connection string internally that includes all active librefs. The procedure then sends this internal connection string to the FedSQL program. The FedSQL language uses the librefs in the program to locate properties for the specific data source connection from within this internal connection string.

This internal connection string primarily includes libref attributes for data source connection (such as physical location and authentication). The string might not include libref attributes that define behavior. For example, if a previously submitted LIBNAME statement for the SAS V9 engine specifies that SAS data sets are to be compressed, the compression attribute is not used by the procedure. In order to request compression, you must specify the COMPRESS= table option in your FedSQL program.

Not all LIBNAME statement options have a corresponding table option. Some LIBNAME behavior attributes are used by the procedure.

TIP You can determine which LIBNAME options are used by the procedure by setting the MSGLEVEL=i system option before submitting a LIBNAME statement.

The procedure's default connection behavior is as follows. When a table is referenced in a FedSQL program without a libref, or when an engine name is omitted from a LIBNAME statement, the default SAS V9 engine is used. Tables that are referenced without a libref are created in the SAS Work library, or in the User library, if a User library is defined.

Connecting to the CAS Server

To submit FedSQL statements to the CAS server:

- 1 Submit the CAS statement to establish a CAS session in the SAS session.
- 2 Assign a caslib (or use a pre-defined caslib) for the data source that you want to interact with in the session. You can assign a caslib and list the caslibs that are available to your CAS session by using the CASLIB statement. A caslib uses a SAS Data Connector (or SAS Data Connector Accelerator) to access data. For information about SAS Data Connectors, see [SAS Cloud Analytic Services: User's Guide](#).
- 3 Specify the SESSREF= (or SESSUUID=) procedure option with the CAS session name or session identifier in the PROC FEDSQL statement. When SESSREF= (or SESSUUID=) is specified, both the default connection mechanism and the LIBS= procedure option are ignored. Instead, the procedure connects to the specified CAS session.
- 4 You can now reference tables in the caslib in your FedSQL statements. Tables that are in the active caslib can be referenced simply by name. Use a two-level name in the form *caslib.table-name* to access tables that are not in the active caslib.

The following example illustrates a PROC FEDSQL request to the CAS server.

```
cas; 1

caslib castera desc='Teradata Caslib' 2
datasource=(srctype='teradata'
username='myname'
password='mypw'
server='testserver',
db='testdb');

proc fedsql sessref=casauto; 3
  create table newtable as
  select * from employees; 4
```

```
quit;
```

- 1 The CAS statement establishes a CAS session. The CAS session has the default name Casauto.
- 2 The CASLIB statement assigns the caslib Castera. Castera becomes the active caslib.
- 3 The PROC FEDSQL statement specifies the SESSREF= procedure option with the CAS session name Casauto.
- 4 The FedSQL CREATE TABLE statement specifies to select all columns in table Employees and create a table named Newtable in the active caslib (Castera). There is no need to qualify the names of the tables that exist in the active caslib with a caslib. In the active caslib, use of the caslib in the table name is optional.

For FedSQL, there are benefits to referencing tables by name only. Referencing a table name in the active caslib without a caslib can improve the likelihood of FedSQL implicit pass-through (IP) in CAS. There are other requirements. For more information, see [“FedSQL Implicit Pass-Through Facility in CAS” in SAS Viya Platform: FedSQL Programming for SAS Cloud Analytic Services](#).

You can set or change the active caslib for a CAS session explicitly by using the [CAS statement](#) with the [CASLIB= session option](#). See [“Example 8: Joining Tables from Multiple CAS Libraries” on page 833](#) for an example of how the CASLIB= session option is used.

For more information about the CAS statement, CASLIB statement, and CASUTIL procedure, see [SAS Cloud Analytic Services: User's Guide](#).

The previous example uses a data connector to automatically load data into CAS. Data can also be loaded into CAS from path-based locations. For an example that explicitly loads data into the CAS session for FedSQL to process, see [“Example 7: Explicitly Loading and Joining Tables in CAS” on page 828](#).

The CAS tables that you create and work with from PROC FEDSQL are in-memory session tables. That is, the tables are available for the duration of the CAS session and are accessible only to the current session. PROC FEDSQL does not provide a way to persist a table to a data source or to share the table with other CAS sessions. To persist or share a CAS output table, use the CASUTIL procedure.

Although CAS tables are in-memory tables, PROC FEDSQL will not overwrite an existing table of the same name. Specify the REPLACE= table option to overwrite an existing table with a replacement table. Or, use the DROP TABLE statement to remove the initial table before creating the replacement table.

FedSQL Functionality with SAS and DBMS Data Sources

All FedSQL language statements are supported in requests to SAS and DBMS data sources. See [SAS FedSQL Language Reference](#) for information about the FedSQL statements.

Note that not all statements are supported for all data sources. [Data type](#) support and [table option](#) support is also data source-specific. For an example, see “[FedSQL Data Type Support for SAS Data Sets](#)” on page 813.

FedSQL Functionality in SAS Cloud Analytic Services

When you specify the `SESSREF=` (or `SESSUID=`) option and a CAS session identifier in the PROC FEDSQL statement, your FedSQL statements are directed to the CAS server, where they are processed by the `fedSql.execDirect` action. For more information about this action, see the `fedSql.execDirect` action in the *SAS Viya Platform: System Programming Guide*. SAS Cloud Analytic Services is an alternative environment for processing FedSQL requests.

The `fedSql.execDirect` action operates on CAS tables that either were pre-loaded into CAS or are automatically loaded upon reference.

The FedSQL statements that are available in CAS data are a subset of those that are available for SAS and DBMS data. The following FedSQL statements are supported in CAS:

- CREATE TABLE, with the AS query expression
- SELECT
- DROP TABLE

The following SELECT statement features are not supported in CAS:

- EXCEPT and INTERSECT SET operations (UNION is supported)
- correlated subqueries
- IN, ANY, and ALL subqueries
- dictionary queries
- views
- DS2 user-defined functions (also known as DS2 package expressions).

For the best results when submitting FedSQL statement text to the CAS server, indent the first line of each FedSQL statement and each SELECT statement clause (except WHERE) in the FedSQL program.

For more information about FedSQL functionality on the CAS server, see [SAS Viya Platform: FedSQL Programming for SAS Cloud Analytic Services](#).

Applying FedSQL Table Options

When you access a data source with PROC FEDSQL, you can apply FedSQL table options in the subsequent FedSQL statements. A table option specifies actions

that enable you to perform operations on a table such as assigning buffer page size or specifying passwords.

FedSQL table options are used to apply options for FedSQL language processing, like SAS data set options apply options for DATA step processing. SAS data set options cannot be used with the FedSQL language. For example, the following code applies a table option to a SAS data set in order to specify the size of a permanent buffer page for the new table:

```
libname myfiles base 'library-path';

proc fedsql;
  create table myfiles.table1 {options bufsize=16k}(x double) ;
  insert into myfiles.table1 values (1.0);
  insert into myfiles.table1 values (2.0);
  insert into myfiles.table1 values (3.0);
quit;
```

FedSQL supports different table options on the CAS server than it does on the SAS Compute Server. For a list of table options that are supported in CAS, see [SAS Viya Platform: FedSQL Programming for SAS Cloud Analytic Services](#). For a list of table options that are supported with SAS and SAS/ACCESS data sources, see [SAS FedSQL Language Reference](#).

Macro Variables

This information does not apply to FedSQL use on the CAS server.

Using Macro Variables in a Literal String

Macro variables enable you to dynamically modify text in a program through symbolic substitution. When you reference a macro variable in a program, the macro processor replaces the reference with the value of the specified macro variable.

With PROC FEDSQL, you can use a macro variable on a subsequent FedSQL statement. However, if a macro variable occurs within a literal string, you cannot enclose the string in double quotation marks. The macro processor uses double quotation marks to resolve macro variable references. FedSQL considers a string enclosed in double quotation marks to be a delimited (case sensitive) identifier such as a table or column name.

To reference a macro variable in a literal string, use the SAS macro function %TSLIT. %TSLIT overrides the need for double quotation marks around the literal string and puts single quotation marks around the input value. For example, the following statement includes the %TSLIT function to specify the &SYSHOSTNAME macro variable, which returns the host name of the computer on which it is executed:

```
select %tslit(&syshostname);
```

The %TSLIT macro function is stored in the default autocall macro library. For more information, see [“Referencing a Macro Variable in a Delimited Identifier” in SAS FedSQL Language Reference](#).

Using Macro Variables Set by the Procedure

PROC FEDSQL sets up macro variables with certain values after it executes each statement. These macro variables can be tested inside a macro to determine whether to continue executing the PROC step. After each statement has executed, the following macro variable is updated with these values:

SQLRC

contains the following status values that indicate the success of the PROC FEDSQL statement:

- 0
PROC statement completed successfully with no errors.
- 4
PROC statement encountered a situation for which it issued a warning. The statement continued to execute.
- 8
PROC statement encountered an error. The statement stopped execution at this point.
- 16
PROC statement encountered a run-time error. For example, this error code is used when a subquery (that can return only a single value) evaluates more than one row. These errors can be detected only during run time.

Passwords

This information does not apply to FedSQL use on the CAS server.

SAS software enables you to restrict access to SAS data sets and SPD Engine data sets by assigning SAS passwords to the files. You can specify three levels of protection: read, write, and alter.

Note: A SAS password does not control access to a SAS file beyond SAS. You should use the operating system-supplied utilities and file-system security controls to control access to SAS files outside SAS. For more information about SAS passwords, see [SAS FedSQL Language Reference](#).

With PROC FEDSQL, you can assign or specify a password for a data source using the FedSQL table options ALTER=, PW=, READ=, and WRITE=. For example, the following code applies the FedSQL table option PW= in order to assign READ, WRITE, and ALTER passwords to a SAS data set:

```
libname myfiles base 'library-path';

proc fedsql;
  create table myfiles.table1 {options pw=luke}(x double) ;
  insert into myfiles.table1 values (1.0);
  insert into myfiles.table1 values (2.0);
  insert into myfiles.table1 values (3.0);
quit;
```

This code shows how to specify a table option to read a data set:

```
select * from myfiles.table1 {options pw=luke};
```

CAS tables do not support SAS passwords. You cannot assign a password for a CAS table. When accessing password-protected data from CAS, passwords are specified in the CAS language element that is used to access the data. For example, passwords are supported in the CASUTIL procedure, which loads data into CAS, and in the CASLIB statement. For more information, see the SAS Data Connector documentation in [SAS Cloud Analytic Services: User's Guide](#).

Encryption

SAS software enables you to encrypt the contents of a SAS data set, SPD Engine data set, and SPD Server table. SAS supports SAS proprietary encryption and AES encryption.

SAS proprietary encryption is done by specifying the ENCRYPT= table option with the PW= or READ= table option. A data set or table encrypted with SAS proprietary encryption must be decrypted by specifying the PW= or READ= table option with the appropriate password.

AES encryption is performed by specifying the ENCRYPT= table option with the ENCRYPTKEY= table option. A data set or table encrypted with AES encryption is later decrypted by specifying the ENCRYPTKEY= table option with the appropriate key value. SAS supports two levels of AES encryption: AES and AES2. AES2 meets newer and more secure encryption standards than AES. For more information, see “ENCRYPT= Table Option” in [SAS FedSQL Language Reference](#).

FedSQL currently does not support the encryption attribute for CAS tables. When accessing SAS encrypted and AES-encrypted data sets from CAS, passwords and encryption keys are specified in the CAS language element that is used to access the data. For example, passwords and encryption keys are supported in the CASUTIL procedure, which loads data into CAS, and in the CASLIB statement. For more information, see the SAS Data Connector documentation in [SAS Cloud Analytic Services: User's Guide](#).

FedSQL Data Type Support for SAS Data Sets

In PROC FEDSQL, when you submit FedSQL statements, all FedSQL language data types are supported. For information about the FedSQL data types, see [SAS](#)

[FedSQL Language Reference](#). However, when reading and creating data, the FedSQL data types are translated to and from the data types of the target data source. The following table lists the FedSQL data types and how they are translated to and from SAS data types:

Table 23.1 FedSQL Data Type Translation for SAS Data Sets

FedSQL Data Type	SAS Data Type	Description
BIGINT	SAS numeric	Because a SAS numeric is a DOUBLE, which is an approximate numeric data type rather than an exact numeric data type, there is potential for loss of precision.
BINARY(<i>n</i>)	SAS character	Applies the SAS format \$ <i>n</i> .
CHAR(<i>n</i>)	SAS character	Applies the SAS format \$ <i>n</i> .
DATE	SAS numeric ¹	Applies the SAS format DATE9.
DECIMAL NUMERIC(<i>p,s</i>)	SAS numeric	
DOUBLE	SAS numeric	
FLOAT(<i>p</i>)	SAS numeric	
INTEGER	SAS numeric	
NCHAR(<i>n</i>)	SAS character	Applies the SAS format \$ <i>n</i> .
NVARCHAR(<i>n</i>)	SAS character	Applies the SAS format \$ <i>n</i> .
REAL	SAS numeric	
SMALLINT	SAS numeric	
TIME(<i>p</i>)	SAS numeric	Applies the SAS format TIME8.
TIMESTAMP(<i>p</i>)	SAS numeric	Applies the SAS format DATETIME19.2.
TINYINT	SAS numeric	
VARBINARY(<i>n</i>)	SAS character	Applies the SAS format \$ <i>n</i> .
VARCHAR(<i>n</i>)	SAS character	Applies the SAS format \$ <i>n</i> .

¹ Valid SAS date values are in the range from 1582-01-01 to 9999-12-31. Dates outside the SAS date range are not supported and are treated as invalid dates.

For information about data type support for DBMS data sources, see [“Data Type Reference” in SAS FedSQL Language Reference](#).

FedSQL Data Type Support for CAS Tables

FedSQL supports CAS native data types only in CAS. The following table describes how FedSQL data types are mapped to CAS native types.

Table 23.2 FedSQL Data Type Translation for CAS Tables

FedSQL Data Type	CAS Data Type	Description
BIGINT	INT64	Large signed, exact whole number.
CHAR(<i>n</i>)	CHAR	Fixed-length character string, where <i>n</i> is the maximum number of characters to store.
DOUBLE	DOUBLE	Signed, approximate, double-precision, floating-point number. For the CAS server, this is a 64-bit double-precision, floating-point number.
INTEGER	INT32	Regular signed, exact whole number.
VARBINARY ¹	VARBINARY	Varying-length binary opaque data. This data type is used to store image data, audio, documents, and other unstructured data.
VARCHAR(<i>n</i>)	VARCHAR	Varying-length character string, where <i>n</i> is the maximum number of characters to store.

¹ FedSQL can read and write VARBINARY columns in CAS tables (with CREATE TABLE AS). However, the SELECT statement does not display VARBINARY columns in some clients unless you apply the \$HEX. format to the VARBINARY column with the PUT function. Attempts to read BINARY data yield an error.

Date, time, and timestamp values in CAS tables are supported as DOUBLES, with a SAS format applied. When reading numeric columns that contain date, time, and timestamp values from tables that were created by SAS engines into CAS, FedSQL preserves existing formats. For other input data sources, it applies the DATE9. SAS format to date values, the TIME8. SAS format to time values, and the DATETIME25.6 SAS format to datetime values.

CAS tables use the UTF-8 character set by default.

It is important to understand how FedSQL handles missing values in CAS. See [“Handling of Nonexistent Data” in SAS Viya Platform: FedSQL Programming for SAS Cloud Analytic Services](#).

PROC FEDSQL and PROC SQL: A Comparison

See [“A High-Level Comparison of PROC SQL and FedSQL”](#).

Examples: FEDSQL Procedure

Example 1: Creating a SAS Data Set

Features:

- PROC FEDSQL statement
- FedSQL CREATE TABLE statement
- FedSQL INSERT statement
- FedSQL SELECT statement
- QUIT statement
- LIBNAME statements

Details

Note: Data definition with the FedSQL language is not supported on the CAS server. In CAS, FedSQL can create new tables only by selecting and joining columns from existing tables.

This example creates a SAS data set by submitting FedSQL statements in the FEDSQL procedure. The example submits the FedSQL CREATE TABLE, INSERT, and SELECT statements.

Program

```
libname mybase 'library-path';

proc fedsql;

  create table mybase.sales (prodid double not null,
                             custid double not null,
```

```

totals double having format dollar10.,
country char(30));

insert into mybase.sales values (3234, 1, 189400, 'United States');
insert into mybase.sales values (1424, 3, 555789, 'Japan');
insert into mybase.sales values (3421, 4, 781183, 'Japan');
insert into mybase.sales values (3422, 2, 2789654, 'United States');
insert into mybase.sales values (3975, 5, 899453, 'Argentina');

title 'SAS Data Set "Sales"';
select * from mybase.sales;

quit;

```

Program Description

Assign a libref for a SAS library. The LIBNAME statement assigns the libref Mybase and specifies a physical location for the SAS data set. Because the LIBNAME statement does not specify an engine, the default SAS V9 engine is used to create the data set.

```
libname mybase 'library-path';
```

Execute the PROC FEDSQL statement. The PROC FEDSQL statement sets up an environment to submit FedSQL statements.

```
proc fedsql;
```

Enter the FedSQL CREATE TABLE statement. The statement specifies to create a table named Sales. Note that the FedSQL CREATE TABLE statement specifies the catalog identifier Mybase, which is the assigned libref. The variable declaration also defines a NOT NULL integrity constraint on columns Prodid and Custid, and applies the SAS format DOLLARw. on column Totals. FedSQL supports SAS formats, informats, and labels as SAS extensions for SAS data sources.

```

create table mybase.sales (prodid double not null,
                           custid double not null,
                           totals double having format dollar10.,
                           country char(30));

```

Enter INSERT statements to populate the table with data. Note that the table name is preceded with the catalog identifier in each statement. The data values in each statement are submitted in a comma-delimited string, preceded by the keyword VALUES.

```

insert into mybase.sales values (3234, 1, 189400, 'United States');
insert into mybase.sales values (1424, 3, 555789, 'Japan');
insert into mybase.sales values (3421, 4, 781183, 'Japan');
insert into mybase.sales values (3422, 2, 2789654, 'United States');
insert into mybase.sales values (3975, 5, 899453, 'Argentina');

```

Enter the FedSQL SELECT statement to display the content of the table. The asterisk within the SELECT statement specifies to displays the values in all columns and rows of the table specified in the FROM clause. Note that the table name is preceded by the catalog identifier. The SAS global statement TITLE specifies a title for the display.

```
title 'SAS Data Set "Sales"';
```

```
select * from mybase.sales;
```

Stop the procedure. The QUIT statement stops the procedure.

```
quit;
```

SAS Data Set "Sales"

Obs	PRODID	CUSTID	TOTALS	COUNTRY
1	3234	1	\$189,400	United States
2	1424	3	\$555,789	Japan
3	3421	4	\$781,183	Japan
4	3422	2	\$2,789,654	United States
5	3975	5	\$899,453	Argentina

Example 2: Creating a New Table from Existing Tables

Features:

PROC FEDSQL statement

CREATE TABLE statement with AS expression

LIBNAME statements

Three-way join

Details

This example creates a new SAS data set from existing tables by using PROC FEDSQL and the CREATE TABLE statement with the AS query expression syntax. The query expression selects rows from three existing tables — SAS data set Sales, SPD Engine data set Products, and Oracle table Customers — to create the new table: Results.

Program

```
libname mybase base 'library-path';
libname myspde spde 'library-path';
libname myoracle oracle path=orallg user=xxxxxx password=xxxxxx
schema=xxxxxx;

proc fedsql;
```



```

create table mybase.results as
  select products.prodid, products.product, customers.name,
         sales.totals, sales.country
  from myspde.products, mybase.sales, myoracle.customers
  where products.prodid = sales.prodid and
         customers.custid = sales.custid;

select * from mybase.results;

quit;

```

Program Description

Assign three librefs. The first LIBNAME statement assigns the libref Mybase, specifies the base engine, and specifies the physical location for the SAS data set to be created. The second LIBNAME statement assigns the libref Myspde, specifies the SPDE engine, and specifies the physical location for the existing SPD Engine data set. The third LIBNAME statement assigns the libref Myoracle, specifies the ORACLE engine, and specifies the connection information to the Oracle database that contains the existing Oracle tables.

```

libname mybase base 'library-path';
libname myspde spde 'library-path';
libname myoracle oracle path=orallg user=xxxxxx password=xxxxxx
schema=xxxxxx;

```

Execute the PROC FEDSQL statement. The PROC FEDSQL statement sets up an environment for submitting FedSQL statements.

```
proc fedsql;
```

Create the new table. The CREATE TABLE statement creates a new SAS data set from three existing SAS data sets by using a query expression to select rows from the existing data sets. The SELECT statement retrieves the qualified columns and rows from the existing data sets to create the new SAS data set.

```

create table mybase.results as
  select products.prodid, products.product, customers.name,
         sales.totals, sales.country
  from myspde.products, mybase.sales, myoracle.customers
  where products.prodid = sales.prodid and
         customers.custid = sales.custid;

```

Retrieve data in the SAS data set. This second SELECT statement displays the contents of the output data set.

```
select * from mybase.results;
```

Stop the procedure.

```
quit;
```

Output: Creating a New Table from Existing Tables

Output 23.1 Contents of Output Data Set Mybase.Results

PRODID	PRODUCT	NAME	TOTALS	COUNTRY
3234	Rice	Peter Frank	189,400	United States
3422	Oat	Jim Stew art	2789654	United States
1424	Corn	Janet Chien	555,789	Japan
3421	Wheat	Qing Ziao	781,183	Japan
3975	Barley	Humberto Sertu	899,453	Argentina

Example 3: Querying Data Using a Correlated Subquery

Features:

- PROC FEDSQL statement
- FedSQL SELECT statement
- LIBNAME statements

Details

This example illustrates querying data using a correlated subquery. In a correlated subquery, the WHERE clause in the subquery refers to values in a table in the outer query. The correlated subquery is evaluated for each row in the outer query. With correlated subqueries, FedSQL executes the subquery and the outer query together. FedSQL can perform heterogeneous correlated subqueries. FedSQL sends the subquery to be performed by the data source, which limits the result set that is transferred from the data source.

Note: Correlated subqueries are not yet supported on the CAS server.

Program

```
libname myspde spde 'library-path';
libname myoracle oracle path=orallg user=xxxxxx password=xxxxxx
schema=xxxxxx;
```

```
proc fedsql;
    title 'Result of Correlated Subquery';
    select * from myspde.product
        where exists (select * from myoracle.sales
                     where product.prodId=sales.prodId);
quit;
```

Program Description

Assign two librefs. The first LIBNAME statement assigns the libref Myspde, specifies the SPDE engine, and specifies the physical location for the SPD Engine data set. The second LIBNAME statement assigns the libref Myoracle, specifies the Oracle engine, and specifies the connection information to the Oracle database.

```
libname myspde spde 'library-path';
libname myoracle oracle path=orallg user=xxxxxx password=xxxxxx
schema=xxxxxx;
```

Execute the PROC FEDSQL statement. The PROC FEDSQL statement creates an environment for submitting FedSQL statements.

```
proc fedsql;
```

Submit a correlated query. The query specifies to retrieve the values from all columns in the Products table that have a ProdId value that matches a ProdId value in the Sales table. The global statement TITLE specifies a title for the output.

```
title 'Result of Correlated Subquery';
select * from myspde.product
    where exists (select * from myoracle.sales
                 where product.prodId=sales.prodId);
```

Stop the procedure.

```
quit;
```

Output: Querying Data Using a Correlated Subquery

Result of Correlated Subquery

PROID	PRODUCT
3234	Rice
1424	Corn
3421	Wheat
3422	Oat
3975	Barley

Example 4: Creating and Using an Index to Perform a Join

Features: PROC FEDSQL statement
 FedSQL language statements
 LIBNAME statements

Details

This example illustrates how to create an index for an Oracle table. It then illustrates how to use the index to perform a join of the Oracle table and an SPD Engine data set.

Note: The CREATE INDEX statement is not supported on the CAS server.

Program

```
libname myspde spde 'library-path';
libname myoracle oracle connection-options;

proc fedsql;
    create index prodid on myoracle.sales (prodid);
    title 'Result of Indexed Join';
    select * from myspde.product, myoracle.sales
        where product.prodid=sales.prodid;
quit;
```

Program Description

Assign librefs for the data sources. The first LIBNAME statement assigns the libref Myspde, specifies the SPDE engine, and specifies the physical location for the SPD Engine data set. The second LIBNAME statement assigns the libref Myoracle, specifies the ORACLE engine, and specifies the connection information to the Oracle database.

```
libname myspde spde 'library-path';
libname myoracle oracle connection-options;
```

Execute the PROC FEDSQL statement. The PROC FEDSQL statement creates an environment for submitting FedSQL statements.

```
proc fedsql;
```

Create an index for the Oracle table. The CREATE INDEX statement creates an index named Prodid in the Oracle table named Sales for the column Prodid.

```
create index prodid on myoracle.sales (prodid);
```

Retrieve columns and rows. The SELECT statement retrieves data from the SPD Engine data set named Product and the Oracle table named Sales. Even though the index is in the Oracle database, FedSQL can take advantage of the index to perform the join.

```
title 'Result of Indexed Join';
select * from myspde.product, myoracle.sales
where product.prodid=sales.prodid;
```

Stop the procedure.

```
quit;
```

Output: Creating and Using an Index to Perform a Join

Result of Indexed Join

PRODID	PRODUCT	Product ID	Customer ID	TOTALS	COUNTRY
3234	Rice	3234	1	\$189,400	United States
1424	Corn	1424	3	\$555,789	Japan
3421	Wheat	3421	4	\$781,183	Japan
3422	Oat	3422	2	\$2,789,654	United States
3975	Barley	3975	5	\$899,453	Argentina

Example 5: Using a DS2 Package in an Expression

Features:

- FedSQL SELECT statement
- Package method declaration
- DS2 PACKAGE statement
- DS2 METHOD statement
- DS2 DATA statement

Details

The FedSQL language supports the ability to invoke user-defined DS2 package methods as functions in the SELECT statement. This example creates and submits a DS2 package method named Add on a table named Numbers from PROC FEDSQL. The package method and table are created in the Work library. This functionality is also available when connecting to third-party databases.

Note: Package methods that are run from PROC FEDSQL can have only input arguments in the method. For more information, see [“Using DS2 Packages in Expressions” in SAS FedSQL Language Reference](#).

Note: User-defined package methods are not supported on the CAS server.

Program

```
proc ds2;
    package adder / overwrite =yes;
        method add( double x, double y ) returns double;
            return x + y;
        end;
    endpackage;
data numbers / overwrite = yes;
    dcl double x y;
    method init();
        dcl int i;
        do i = 1 to 10;
            x = i; y = i * i;
            output;
        end;
    end;
enddata;

run;

quit;

proc fedsql;
    select x, y, work.adder.add( x, y ) as z from work.numbers;
quit;
```

Program Description

Invoke the DS2 procedure.

```
proc ds2;
```

Define a package. This PACKAGE statement specifies to create a package named Adder.

```
package adder / overwrite =yes;
```

Define a method. This METHOD statement specifies to create a method named Add. The method Add has two input variables, X and Y, both of type DOUBLE, and it returns an output of type DOUBLE. The output variable contains the sum from adding the value of variable X to variable Y. The END and ENDPACKAGE statements signal the completion of the METHOD and PACKAGE declarations.

```
method add( double x, double y ) returns double;
return x + y;
end;
endpackage;
```

Create a table. This DATA statement specifies to create a table named Numbers. A library is not specified, so the Work library will be used. The table has two columns, X and Y, of type DOUBLE. The system INIT method is called to populate the table with values. A variable, I, is defined to hold input values. A DO statement uses variable I to insert rows containing the values 1 through 10 into column X. Then, it multiplies each instance of I with itself and inserts the result into column Y. The END and ENDDATA statements signal the completion of the DO statement, INIT method, and DATA statement declarations.

```
data numbers / overwrite = yes;
dcl double x y;
method init();
dcl int i;
do i = 1 to 10;
x = i; y = i * i;
output;
end;
end;
enddata;
```

Submit the DS2 statements.

```
run;
```

Stop the DS2 procedure.

```
quit;
```

Invoke the FEDSQL procedure.

```
proc fedsql;
```

Submit a SELECT statement that invokes the package method on the table. The statement specifies to select columns X and Y, and the result of the package method expression is column Z from table Numbers. The package method is referenced using a three-part name in the form [catalog.][schema.]package.method.

```
select x, y, work.adder.add( x, y ) as z from work.numbers;
```

Stop the FEDSQL procedure.

```
quit;
```

Here is the output from the SELECT statement:

x	y	z
1	1	2
2	4	6
3	9	12
4	16	20
5	25	30
6	36	42
7	49	56
8	64	72
9	81	90
10	100	110

Example 6: Querying Data in CAS

Features:

- PROC FEDSQL statement
- SESSREF= procedure option
- FedSQL language statements
- CAS statement
- CASLIB statement

Details

This example illustrates the steps necessary to query a database table named Employees from CAS.

Program

```
cas;
```



```
caslib castera desc='Teradata Caslib'
  datasource=(srctype='teradata',
    dataTransferMode='serial',
    username='myname',
    password='mypw',
    server='testserver',
    db='test');

proc fedsql sessref=casauto;

select Pos, count(Pos) as Count_Pos
  from employees
  group by Pos
  having count(Pos) >= 2;

quit;
```

Program Description

Establish a session on the CAS server. You establish a CAS session by submitting the CAS statement. This CAS session is created with the default name Casauto.

```
cas;
```

Add a caslib. In CAS, you identify your target data source using a caslib instead of a libref. This CASLIB statement defines a connection to a Teradata database and assigns the caslib Castera. Castera becomes the active caslib in the CAS session.

```
caslib castera desc='Teradata Caslib'
  datasource=(srctype='teradata',
    dataTransferMode='serial',
    username='myname',
    password='mypw',
    server='testserver',
    db='test');
```

Specify the PROC FEDSQL statement with the SESSREF= procedure option. In the SESSREF= procedure option, specify the CAS session name. The SESSREF= option specifies Casauto. The SESSREF= option instructs the procedure to pass the FedSQL language statements that follow to the fedSql.execDirect action and to process them in CAS session Casauto.

```
proc fedsql sessref=casauto;
```

Specify FEDSQL statements. This SELECT statement specifies to list all of the job titles in database table Employees that have at least two employees. The table reference indicates that table Employees exists in the active caslib. The FedSQL language supports single-source, full-query implicit SQL pass-through in CAS. If the target table has not yet been loaded into the CAS session, the request is evaluated for implicit pass-through. Implicit pass-through passes eligible requests to the data source for processing and loads the result set into CAS. FedSQL implicit pass-through is possible only for tables that have not yet been loaded into the CAS session. There are other important requirements. For information about these requirements, see [“FedSQL Implicit Pass-Through Facility in CAS” in SAS Viya Platform: FedSQL Programming for SAS Cloud Analytic Services](#). Unloaded tables that are not eligible for pass-through are automatically loaded into the CAS session

for processing by the CAS server. Tables already existing in the CAS session are processed by the CAS server.

```
select Pos, count(Pos) as Count_Pos
  from employees
 group by Pos
 having count(Pos) >= 2;
```

Stop the procedure.

```
quit;
```

Output: Result of Database Query from CAS

Output 23.2 Output of CAS Database Query

Pos	COUNT_POS
Manager	4.000000
Developer	2.000000
Sales Associate	2.000000

```
1          OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK
72
73          proc fedsql sessref=casauto;
74          select Pos, count(Pos) as Count_Pos
75              from castera.employees
76              groupby Pos
77              having count(Pos) >=2;
NOTE: The SQL statement was fully offloaded to the underlying data source via
full pass-through
78          quit;
```

Example 7: Explicitly Loading and Joining Tables in CAS

Features:

- PROC FEDSQL statement
- SESSREF= procedure option
- CAS statement
- CASLIB statement
- CAS LIBNAME statement
- DATA step
- CASUTIL procedure

Details

In some cases, some formatting is required before data can be processed successfully. This example explicitly loads three files that contain comma-delimited data into CAS. The CAS LIBNAME engine and the DATA step are used to format and load the tables. Then, after the tables are in CAS, PROC FEDSQL is used to join them and to create a new CAS table that contains the result set. The input files are named Supplier, Nation, and Customer. The output CAS table is named Newtable. All of the CAS tables are in-memory tables. They disappear at the end of the CAS session, unless you save or promote them using PROC CASUTIL.

The example assigns a caslib and a libref. The libref is mapped to the caslib in the CAS LIBNAME statement. The DATA step executes in the libref. When the SESSREF= option is specified in the PROC FEDSQL procedure statement, FedSQL statements are executed in a caslib.

Note: When formatting your FEDSQL requests, be aware that leading spaces before statements and clauses are important. Do not begin statements and clauses flush with the left margin. If you put a line break in a quoted string, always follow the line break with at least one blank.

Program

```
cas;

caslib casdata path='/r/ge.unx.company.com/vol/vol210/u21/myID/hold';

libname mycas cas sessref=casauto caslib=casdata;

data mycas.supplier;
  infile "/r/ge.unx.company.com/vol/vol210/u21/myID/hold/supplier.tbl"
  delimiter='|';
  length S_SUPPKEY 8. S_NAME VARCHAR(25) S_ADDRESS VARCHAR(40)
  S_NATIONKEY 8.
  S_PHONE VARCHAR(15) S_ACCTBAL 8. S_COMMENT VARCHAR(101);
  input  S_SUPPKEY S_NAME S_ADDRESS S_NATIONKEY S_PHONE S_ACCTBAL
  S_COMMENT;
run;

data mycas.nation;
  infile "/r/ge.unx.company.com/vol/vol210/u21/myID/hold/nation.tbl"
  delimiter='|';
  length N_NATIONKEY 8. N_NAME VARCHAR(25) N_REGIONKEY 8. N_COMMENT
  VARCHAR(152);
  input  N_NATIONKEY N_NAME N_REGIONKEY N_COMMENT;
run;

data mycas.customer;
```

```

infile "/r/ge.unx.company.com/vol/vol210/u21/myID/hold/customer.tbl"
delimiter='|';
length C_CUSTKEY 8. C_NAME VARCHAR(25) C_ADDRESS VARCHAR(40)
C_NATIONKEY 8.
C_PHONE VARCHAR(15) C_ACCTBAL 8. C_MKTSEGMENT VARCHAR(10) C_COMMENT
VARCHAR(117);
input C_CUSTKEY C_NAME C_ADDRESS C_NATIONKEY C_PHONE C_ACCTBAL
C_MKTSEGMENT
C_COMMENT;
run;

proc casutil;
list tables incaslib="casdata";
run;

proc fedsql sessref=casauto;

create table newtable {options replace=true} as
select
s_name, s_acctbal, n_name, sum_c_acctbal
from
supplier,
nation,
(select c_nationkey, sum(c_acctbal) as sum_c_acctbal from customer
group by
c_nationkey) C
where
s_nationkey = n_nationkey and
s_nationkey = c_nationkey
;

select * from newtable;

quit;

```

Program Description

Establish a session on the CAS server. This step is necessary only if a CAS session does not already exist or you want to create a second CAS session. This session uses the default name Casauto.

```
cas;
```

Assign a caslib that points to your input files. This CASLIB statement assigns the caslib Casdata to the location specified in the PATH= parameter. The path specification must use an absolute pathname.

```
caslib casdata path='/r/ge.unx.company.com/vol/vol210/u21/myID/hold';
```

Assign a CAS engine libref. The LIBNAME statement specifies the libref Mycas, the CAS engine, and the Casdata caslib. The LIBNAME statement invokes the CAS engine and maps the libref to the caslib.

```
libname mycas cas sessref=casauto caslib=casdata;
```

Use the DATA step to format and load the first file into CAS. The DATA statement specifies to create a table named Mycas.Supplier. The CAS engine creates the output table as a CAS table. In the SAS session, the table is known as

Mycas.Supplier. In the CAS session (Casauto), the table is known as Casdata.Supplier. The INFILE= statement specifies to read the contents of the file using a | (pipe symbol) as a column delimiter. The LENGTH statement specifies column names and lengths for the output table. (Note that the INFILE specification can be relative to the path that is specified in the CASLIB statement.) The CAS engine creates table Supplier in caslib Casdata.

```
data mycas.supplier;
  infile "/r/ge.unx.company.com/vol/vol210/u21/myID/hold/supplier.tbl"
  delimiter='|';
  length S_SUPPKEY 8. S_NAME VARCHAR(25) S_ADDRESS VARCHAR(40)
  S_NATIONKEY 8.
  S_PHONE VARCHAR(15) S_ACCTBAL 8. S_COMMENT VARCHAR(101);
  input S_SUPPKEY S_NAME S_ADDRESS S_NATIONKEY S_PHONE S_ACCTBAL
  S_COMMENT;
run;
```

Format and load the second file into CAS. This DATA statement specifies to create a CAS table named Mycas.Nation. The INFILE= statement reads the contents of the file using a pipe symbol as a delimiter. The LENGTH statement specifies column names and lengths for the output table. The CAS engine creates table Nation in caslib Casdata.

```
data mycas.nation;
  infile "/r/ge.unx.company.com/vol/vol210/u21/myID/hold/nation.tbl"
  delimiter='|';
  length N_NATIONKEY 8. N_NAME VARCHAR(25) N_REGIONKEY 8. N_COMMENT
  VARCHAR(152);
  input N_NATIONKEY N_NAME N_REGIONKEY N_COMMENT;
run;
```

Format and load the third file into CAS. The DATA step specifies to create a CAS table named Mycas.Customer. The INFILE= statement reads the contents of the file. The LENGTH statement specifies column names and lengths for the output table. The CAS engine creates table Customer in caslib Casdata.

```
data mycas.customer;
  infile "/r/ge.unx.company.com/vol/vol210/u21/myID/hold/customer.tbl"
  delimiter='|';
  length C_CUSTKEY 8. C_NAME VARCHAR(25) C_ADDRESS VARCHAR(40)
  C_NATIONKEY 8.
  C_PHONE VARCHAR(15) C_ACCTBAL 8. C_MKTSEGMENT VARCHAR(10) C_COMMENT
  VARCHAR(117);
  input C_CUSTKEY C_NAME C_ADDRESS C_NATIONKEY C_PHONE C_ACCTBAL
  C_MKTSEGMENT
  C_COMMENT;
run;
```

Verify that the files were created in CAS. Submit the CASUTIL procedure to list the tables that are available in caslib Casdata.

```
proc casutil;
  list tables incaslib="casdata";
run;
```

Specify the PROC FEDSQL statement. In the PROC FEDSQL statement, the SESSREF= procedure option specifies the CAS session name. The SESSREF=

option specifies Casauto. Because of earlier activity in the CAS session, Casdata is the active caslib.

```
proc fedsql sessref=casauto;
```

Submit the FedSQL statements. The CREATE TABLE statement specifies to create a table named Newtable using columns from the Supplier, Nation, and Customer tables. The SELECT statement retrieves the columns S_NAME, S_ACCTBAL, N_NAME from the tables and creates a new column SUM_C_ACCTBAL by issuing a subquery. The tables are joined based on the values in the S_NATIONKEY, N_NATIONKEY, and C_NATIONKEY columns. FedSQL combines the data and returns the results in the new CAS table.

```
create table newtable {options replace=true} as
select
  s_name, s_acctbal, n_name, sum_c_acctbal
from
  supplier,
  nation,
  (select c_nationkey, sum(c_acctbal) as sum_c_acctbal from customer
group by
  c_nationkey) C
where
  s_nationkey = n_nationkey and
  s_nationkey = c_nationkey
;
```

Display the contents of table Newtable. The SELECT statement specifies to display the content of CAS table Newtable. Newtable is an in-memory table. To persist or promote it, you must use PROC CASUTIL.

```
select * from newtable;
```

Stop the procedure.

```
quit;
```

Output: Joining Tables That Were Explicitly Loaded into CAS

Output 23.3 Output of the CASUTIL Procedure LIST Statement

The CASUTIL Procedure									
Table Information for Caslib CASDATA									
Table Name	Number of Rows	Number of Columns	NLS encoding	Created	Last Modified	Promoted Table	Repeated Table	View	Compressed
SUPPLIER	10000	7	utf-8	08Jul2016:13:57:25	08Jul2016:13:57:25	No	No	No	No
NATION	25	4	utf-8	08Jul2016:13:57:43	08Jul2016:13:57:43	No	No	No	No
CUSTOMER	150000	8	utf-8	08Jul2016:13:58:14	08Jul2016:13:58:14	No	No	No	No

Output 23.4 A Portion of the SELECT Results for Table Newtable

S_NAME	S_ACCTBAL	N_NAME	SUM_C_ACCTBAL
Supplier#000000014	9189.820000	MOROCCO	26625512
Supplier#000000153	850.550000	INDONESIA	27930482
Supplier#000000292	9598.620000	VIETNAM	27081998
Supplier#000000431	9477.340000	CANADA	27025344
Supplier#000000570	922.720000	PERU	26556292
Supplier#000000709	8638.350000	SAUDI ARABIA	26455693
Supplier#000000848	4404.290000	CANADA	27025344
Supplier#000000987	-40.300000	CHINA	26740212
Supplier#000001126	7243.050000	JAPAN	26898469
Supplier#000001265	3986.630000	IRAQ	26919597
Supplier#000001404	5223.720000	ALGERIA	26322970
Supplier#000001543	2107.210000	EGYPT	27100354
Supplier#000001682	9413.980000	CANADA	27025344
Supplier#000001821	7431.290000	VIETNAM	27081998
Supplier#000001960	443.600000	SAUDI ARABIA	26455693
Supplier#000002099	7043.940000	ALGERIA	26322970
Supplier#000002238	5244.930000	JORDAN	26891366
Supplier#000002377	7455.660000	MOZAMBIQUE	27022906
Supplier#000002516	2819.790000	EGYPT	27100354
Supplier#000002655	4856.740000	VIETNAM	27081998
Supplier#000002794	4271.280000	KENYA	27406567
Supplier#000002933	3557.950000	BRAZIL	26821676
Supplier#000003072	7279.730000	INDIA	27293627
Supplier#000003211	8315.050000	UNITED STATES	27316299

Example 8: Joining Tables from Multiple CAS Libraries

Features:

- PROC FEDSQL statement
- SESSREF= procedure option
- FedSQL language statements
- CAS statement
- CASUTIL procedure
- CASLIB statements
- CASLIB session option

Details

This example shows how to perform the join in [“Example 2: Creating a New Table from Existing Tables” on page 818](#) on the CAS server. A difference is that this example joins a SAS data set and an SPD Engine data set with a Teradata table instead of an Oracle table.

Start a CAS session. This step is necessary only if a CAS session does not already exist or you want to create a second CAS session. Here, a new session is created. The session is created with the name Mysess.

```
cas mysess;
```

Load the SAS data set Customers into your CAS session. PROC CASUTIL loads the data set into the initially active caslib, Casuser. It does so by specifying a data path.

```
proc casutil;
  load data="path-to-customers-data-set" outcaslib="casuser";
quit;
```

Add a Teradata caslib. The CASLIB statement specifies to add caslib TDcaslib.

```
caslib TDcaslib desc='Teradata Caslib'
  datasource=(srctype='teradata'
    username='myname'
    password='mypw'
    server='testserver',
    db='test')
  notactive;
```

Add an SPD Engine caslib. The CASLIB statement specifies to add caslib spdecaslib. The caslib specifies SrcType=SPDE and it specifies the path to the directory that contains the metadata file for SPD Engine data set Products. Because spdecaslib is the last caslib assigned, it is the active caslib.

```
caslib spdecaslib Desc="SPD Engine caslib"
  datasource=(srctype="spde", username="",
  mdfilepath="path-to-metafile",
  dataTransferMode="serial");
run;
```

(Optional) Change the active caslib. This example shows how to set the active caslib by using the CAS statement and the CASLIB session option. The example sets TDcaslib as the active caslib.

```
cas mysess sessopts=(caslib=TDcaslib);
```

Submit the join request. The PROC FEDSQL statement specifies the SESSREF= procedure option. The SESSREF= option specifies CAS session Mysess. Table Products is an SPD engine data set. Table Customers is a SAS data set. These are identified in the CREATE TABLE statement using two-part table names in the form *caslib.table-name*. Table Sales, a Teradata table, is identified simply by name. The output table, Results, is created in the active caslib. If you intend to save the

output, you might want to create the table in a different caslib. A SELECT statement requests to print the contents of table Results.

```
proc fedsql sessref=mysess;
create table results as
    select products.prodid, products.product, customers.name,
           sales.totals, sales.country
    from spdecaslib.products, sales, casuser.customers
    where products.prodid = sales.prodid and
           customers.custid = sales.custid;

select * from results;
quit;
```

Output: Joining Tables from Multiple CAS Libraries

Output 23.5 Contents of CAS Table Results

PRODID	PRODUCT	NAME	TOTALS	COUNTRY
3421.000000	Wheat	Qing Ziao	781183	Japan
1424.000000	Corn	Janet Chien	555789	Japan
3975.000000	Barley	Humberto Sertu	899453	Argentina
3421.000000	Wheat	Jim Stewart	2789654	United States
3234.000000	Rice	Peter Frank	189400	United States

FMTC2ITM Procedure

Overview: FMTC2ITM Procedure	837
What Does the FMTC2ITM Procedure Do?	837
Syntax: FMTC2ITM Procedure	838
FMTC2ITM Statement	838
SELECT Statement	840
Example: Migrate Formats to a CAS Session	841

Overview: FMTC2ITM Procedure

What Does the FMTC2ITM Procedure Do?

The FMTC2ITM procedure converts one or more format catalogs into a single item store that can be made available to CAS. PROC FMTC2ITM enables you to create an item store that has the following contents:

- all of the formats from one catalog
- a subset of the formats from one catalog
- all of the formats from multiple catalogs
- a subset of the formats from each of multiple catalogs.

After you create an item store with PROC FMTC2ITM, you can use the CAS statement with the addFmtLib action to make the item store available to CAS. For more information, see [“CAS Statement” in SAS Cloud Analytic Services: User's Guide](#) and [addFmtLib Action](#).

Syntax: FMTC2ITM Procedure

```
PROC FMTC2ITM <options>;  
  <SELECT member-list>;
```

Statement	Task
FMTC2ITM	Converts one or more format catalogs into a single item store.
SELECT	Lists the formats to be placed in the item store.

FMTC2ITM Statement

Converts one or more format catalogs into a single item store.

Note: The item store is written as a new file. If you specify the name of an existing item store with the ITEMSTORE option, it overwrites the contents of the existing item store.

Syntax

```
FMTC2ITM <options>;
```

Required Arguments

CATALOG=memname | libname.memname(| list)
specifies a catalog that is to be converted to an item store. If you do not specify the CATALOG option, the default catalog is WORK.FORMATS.

You can specify the following values for the CATALOG option:

- A single-level name that SAS interprets as a catalog name in the WORK library.
- A two-level name that SAS interprets as a libname.memname for a catalog.
- A list enclosed in parentheses that SAS interprets as a list of catalog names.

SAS opens each catalog in the list in the listed order and writes the members of the catalogs to the item store. Only the first occurrence of the member is written. For example, if CATALOG A has members X and Y, and CATALOG B has members X and Z as specified in this code example:

```
proc fmtc2itm catalog=(a b) itemstore='itemstoreA'; run;
```

The resulting item store contains members X and Y from CATALOG A, and member Z from CATALOG B.

ITEMSTORE=fileref | 'filename'

specifies the name of the item store. You can specify a fileref, or you can specify a pathname in quotation marks.

Note: You can specify only one item store with each invocation of PROC FMTC2ITM.

ENCODING=encoding-name

specifies an encoding for a catalog or for all of the catalogs in a list.

To specify an encoding for one catalog, specify the ENCODING= option after the catalog name as shown in this example code.

```
proc fmtc2itm cat=(abc.fmtlib1/encoding=utf8 abc.fmtlib2);
```

SAS applies the UTF8 encoding to only the Abc.Fmtlib1 catalog. The Abc.Fmtlib2 catalog uses the session encoding.

To specify an encoding for a list of format catalogs, specify the ENCODING= option at the end of a list of catalogs.

```
proc fmtc2itm cat=(abc.fmtlib1 abc.fmtlib2) encoding=utf8;
```

SAS applies the UTF8 encoding to all of the catalogs in the list.

If you do not specify the ENCODING= option for a catalog, then SAS assigns the session encoding option to the catalog.

PROC FMTC2ITM validates all of the character data (all labels and character range values) in a catalog to ensure that they are valid for the specified encoding. SAS issues an error if any of the characters do not transcode successfully.

Optional Arguments

PRINT

displays information about each catalog member that is written.

LOCALE

adds locale-sensitive prefixes to the names of members of an item store.

If you specify the LOCALE option, then the processing of the parenthetical list of member names is different from the usual processing of member names in a list. If any catalog has a locale suffix (of the form `_xx` or `_xx_yy`), then the members from the catalog are written to the item store with that suffix as a prefix. For example, if catalog X_EN_US has members ABC, DEF, and GHI, and catalog X_FR_FR also has members ABC, DEF and JKL as specified in this code example

```
proc fmtc2itm catalog=(x_en_us x_fr_fr) itemstore locale; run;
```

The resulting item store contains the members EN_US-ABC, EN_US-DEF, EN_US-GHI, FR_FR-ABC, FR_FR-DEF, and FR_FR-JKL. A subsequent usage of

the item store in CAS allows for ABC or DEF to be loaded properly based on an EN_US or FR_FR locale. If the LOCALE option is not provided, the item store contains the members ABC, DEF, and GHI from X_EN_US, and member JKL from X_FR_FR.

Note: When you specify PROC FMTC2ITM, your SAS session must use the same encoding that was used when the format catalogs were created. For example, if the EN_US locale was used when the catalogs were created, then the session where you specify PROC FMTC2ITM must also use the EN_US locale.

SELECT Statement

Lists the formats to place in the item store.

Tip: If you do not specify the members of the format catalog with a SELECT statement, then all formats in the catalog are written to the item store.

Syntax

```
SELECT <member-list>;
```

Optional Argument

member-list

Contains the names of the formats to place in the item store.

Details

The SELECT statement enables you to select only the formats from a format library that you want to add to an item store. For example, if you had a format library that contained the formats CHOICE, SINGLE2, TESTFMTA, WHEN, and WHERE, you could specify the WHEN and WHERE formats with the SELECT statement. The WHEN and WHERE formats are then added to the item store, but the CHOICE, SINGLE2, and TESTFMTA, formats are not added.

Example: Migrate Formats to a CAS Session

Features:

- PROC FMTC2ITM statement options
 - CATALOG
 - ITEMSTORE
- CAS statement options
 - LOADFORMATS
 - LISTFORMAT
 - SAVEFMTLIB

Details

This example uses the FMTC2ITM procedure to migrate user-defined formats that are stored in one or more SAS format catalogs to a format library in a CAS session.

Program

```
libname orion "path-to-library";

proc format;
  value $codes
    "A" = "Alpha"
    "B" = "Beta"
    "C" = "Charlie"
    "D" = "Delta";
  value response
    1 = "Yes"
    2 = "No"
    3 = "Undecided"
    4 = "No response";
  value MPGrating
    34 - HIGH = "Excellent"
    24 -< 34 = "Good"
    19 -< 24 = "Fair"
    LOW -< 19 = "Poor";
run;

proc format library=orion.mailfmts;
  value $officeCodes
    "CHI" = "Chicago"
```

```

        "NYC" = "New York"
        "ATL" = "Atlanta"
        "CUP" = "Cupertino";
value $regionCodes
    "E" = "East"
    "W" = "West"
    "N" = "North"
    "S" = "South";
run;

cas casauto;

proc fmtc2itm
    catalog=(work.formats orion.mailfmts)
    itemstore="path-to-item-store-file";
run;

cas casauto addfmtlib fmtlibname="myfmtlib"
    path=path-to-item-store-file
    replacefmtlib;

cas casauto listformat fmtlibname="myfmtlib"
    members;

cas casauto savefmtlib fmtlibname=myfmtlib
    caslib=casuser table=myfmtlib replace;

```

Program Description

Create the Orion library and add the formats.

```

libname orion "path-to-library";

proc format;
    value $codes
        "A" = "Alpha"
        "B" = "Beta"
        "C" = "Charlie"
        "D" = "Delta";
    value response
        1 = "Yes"
        2 = "No"
        3 = "Undecided"
        4 = "No response";
    value MPGGrating
        34 - HIGH = "Excellent"
        24 -< 34 = "Good"
        19 -< 24 = "Fair"
        LOW -< 19 = "Poor";
run;

proc format library=orion.mailfmts;
    value $officeCodes
        "CHI" = "Chicago"
        "NYC" = "New York"
        "ATL" = "Atlanta"
        "CUP" = "Cupertino";

```



```

value $regionCodes
  "E" = "East"
  "W" = "West"
  "N" = "North"
  "S" = "South";
run;

```

Start a CAS session.

```
cas casauto;
```

Create the item store with the formats. The FMTC2ITM procedure writes the formats in format catalogs Work.Formats and Orion.Mailfmts to an item store file. The CATALOG option specifies to search the format catalogs Work.Formats and Orion.Mailfmts. The ITEMSTORE option specifies the path where the item store is to be written. To select a subset of the formats in the specified format catalogs, specify a SELECT statement in the FMTC2ITM procedure step.

```

proc fmtc2itm
  catalog=(work.formats orion.mailfmts)
  itemstore="path-to-item-store-file";
run;

```

Load the Formats The CAS statement ADDFMTLIB option uses the item store file that you created with the FMTC2ITM procedure to add the format library Myfmtlib.

```

cas casauto addfmtlib fmtlibname="myfmtlib"
  path=path-to-item-store-file
  replacefmtlib;

```

List the formats. The CAS statement LISTFORMAT option lists the formats in format library Myfmtlib to the SAS log for verification.

```

cas casauto listformat fmtlibname="myfmtlib"
  members;

```

Save the format library. The CAS statement SAVEFMTLIB option saves the format library to a SASHDAT file. This step is optional.

For format libraries that you use repeatedly, saving to a caslib is a best practice. Use the CAS statement ADDFMTLIB option with parameters CASLIB= and TABLE= when adding a format library from a caslib.

```

cas casauto savefmtlib fmtlibname=myfmtlib
  caslib=casuser table=myfmtlib replace;

```


FONTREG Procedure

Overview: FONTREG Procedure	845
What Does the FONTREG Procedure Do?	845
Concepts: FONTREG Procedure	846
Supported Font Types and Font Naming Conventions	846
Registering Fonts with PROC FONTREG	847
Removing Fonts from the SAS Registry	848
Using a Fileref with PROC FONTREG	849
Font Aliases and Locales	849
Syntax: FONTREG Procedure	850
PROC FONTREG Statement	850
FONTFILE Statement	852
FONTPATH Statement	854
REMOVE Statement	855
TRUETYPE Statement	857
TYPE1 Statement	857
OPENTYPE Statement	858
Examples: FONTREG Procedure	858
Example 1: Adding a Single Font File	858
Example 2: Adding All Font Files from Multiple Directories	859
Example 3: Replacing Existing TrueType Font Files from a Directory	861

Overview: FONTREG Procedure

What Does the FONTREG Procedure Do?

The FONTREG procedure enables you to update the SAS registry to include system fonts, which can then be used in SAS output. PROC FONTREG uses FreeType font-

rendering to recognize and incorporate various types of font definitions. Fonts of any type that can be incorporated and used by SAS are known collectively in this documentation as fonts in the FreeType library.

Note: Including a system font in the SAS registry means that SAS knows where to find the font file. The font file is not actually used until the font is called for in a SAS program. Therefore, do not move or delete font files after you have included the fonts in the SAS registry.

For more information, see the following sources:

- “Specifying Fonts in SAS/GRAPH Programs” in *SAS/GRAPH: Reference*
- “GDEVICE Procedure” in *SAS/GRAPH: Reference*
- “FONTSLOC= System Option” in *SAS System Options: Reference*
- www.freetype.org for information about the FreeType project

Concepts: FONTREG Procedure

Supported Font Types and Font Naming Conventions

When a font is added to the SAS registry, the font name is prefixed with a three-character tag, enclosed in angle brackets (< >). This prefix indicates the font type. For example, if you add the TrueType font **Arial** to the SAS registry, then the name in the registry is <ttf> **Arial**. This naming convention enables you to add and distinguish between fonts that have the same name but are of different types.

When you specify a font in a SAS program, use the three-character tag to distinguish between fonts that have the same name:

```
proc report data=sashelp.class nowd
            style(header)=[font_face='<ttf> Palatino Linotype'];
run;
```

Examples of when you can specify a font in a SAS program are in the TEMPLATE procedure or in the STYLE= option in the PROC REPORT.

If you do not include a tag in your font specification, SAS searches the registry for fonts with that name. If more than one font with that name is found, SAS uses the font that has the highest rank in the following table.

Table 25.1 Supported Font Types

Rank	Type	Tag	File Extension
1	TrueType	<ttf>	.ttf
2	Type1	<at1>	.pfa .pfb
3	OpenType	<cff>	.otf

Note: OpenType font is an extension of TrueType font and is supported by SAS. OpenType contains the family values for serif, sans-serif, monospace, and symbol fonts. OpenType registers .otf font files.

Note: PDF and PostScript do not support double-byte Type1 fonts.

SAS does not support any type of nonscalable fonts that require FreeType font-rendering. Even if they are recognized as valid fonts, they will not be added to the SAS registry.

Font files that are not produced by major vendors can be unreliable, and in some cases SAS might not be able to use them.

The following SAS output methods and device drivers can use FreeType font-rendering:

SAS/GRAPH GIF, GIFANIM	Universal PNG
SAS/GRAPH JPEG	Universal Printing GIF
SAS/GRAPH PCL	Universal Printing TIFF
SAS/GRAPH PNG	Universal Printing PCL
SAS/GRAPH SASEMF	Universal Printing PDF
SAS/GRAPH SASWMF	Universal PS
SAS/GRAPH TIFFP, TIFFB	Universal SVG
Universal EMF	

Registering Fonts with PROC FONTREG

PROC FONTREG is used to register fonts in the SAS Registry. For example, if you have a Type1 or OpenType font in your Windows font directory, you can register the font by submitting the following code:

```
proc fontreg mode=add;
fontpath '!SYSTEMROOT\fonts';
run;
```

This code will register all of the other font files in the Windows font directory.

Removing Fonts from the SAS Registry

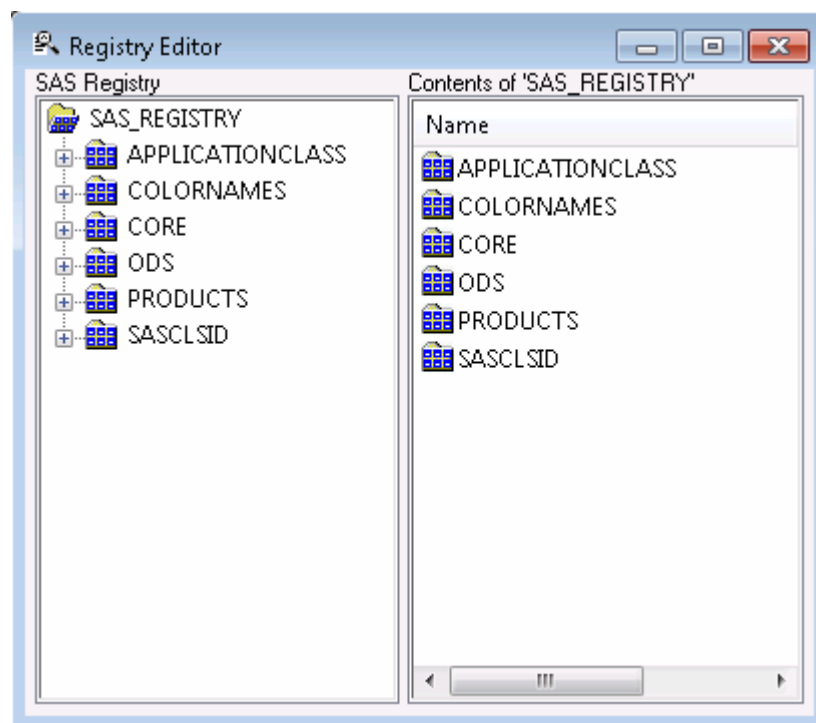
You can remove a font from the SAS registry in the following ways:

- by using the SAS Registry Editor
- by using PROC REGISTRY
- by using the REMOVE statement in PROC FONTREG

To remove a font by using the SAS Registry Editor, select **Solutions** ⇒ **Accessories** ⇒ **Registry Editor**. Alternatively, you can enter `regedit` in the command window or **Command** ==> prompt.

The following display shows the SAS Registry Editor window.

Figure 25.1 SAS Registry Editor Window



In the left pane of the Registry Editor window, navigate to the [CORE\PRINTING\FREETYPE\FONTS] key. Select the font that you want to delete, and use one of these methods to delete it:

- Right-click the font name and select **Delete** from the menu.
- Select the **Delete** button .
- Select **Edit** ⇒ **Delete** ⇒ **Key**.

To delete a font by using PROC REGISTRY, submit a program similar to the following example. This example removes the <ttf> Arial font.

```
/* Write the key name for the font to an external file */
proc registry export='external-filename'
               startat='core\printing\freetype\fonts\<ttf> Arial';
run;

/* Remove the "<ttf> Arial" font from the SAS registry */
proc registry
  uninstall='external-filename' fullstatus;
run;
```

To delete a font by using the REMOVE statement in PROC FONTREG, see [“REMOVE Statement” on page 855](#).

For more information about PROC REGISTRY, see [Chapter 50, “REGISTRY Procedure,” on page 1733](#).

Using a Fileref with PROC FONTREG

You can use a fileref with the FONTPATH, TRUETYPE, TYPE1, and OPENTYPE statements in PROC FONTREG, if you first define a filename. The following examples show how a fileref is used:

```
filename fonts1 'c:\windows\fonts';
proc fontreg mode=all;
  fontpath fonts1;
run;

proc fontreg mode=all;
  truetype fonts1;
run;
```

The ability to use a fileref enables you to directly use the FILENAME statement and its features. For example, you can register available fonts by using a URL. With fileref support, you would use a FILENAME statement and a PROC FONTREG step.

Font Aliases and Locales

The FONTFILE, FONTPATH, TRUETYPE, and OPENTYPE statements support aliases and locales. If the font that is being processed contains a localized name in the same locale as the current SAS session, then an alias of that localized name will be added to the SAS registry to reference the font family.

Syntax: FONTREG Procedure

Interaction: If no statements are specified, then PROC FONTREG searches for TrueType font files in the directory that is indicated in the FONTSLOC= SAS system option.

Tip: If you specify more than one statement, then the statements are executed in the order in which they appear, except for REMOVE statements, which are always executed first. You can use the same statement more than once in a single PROC FONTREG step.

PROC FONTREG <options>;

FONTFILE 'file' <...'file'> | 'file-1, pfm-file-1, afm-file-1' <...'file-n'> ||;

FONTPATH <fileref> 'directory' <...'directory'>;

OPENTYPE <fileref> 'directory' <...'directory'>

REMOVE 'family-name' | 'alias' | family-type | _ALL_;

TRUETYPE <fileref> 'directory' <...'directory'>;

TYPE1 <fileref> 'directory' <...'directory'>;

Statement	Task
PROC FONTREG	Specify how to handle new and existing fonts
FONTFILE	Identify which font files to process
FONTPATH	Search directories to identify valid font files to process
REMOVE	Remove a font family, all fonts of a particular type, or all fonts from the Core\Printing\Freetype\Fonts location of the SAS registry
TRUETYPE	Search directories to identify TrueType font files
TYPE1	Search directories to identify valid Type 1 font files
OPENTYPE	Search directories to identify valid OpenType font files

PROC FONTREG Statement

Enables you to update the SAS registry to include system fonts, which can then be used in SAS output.

Syntax

PROC FONTREG *<options>*;

Summary of Optional Arguments

MODE=ADD | REPLACE | ALL

specifies how to handle new and existing fonts.

MSGLEVEL=VERBOSE | NORMAL | TERSE | NONE

specifies the level of detail to include in the SAS log.

NOUPDATE

specifies that the procedure should run without updating the SAS registry.

USESASHELP

specifies that the SAS registry in the Sashelp library be updated.

Optional Arguments

MODE=ADD | REPLACE | ALL

specifies how to handle new and existing fonts in the SAS registry:

ADD

specifies to add fonts that do not already exist in the SAS registry. Do not modify existing fonts.

REPLACE

specifies to replace fonts that already exist in the SAS registry. Do not add new fonts.

ALL

specifies to add new fonts that do not already exist in the SAS registry and replace fonts that already exist in the SAS registry.

Default **ADD**

Example [“Example 3: Replacing Existing TrueType Font Files from a Directory” on page 861](#)

MSGLEVEL=VERBOSE | NORMAL | TERSE | NONE

specifies the level of detail to include in the SAS log.

Here are the levels:

VERBOSE

SAS log messages include which fonts were added, which fonts were not added, and which fonts were not understood. The log also contains a summary that indicates the number of fonts that were added, not added, and not understood.

NORMAL

SAS log messages include which fonts were added, and a summary that indicates the number of fonts that were added, not added, and not understood.

TERSE

SAS log messages include only the summary that indicates the number of fonts that were added, not added, and not understood.

NONE

No messages are written to the SAS log, except for errors (if encountered).

Default **TERSE**

Example [“Example 2: Adding All Font Files from Multiple Directories” on page 859](#)

NOUPDATE

specifies that the procedure should run without actually updating the SAS registry. This option enables you to test the procedure on the specified fonts before modifying the SAS registry.

USESASHELP

specifies that the SAS registry in the Sashelp library should be updated. You must have Write access to the Sashelp library in order to use this option. If the USESASHELP option is not specified, then the SAS registry in the Sasuser library is updated.

FONTFILE Statement

Specifies one or more font files to be processed.

See: [“Example 1: Adding a Single Font File” on page 858](#)

Syntax

FONTFILE '*file*' <...'file'> | '*file-1*, *pfm-file-1*, *afm-file-1*' <...'file-*n*'>;

Required Arguments

file

is the complete pathname to a font file. If the file is recognized as a valid font file, then the file is processed. Each pathname must be enclosed in quotation marks. If you specify more than one pathname, then you must separate the pathnames with a space.

pfm-file

specifies a file specific to Windows that contains font metrics as well as the value of the Windows font name.

afm-file

specifies a file that contains font metrics.

Details

Processing a Type1 Font

When a valid Type1 font is processed by the TYPE1 statement or the FONTPATH statement, SAS attempts to find a corresponding PFM or AFM font metric file in the same directory that contains the font file. The font filename prefix is used with the .PFM and .AFM extensions to generate metric filenames. If these files are opened successfully and are determined to be valid metric files, then they will be associated with the font in the font family when they are added to the SAS registry.

If you specify a Type1 font in the FONTFILE statement, and you do not specify a PFM or AFM file, then SAS does not search for the PFM or AFM files.

Specifying a PFM or AFM File

If the font file contains a Type1 font, then you can also specify its corresponding PFM or AFM file as well. You must specify the full host name (directory and filename) for each file, and all files must be grouped together and enclosed in quotation marks, as in this example:

```
fontfile 'c:\winnt\fonts\alpinerg.pfb,
        c:\winnt\fonts\alpinerg.pfm,
        c:\winnt\fonts\alpinerg.afm';
```

If you specify an AFM file but do not specify a PFM file, then you must use a comma as a placeholder for the missing PFM file, as in this example:

```
fontfile 'c:\winnt\fonts\alpinerg.pfb, ,
        c:\winnt\fonts\alpinerg.afm';
```

If you specify a PFM file but do not specify an AFM file, then you do not need a comma as a placeholder for the missing AFM file, as in this example:

```
fontfile 'c:\winnt\fonts\alpinerg.pfb,
        c:\winnt\fonts\alpinerg.pfm';
```

When you specify a PFM or AFM file, SAS attempts to open the file and determine whether the file is of the specified type. If it is not, then SAS writes a message to the log and the file is not used.

The PFM file is a file that is specific to Windows, and contains font metrics as well as a value for the Windows Font Name field. If you specify a valid PFM file, then SAS opens the file, retrieves the value in Windows Font Name, and saves it with the font in the SAS registry. SAS uses this field when it creates a file (such as an EMF formatted file) to export into a Windows application.

Not Specifying a PFM or AFM File

You do not need to specify a PFM or AFM file along with a Type1 font file in a FONTFILE statement. In this case, no metric file information is added to the font in

the font family in the SAS registry. If an existing font family that contains multiple styles and weights already exists in the SAS registry, and the FONTFILE statement is used to replace one of the fonts in that family, then all of the information for that font will be updated. The replacement also updates the Host Filename, PFM Name, AFM Name, and Windows Font Name.

Note: If you replace a font in a family and the font contains values for the PFM Name or AFM Name, specifying a missing or invalid value for the metric in the FONTFILE statement causes the corresponding metric value to be deleted from the font in the registry.

Note: You cannot use a PFM or AFM file specification if you specify a TrueType font.

FONTPATH Statement

Specifies one or more directories to be searched for valid font files to process.

See: [“Example 2: Adding All Font Files from Multiple Directories” on page 859](#)

Syntax

FONTPATH <fileref> '*directory*' <...'*directory*'>;

Required Argument

directory

specifies a directory to search. All files that are recognized as valid font files are processed. Each directory must be enclosed in quotation marks. If you specify more than one directory, then you must separate the directories with a space.

Operating Environment Information: In the Windows operating environment only, you can locate the fonts folder if you do not know where the folder resides. In addition, you can register system fonts without having to know where the fonts are located. To find this information, submit the following program:

```
proc fontreg;
    fontpath "%sysget(systemroot)\fonts";
run;
```

The %SYSGET macro retrieves the value of the Windowing environment variable SYSTEMROOT, and resolves to the location of your system directory. The fonts subdirectory is located one level below the system directory.

Optional Argument

fileref

specifies a fileref to use with the FONTPATH statement.

REMOVE Statement

Removes a font family, all fonts of a particular type (such as TrueType or Type1), or all fonts from the Core\Printing\Freetype\Fonts location of the SAS registry.

Syntax

REMOVE *'family-name'* | *'alias'* | *family-type* | *_ALL_*;

Required Arguments

family-name

specifies the family name of the font that you want to remove from the Core\Printing\Freetype\Fonts key in the SAS registry. Enclose *family-name* in quotation marks if the value contains one or more spaces.

alias

specifies an alternative name, usually in a shortened form, for *family-name*. Enclose the alias name in quotation marks if the value contains one or more spaces.

Note The valid values that can be specified as an alias are listed in the Core\Printing\Alias\Fonts\Freetype key in the SAS registry.

family-type

specifies the name of a font type (such as TrueType or Type1) that SAS supports and that you want removed from the SAS registry.

Note: The font type is not removed from the operating system location in which they reside. The registration of the font type from the SAS registry is removed so that SAS does not recognize the fonts.

ALL

specifies that all font families in the Core\Printing\Freetype\Fonts key in the SAS registry will be deleted.

Details

Removing Fonts from the Registry

The REMOVE statement removes a font family, all fonts of a particular type, or all fonts from the `Core\Printing\Freetype\Fonts` location in the SAS registry. If you specify the USESASHELP procedure option, then fonts are removed from the Sashelp portion of the registry. If you do not specify USESASHELP, then fonts are removed from the Sasuser portion of the registry. Removal from the Sasuser portion of the registry is the default.

Note that when you specify the *family-name* argument in the REMOVE statement, SAS removes font families rather than individual fonts within the family. For example, you might register several fonts within the **Arial** family. When you use the `REMOVE Arial;` statement, all fonts in the **Arial** family are removed from the registry. Similarly, when you specify the *family-type* argument and use the `REMOVE Type1;` statement, all Type1 font families are removed from the registry.

The Order in Which Fonts Are Added or Removed

Fonts are removed from the SAS registry before any fonts are added or replaced in the registry using other procedure statements. The REMOVE statement removes a font family from the registry as soon as the statement is processed. Other font statements, such as FONTFILE, FONTPATH, TRUETYPE, and TYPE1, are processed in the order in which they are received. The font information is stored until all of the statements are processed. SAS then updates the registry.

Searching for a Font That Is Specified in the REMOVE Statement

If the name that you specify in a REMOVE statement does not exist, then SAS adds a font tag prefix (for example, <ttf>) to the specified name to determine whether it exists in the SAS registry. For example, if you specify **Arial**, SAS uses the <ttf> prefix tag and first searches for a TrueType font type so that it can be removed from the registry. If the search is not successful, then SAS uses the <at1> prefix tag and searches for a Type1 font type so that it can be removed from the registry.

When SAS Is Unable to Remove a Font Family

If SAS is unable to remove a font family after processing the information in the `_ALL_`, *family-type*, or *family-name* arguments, then SAS looks in the `Core\Printing\Alias\Fonts\Freetype` key in the SAS registry to determine whether the specified value is an alias. If the specified value exists as an alias in this key, then SAS deletes the font family that corresponds to the alias and deletes the alias as well. For example, if an alias of **Test** refers to the **Arial** font family, and you specify the `REMOVE test;` statement with PROC FONTREG, then SAS determines that **Test** is an alias for **Arial**. SAS removes the **Arial** font family from the `Core\Printing\Freetype\Fonts` key and the **Test** alias from the `Core\Printing\Alias\Fonts\Freetype` key in the SAS registry.

If SAS is unable to remove a font family at this point, then SAS writes a message to the log indicating that the specified value in the REMOVE statement is invalid.

TRUETYPE Statement

Specifies one or more directories to be searched for TrueType font files.

See: [“Example 3: Replacing Existing TrueType Font Files from a Directory” on page 861](#)

Syntax

TRUETYPE <fileref> 'directory' <...'directory'>;

Required Argument

directory

specifies a directory to search. Only files that are recognized as valid TrueType font files are processed. Each directory must be enclosed in quotation marks. If you specify more than one directory, then you must separate the directories with a space.

Optional Argument

fileref

specifies a fileref to use with the TRUETYPE statement.

TYPE1 Statement

Specifies one or more directories to be searched for valid Type1 font files.

Syntax

TYPE1 <fileref> 'directory' <...'directory'>;

Required Argument

directory

specifies a directory to search. Only files that are recognized as valid Type1 font files are processed. Each directory must be enclosed in quotation marks. If you specify more than one directory, then you must separate the directories with a space.

Optional Argument

fileref

specifies a fileref to use with the TYPE1 statement.

OPENTYPE Statement

Specifies one or more directories to be searched for valid OpenType font files.

Syntax

OPENTYPE <fileref> 'directory' <...!directory'>;

Required Argument

directory

specifies a directory to search. Only files that are recognized as valid OpenType font files are processed. Each directory must be enclosed in quotation marks. If you specify more than one directory, then you must separate the directories with a space.

Optional Argument

fileref

specifies a fileref to use with the OPENTYPE statement.

Examples: FONTREG Procedure

Example 1: Adding a Single Font File

Features: FONTFILE statement

Details

This example shows how to add a single font file to the SAS registry. The FONTFILE statement specifies the complete path to a single font file.

Program

```
proc fontreg;  
    fontfile '<ttf> Arial';  
run;
```

Log

Example Code 25.1 Adding a Single Font File to the SAS Registry

```
SUMMARY:  
Files processed: 1  
Unusable files: 0  
Files identified as fonts: 1  
Fonts that were processed: 1  
Fonts replaced in the SAS registry: 0  
Fonts added to the SAS registry: 1  
Fonts that could not be used: 0  
Font Families removed from SAS registry: 0
```

Example 2: Adding All Font Files from Multiple Directories

Features: MSGLEVEL= option
FONTPATH statement

Details

This example shows how to add all valid font files from two different directories and how to write detailed information to the SAS log.

Program

```
proc fontreg msglevel=verbose;  
fontpath 'your-font-directory-1' 'your-font-directory-2';  
run;
```

Program Description

Write complete details to the SAS log. The MSGLEVEL=VERBOSE option writes complete details about what fonts were added, what fonts were not added, and what font files were not understood.

```
proc fontreg msglevel=verbose;
```

Specify the directories to search for valid fonts. You can specify more than one directory in the FONTPATH statement. Each directory must be enclosed in quotation marks. If you specify more than one directory, then you must separate the directories with a space.

```
fontpath 'your-font-directory-1' 'your-font-directory-2';  
run;
```

Log

Example Code 25.2 Messages from Adding All Font Files from Multiple Directories

```

1  proc fontreg msglevel=verbose;
2  fontpath 'your-font-directory-1'
3          'your-font-directory-2';
4  run;

ERROR: FreeType base module FT_New_Face -- unknown file format.
ERROR: A problem was encountered with file
"your-font-directory-2\MODERN.FON".

. . . more log entries . . .

WARNING: The "Sasfont" font in file
        "your-font-directory-2\SAS1252.FON" is non-scalable.      Only scalable
fonts are supported.

. . . more log entries . . .

NOTE: The font "Albertus Medium" (Style: Regular, Weight: Normal) has been
      added to the SAS Registry at
      [CORE\PRINTING\FREETYPE\FONTS\<ttf>Albertus Medium]. Because it is a
      TRUETYPE font, it can be referenced as "Albertus Medium" or
      "<ttf>Albertus Medium" in SAS. The font resides in file
      "your-font-directory-1\albr55w.ttf".

. . . more log entries . . .

WARNING: The font "Georgia" (Style: Regular, Weight: Normal) will not be added
because it already exists in the "<ttf>Georgia" font family of the SAS Registry.

. . . more log entries . . .

SUMMARY:
  Files processed: 138
  Unusable files: 3
  Files identified as fonts: 135
  Fonts that were processed: 135
  Fonts replaced in the SAS registry: 0
  Fonts added to the SAS registry: 91
  Fonts that could not be used: 44
  Font Families removed from SAS registry: 0

NOTE: PROCEDURE FONTREG used (Total process time):
      real time          27.81 seconds
      cpu time           1.18 seconds

```

Example 3: Replacing Existing TrueType Font Files from a Directory

Features: MODE= option
 TRUETYPE statement

Details

This example reads all the TrueType fonts in the specified directory and replaces the ones that already exist in the SAS registry.

Program

```
proc fontreg mode=replace;
    truetype 'your-font-directory';
run;
```

Program Description

Replace existing fonts only. The MODE=REPLACE option limits the action of the procedure to replacing fonts that are already defined in the SAS registry. New fonts will not be added.

```
proc fontreg mode=replace;
```

Specify a directory that contains TrueType font files. Files in the directory that are not recognized as being TrueType font files are ignored.

```
    truetype 'your-font-directory';
run;
```

Log

Example Code 25.3 Replacing Existing TrueType Font Files from a Directory

```
SUMMARY:
Files processed: 49
Unusable files: 3
Files identified as fonts: 46
Fonts that were processed: 40
Fonts replaced in the SAS registry: 40
Fonts added to the SAS registry: 0
Fonts that could not be used: 0
Font Families removed from SAS registry: 0
```

FORMAT Procedure

Overview: <i>FORMAT Procedure</i>	863
What Does the FORMAT Procedure Do?	864
Format Encodings in SAS Viya Platform	864
What Are Formats and Informats?	865
How Are Formats and Informats Associated with a Variable?	865
Concepts: <i>FORMAT Procedure</i>	866
Associating Informats and Formats with Variables	866
Storing Informats and Formats	868
Printing Informats and Formats	870
Using Formats in a CAS Session	871
A Binary Search Determines the User-Defined Format or Informat for a Value	871
Syntax: <i>FORMAT Procedure</i>	872
PROC FORMAT Statement	873
EXCLUDE Statement	879
INVALU Statement	880
PICTURE Statement	887
SELECT Statement	909
VALUE Statement	910
Usage: <i>FORMAT Procedure</i>	918
Specifying Values or Ranges	918
Using a Function to Format Values	921
Results: <i>FORMAT Procedure</i>	923
Output Control Data Set	923
Input Control Data Set	927
Procedure Output	928
Examples: <i>FORMAT Procedure</i>	931
Example 1: Create a Format Library in a CAS Session	931
Example 2: Create the Example Data Set	937
Example 3: Creating a Picture Format	938
Example 4: Creating a Picture Format for Large Dollar Amounts	940
Example 5: Filling a Picture Format	943
Example 6: Create a Date or Time Format with Directives	945
Example 7: Change the 24-Hour Clock to 00:00:01–24:00:00	947
Example 8: Creating a Format for Character Values	948
Example 9: Creating a Format for Missing and Nonmissing Variable Values	950
Example 10: Creating an Informat Using Perl Regular Expressions	953

Example 11: Writing a Format for Dates Using a Standard SAS Format and a Color Background	955
Example 12: Converting Raw Character Data to Numeric Values	958
Example 13: Creating a Format from a CNTLIN= Data Set	961
Example 14: Creating an Informat from a CNTLIN= Data Set	966
Example 15: Printing the Description of Informats and Formats	971
Example 16: Retrieving a Permanent Format	973
Example 17: Writing Ranges for Character Strings	975
Example 18: Creating a Format in a non-English Language	978
Example 19: Creating a Locale-Specific Format Catalog	981
Example 20: Creating a Function to Use as a Format	986
Example 21: Using a Format to Create a Drill-down Table	989

Overview: FORMAT Procedure

What Does the FORMAT Procedure Do?

The FORMAT procedure enables you to define your own informats and formats for variables. In addition, you can perform these actions:

- print the parts of a catalog that contain informats or formats
- store descriptions of informats or formats in a SAS data set
- use a SAS data set to create informats or formats.

Format Encodings in SAS Viya Platform

SAS Viya platform supports only UTF-8 encoding.

When you store formats in a library, SAS uses the session encoding in which the formats were created. If the original encoding is not UTF-8, then truncation of characters might occur if you convert the format library to an encoding that requires more bytes to represent the characters.

Note: Moving format libraries between previous versions of SAS and CAS might have some risk. SAS recommends that you use CNTLOUT data sets to reduce this risk.

For more information about using format libraries in SAS Viya platform, see [Migrating Data to UTF-8 for the SAS Viya Platform](#) and [SAS Viya FAQ for Processing UTF-8 Data](#).

What Are Formats and Informats?

Informats determine how raw data values are read and stored. Formats determine how variable values are printed. For simplicity, this section uses the terminology the informat converts and the format prints.

Informats and formats tell SAS the data's type (character or numeric) and form (such as how many bytes it occupies; decimal placement for numbers; how to handle leading, trailing, or embedded blanks and zeros; and so on). SAS provides informats and formats for reading and writing variables. For a thorough description of informats and formats that SAS provides, see [SAS Formats and Informats: Reference](#).

With informats, you can do the following:

- Convert a number to a character string (for example, convert 1 to **YES**).
- Convert a character string to a different character string (for example, convert **'YES'** to **'OUI'**).
- Convert a character string to a number (for example, convert **YES** to 1).
- Convert a number to another number (for example, convert 0–9 to 1, 10–100 to 2, and so on).

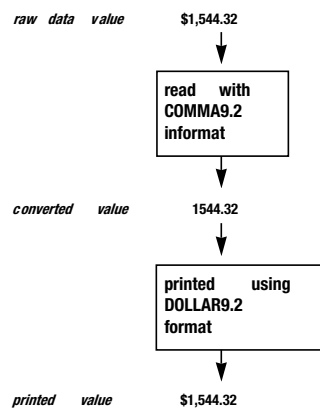
Note: User-defined informats read-only character data. They can convert character values into real numeric values, but they cannot convert real numbers into characters.

With formats, you can do the following:

- Print numeric values as character values (for example, print 1 as **MALE** and 2 as **FEMALE**).
- Print one character string as a different character string (for example, print **YES** as **OUI**).
- Print numeric values using a template (for example, print 9458763450 as **945-876-3450**).

How Are Formats and Informats Associated with a Variable?

The following figure summarizes what occurs when you associate an informat and format with a variable. The `COMMAw.d` informat and the `DOLLARw.d` format are provided by SAS.

Figure 26.1 Associating an Informat and a Format with a Variable

In the figure, SAS reads the raw data value that contains the dollar sign and comma. The COMMA9.2 informat ignores the dollar sign and comma and converts the value to 1544.32. The DOLLAR9.2 format prints the value, adding the dollar sign and comma. For more information about associating informats and formats with variables, see [“Associating Informats and Formats with Variables”](#) on page 866.

Concepts: FORMAT Procedure

Associating Informats and Formats with Variables

Methods of Associating Informats and Formats with Variables

The following table summarizes the different methods for associating informats and formats with variables.

Table 26.1 *Associating Informats and Formats with Variables*

Step	Informats	Formats
In a DATA step	Use the ATTRIB or INFORMAT statement to permanently associate an informat with a variable. Use the INPUT function or INPUT statement to associate the informat with the variable only for the duration of the DATA step.	Use the ATTRIB or FORMAT statement to permanently associate a format with a variable. Use the PUT function or PUT statement to associate the format with the variable only for the duration of the DATA step.
In a PROC step	The ATTRIB and INFORMAT statements are valid in Base SAS procedures. However, in Base SAS software, typically you do not assign informats in PROC steps because the data has already been read into SAS variables.	Use the ATTRIB statement or the FORMAT statement to associate formats with variables. If you use either statement in a procedure that produces an output data set, then the format is permanently associated with the variable in the output data set. If you use either statement in a procedure that does not produce an output data set or modify an existing data set, the statement associates the format with the variable only for the duration of the PROC step.

Differences between the FORMAT Statement and PROC FORMAT

Do not confuse the FORMAT statement with the FORMAT procedure. The FORMAT and INFORMAT statements associate an existing format or informat (either standard SAS or user-defined) with one or more variables. PROC FORMAT creates user-defined formats or informats.

Assigning Formats and Informats to a Variable

Assigning your own format or informat to a variable is a two-step process:

- 1 creating the format or informat with the FORMAT procedure
- 2 assigning the format or informat with the ATTRIB, FORMAT, or INFORMAT statements, or the INPUT or PUT functions

For complete documentation on the ATTRIB, INFORMAT, and FORMAT statements, see [SAS DATA Step Statements: Reference](#). For complete documentation on the

INPUT and PUT functions, see [SAS Functions and CALL Routines: Reference](#). For more information and example of using formats in Base SAS procedures, see [“Procedure Concepts”](#).

Storing Informats and Formats

Format Catalogs

PROC FORMAT stores user-defined informats and formats as entries in SAS catalogs.¹ You use the LIBRARY= option in the PROC FORMAT statement to specify the catalog. If you omit the LIBRARY= option, then formats and informats are stored in the Work.Formats catalog. If you specify LIBRARY=*libref* but do not specify a catalog name, then formats and informats are stored in the *libref*.FORMATS catalog. Note that this use of a one-level name differs from the use of a one-level name elsewhere in SAS. With the LIBRARY= option, a one-level name indicates a library; elsewhere in SAS, a one-level name indicates a file in the WORK library.

The name of the catalog entry is the name of the format or informat. The entry types are as follows:

- FORMAT for numeric formats
- FORMATC for character formats
- INFMT for numeric informats
- INFMTC for character informats

Temporary Informats and Formats

Informats and formats are temporary when they are stored in a catalog in the WORK library. If you omit the LIBRARY= option, then PROC FORMAT stores the informats and formats in the temporary catalog Work.Formats. You can retrieve temporary informats and formats only in the same SAS session or job in which they are created. To retrieve a temporary format or informat, simply include the name of the format or informat in the appropriate SAS statement. SAS automatically looks for the format or informat in the Work.Formats catalog.

1. Catalogs are a type of SAS file and reside in a SAS library. If you are unfamiliar with the types of SAS files or the SAS library structure, then see the section on SAS files in *SAS Programmer's Guide: Essentials*.

Permanent Informats and Formats

If you want to use a format or informat that is created in one SAS job or session in a subsequent job or session, then you must permanently store the format or informat in a SAS catalog.

You permanently store informats and formats by using the LIBRARY= option in the PROC FORMAT statement. See the discussion of the LIBRARY= option in the [PROC FORMAT Statement on page 873](#).

Accessing Permanent Informats and Formats

After you have permanently stored an informat or format, you can use it in later SAS sessions or jobs. If you associate permanent informats or formats with variables in a later SAS session or job, then SAS must be able to access the informats and formats. Thus, you must use a LIBNAME statement to assign a libref to the library that stores the catalog that stores the informats or formats.

SAS uses one of two methods when searching for user-defined formats and informats:

- By default, SAS always searches a library that is referenced by the Library libref for a FORMATS catalog. If you have only one format catalog, then do the following:
 - 1 Assign the Library libref to a SAS library in the SAS session in which you are running the PROC FORMAT step.
 - 2 Specify the option `library=library` in the PROC FORMAT statement. PROC FORMAT stores the informats and formats that are defined in that step in the Library.Formats catalog.
 - 3 In the SAS program that uses your user-defined formats and informats, include a LIBNAME statement to assign the Library libref to the library that contains the permanent format catalog.
- If you have more than one format catalog, or if the format catalog is named something other than Formats, then do the following:
 - 1 Assign a libref to a SAS library in the SAS session in which you are running the PROC FORMAT step.
 - 2 Specify the option `library=libref` or `library=libref.catalog` in the PROC FORMAT step, where *libref* is the libref that you assigned in step 1.
 - 3 In the SAS program that uses your user-defined formats and informats, use the FMTSEARCH= option in an OPTIONS statement, and include *libref* or *libref.catalog* in the list of format catalogs.

The syntax for specifying a list of format catalogs to search is

OPTIONS FMTSEARCH=(*catalog-specification-1*<*catalog-specification-2* ... >);

Each *catalog-specification* can be *libref* or *libref.catalog*. If only *libref* is specified, then SAS assumes that the catalog name is *Formats*.

When searching for a format or informat, SAS always searches in *Work.Formats* first, and then *Library.Formats*, unless one of them appears in the *FMTSEARCH=* list. SAS searches the catalogs in the *FMTSEARCH=* list in the order in which they are listed until the format or informat is found.

For more information, see [“FMTSEARCH= System Option” in SAS System Options: Reference](#). For an example that uses the *LIBRARY=* and *FMTSEARCH=* options together, see [“Example 17: Writing Ranges for Character Strings” on page 975](#).

Missing Informats and Formats

If you reference an informat or format that SAS cannot find, then you receive an error message and processing stops unless the SAS system option *NOFMTERR* is in effect. When *NOFMTERR* is in effect, SAS uses the *w.* or *\$w.* default format to print values for variables with formats that it cannot find. For example, to use *NOFMTERR*, use this *OPTIONS* statement:

```
options nofmterr;
```

For more information, see [“FMTERR System Option” in SAS System Options: Reference](#).

If SAS encounters a missing variable to format using a user-defined format and the *MISSING=* system option defines a character to be printed for missing values, the missing value is determined as follows:

- If the user-defined format or informat has a value-range-set for missing values, the missing value is defined by the user-defined format.
- If the user-defined format does not have a value-range-set defined for missing values, the missing value is defined by the *MISSING=* system option. The default value for the *MISSING=* system option is *.* (period).

Printing Informats and Formats

The output that is provided when you use the *FMTLIB* option in the *PROC FORMAT* statement is intended to present a brief view of the informat and format values.

Instead of using the *FMTLIB* option, you can use the *CNTLOUT=* option to create an output data set that stores information about informats and formats. You can then use *PROC PRINT* or *PROC REPORT* to print the data set. In this case, labels are not truncated.

Note: You can use data set options to keep or drop references to additional variables that were added by using the *CNTLOUT=* option.

Using Formats in a CAS Session

PROC FORMAT supports creating format libraries in catalogs in a SAS client session and loading format libraries to a SAS Cloud Analytic Services (CAS) session. When you use the SESSREF system option to specify a CAS session, and you also specify the CASFMTLIB option, then PROC FORMAT loads the format libraries to the specified CAS session. Otherwise, PROC FORMAT stores the format libraries where the SAS client session is running. For more information, see [“CASFMTLIB=*name*” on page 874](#).

The SAS client session and the CAS session can interact through the session reference that you establish with the SESSREF system option or the SESSREF argument in the CASLIB statement. When you use a SAS language element that can take advantage of processing in CAS, the session reference identifies where that processing should occur. If you do not specify a session reference, then processing occurs in the client session. If the language element is not supported in CAS, then processing occurs in the client session.

For more information about SESSREF, see:

- [“SESSREF= System Option” in SAS System Options: Reference](#)
- [“CASLIB Statement” in SAS Cloud Analytic Services: User's Guide](#)

For more information about using formats in a CAS session, see:

- [“Example 1: Create a Format Library in a CAS Session” on page 931](#)
- [“Using User-Defined Formats in SAS Cloud Analytic Services” in SAS Cloud Analytic Services: User-Defined Formats](#)

A Binary Search Determines the User-Defined Format or Informat for a Value

PROC FORMAT uses a binary search to determine the correct user-defined format or informat to use for a value. In comparison, using IF-THEN/ELSE statements is essentially a sequential search for a value. Because of the search method that the binary search uses, PROC FORMAT can provide a more efficient search for making a large number of comparisons.

Here are some user-defined format values that could be written using PROC FORMAT:

```
1= 'Yes'
2= 'No'
3= 'Possibly'
```

If you use these IF-THEN/ELSE statements, SAS begins searching with the first value in the range, $x=1$, and steps through the values until it finds a matching value.

This type of search can be more efficient if the value that you want is near the beginning of the range of values.

```
if x=1 then label='Yes';
else if x=2 then label='No';
else if x=3 then label='Possibly';
```

If you are searching for the value 3, there will be 3 comparisons before a match is found. If you are searching for the value 1, there will be only one comparison made.

If you use the VALUE statement of PROC FORMAT, SAS begins searching at the middle value in the range. In this example, SAS begins by comparing the value to the middle range value, 2='No'. It then compares the value to the higher range, value 3="Possibly", and then compares the value to the lower range value, 1="Yes".

```
value
  1="Yes"
  2="No"
  3="Possibly";
```

A binary search like PROC FORMAT uses is more efficient when the range has a large number of values to search.

Syntax: FORMAT Procedure

Restrictions:	You cannot use a SELECT statement and an EXCLUDE statement within the same PROC FORMAT step. When the CASFMTLIB option is specified, the SELECT and EXCLUDE statements ignore format libraries in SAS Cloud Analytics Services (CAS) sessions and refer only to catalogs. Formats that are not enabled for threaded-kernel processing are not written to CAS. Informats cannot be written to a CAS session. Informats included in your SAS code are ignored. Formats that use functions-as-labels or formats-as-labels cannot be written to CAS.
Tips:	User-defined format names cannot end in a number. For more information, see “User-Defined Formats” in SAS Formats and Informats: Reference and “SAS Names” in SAS Programmer’s Guide: Essentials. You can use appropriate global statements with this procedure. See “SAS Names” in SAS Programmer’s Guide: Essentials for a list. For information about generating a list of user-defined formats that are related to a SAS library, see the documentation for the %UDFSEL macro in <i>SAS Cloud Analytic Services: User-Defined Formats</i>.
See:	“Using Formats in a CAS Session” on page 871 PROC FORMAT <options>;

```

EXCLUDE entry(s);
INVALUE <$>name <(informat-options)> <value-range-set(s)>;
PICTURE name <(format-options)>
    <value-range-set-1 <(picture-1-options)>
    <value-range-set-2 <(picture-2-options)>> ...>;
SELECT entry(s);
VALUE <$>name <(format-options)> <value-range-set(s)>;

```

Statement	Task	Example
PROC FORMAT	Define formats and informats for variables	Ex. 3, Ex. 13, Ex. 15, Ex. 16
EXCLUDE	Exclude catalog entries from processing by the FMTLIB and CNTLOUT= options	
INVALUE	Create an informat for reading and converting raw data values	Ex. 12
PICTURE	Create a template for printing numbers	Ex. 3, Ex. 5, Ex. 18
SELECT	Select catalog entries for processing by the FMTLIB and CNTLOUT= options	Ex. 15
VALUE	Create a format that specifies character strings to use to print variable values	Ex. 8, Ex. 11, Ex. 17

PROC FORMAT Statement

Creates user-specified formats and informats for variables.

Tips: User-defined format names cannot end in a number. For more information, see “User-Defined Formats” in *SAS Formats and Informats: Reference* and “SAS Names” in *SAS Programmer’s Guide: Essentials*.

You can use data set options with the CNTLIN= and CNTLOUT= data set options. See “Language Concepts” for a list.

Moving catalogs between previous versions of SAS and CAS might have some risk. SAS recommends that you use CNTLOUT data sets to reduce this risk.

Examples:

- “Example 3: Creating a Picture Format” on page 938
- “Example 13: Creating a Format from a CNTLIN= Data Set” on page 961
- “Example 15: Printing the Description of Informats and Formats” on page 971
- “Example 16: Retrieving a Permanent Format” on page 973

Syntax

PROC FORMAT *<options>*;

Summary of Optional Arguments

CASFMTLIB=*'name'*

adds a format library to a CAS session.

CNTLIN=*input-control-SAS-data-set*

specifies a SAS data set from which PROC FORMAT builds informats or formats.

CNTLOUT=*output-control-SAS-data-set*

creates a SAS data set that stores information about informats or formats that are contained in the catalog specified in the LIBRARY= option.

FMTLIB

prints information about informats or formats in the catalog that is specified in the LIBRARY= option.

LIBRARY=*libref<.catalog>*

specifies a SAS library or catalog that contains the informats or formats that you are creating in the PROC FORMAT step.

LOCALE

specifies to create a format catalog that corresponds to the current SAS locale.

MAXLABELN=*number-of-characters*

specifies the number of characters of the informatted or formatted value that appear in PROC FORMAT output.

MAXSELEN=*number-of-characters*

specifies the number of characters of the start and end values that appear in the PROC FORMAT output.

NOREPLACE

prevents a new informat or format from replacing an existing one of the same name.

PAGE

prints information about each format and informat in the catalog.

Optional Arguments

CASFMTLIB=*'name'*

adds a format library to a CAS session.

You can specify the CASFMTLIB option only in an active SAS Cloud Analytic Services (CAS) session. PROC FORMAT connects to the CAS session and loads a format library. If the format library already exists in the CAS session, then SAS updates it. SAS also appends the format library to the search list for any subsequent referencing by procedures that are operating in CAS in that session. That is, if a format library already exists and you create a new library with

CASFMTLIB, then the new library is appended to the search order. The library name should be a one-level name that does not contain any slashes.

Note: A CAS session can have more than one format library. The libraries are available only to that CAS session. For information about using a CAS action to promote a format library to a global scope, see [“Promote format library” in SAS Viya Platform: System Programming Guide](#).

You can specify additional CAS sessions with the SESSREF= option. For information about the SESSREF= option and other CAS language elements, see [SAS Cloud Analytic Services: User's Guide](#).

SAS formats are available in the local SAS client regardless of whether you add them to a format library in CAS. When the CASFMTLIB option is specified, the EXCLUDE and SELECT statements are applied to the local SAS session format catalogs, not to the CAS session format library. Informats cannot be loaded into a CAS session. If you specify an INVALUE statement with CASFMTLIB, then a note is written to the log and nothing is written to the CAS format library.

Restriction The format library name that you specify with the CASFMTLIB option cannot have a length of more than 63 characters.

Notes The specified value of the CASFMTLIB option must be enclosed in single or double quotation marks.

The library name that you specify is case-sensitive. It has to be a valid SAS name, and cannot contain blank spaces.

Formats defined in the VALUE or PICTURE statements are also written to the format catalog that is specified by the LIBRARY= option. If you do not specify a library, then SAS uses the WORK.FORMATS library.

Tip You can use the CAS action `addFmtLib=fmtsearch` to control the order in which SAS searches for format libraries. For more information, see “Manage User-Defined Formats with CAS Actions” in [SAS Cloud Analytic Services: User's Guide](#)

See [“Using Formats in a CAS Session” on page 871](#)

CNTLIN=input-control-SAS-data-set

specifies a SAS data set from which PROC FORMAT builds informats or formats.

CNTLIN= builds formats and informats without using a VALUE, PICTURE, or INVALUE statement. If you specify a one-level name, then the procedure searches only the default library (either the WORK library or USER library) for the data set, regardless of whether you specify the LIBRARY= option.

Notes When using PROC FORMAT with a CNTLIN data set the START and END columns must have the same length. If the lengths are different, an error might occur.

LIBRARY= can point to either a library or a catalog. If only a libref is specified, a catalog name of FORMATS is assumed.

Tip A common source for an input control data set is the output from the CNTLOUT= option of another PROC FORMAT step.

See [“Input Control Data Set” on page 927](#)

Example [“Example 13: Creating a Format from a CNTLIN= Data Set” on page 961](#)

CNTLOUT=output-control-SAS-data-set

creates a SAS data set that stores information about informats or formats that are contained in the catalog specified in the LIBRARY= option. If you are creating an informat or format in the same step that the CNTLOUT= option appears, then the informat or format that you are creating is included in the CNTLOUT= data set.

If you specify a one-level name, then the procedure stores the data set in the default library (either the WORK library or the USER library), regardless of whether you specify the LIBRARY= option.

If you issue CNTLOUT= with an ENCODING option to create an output data set that has a different encoding, Cross Environment Data Access (CEDA) might issue a truncation error, SAS stops processing, and the CNTLOUT= data set is created with 0 observations. SAS writes an error to the log such as the following about the truncation of data:

```
ERROR: Some character data was lost during transcoding in the dataset
      WORK.TEST. Either the data contains characters that are not
      representable in the new encoding or truncation occurred during
      transcoding.
```

Suppose you are using a data set that contains monetary values in Euros. You are using the Wlatin1 session encoding, and you specify UTF-8 encoding for the CNTLOUT= data set. In Wlatin1 the LABEL variable is predetermined to be 5 bytes long and have a value of €1234 (in hexadecimal representation, '803132334'x). When you attempt to store the variable in the CNTLOUT= data set with UTF-8 encoding, the length of that string must be 7 bytes (in hexadecimal representation, 'E282AC3132334'x). The two additional bytes are needed in UTF-8 encoding for the Euro sign. Without the additional two bytes, the string is truncated. You can use the %COPY_TO_NEW_ENCODING macro to prevent this error. For information about the %COPY_TO_NEW_ENCODING macro, see [“Avoiding Character Truncation Using the %COPY_TO_NEW_ENCODING Macro” in SAS National Language Support \(NLS\): Reference Guide](#).

The macro examines the CNTLOUT data set and re-creates it in the new encoding with the necessary lengths. If a width is stored as part of the associated format, the value is not expanded by the CVP engine. The format can cause truncation when the formatted value is displayed.

The DEFAULT value for the format width in the CNTLOUT data set is the default width (in bytes) for the format. The user can specify DEFAULT=, MIN=, and MAX= when they create the format, or the default is computed based on the largest label. If the START, MIN, MAX, or LABEL variable, in characters, is

larger in UTF-8 encoding, then these widths are not expanded by the CVP engine. Use the %COPY_TO_NEW_ENCODING macro instead. For more information about using CNTLOUT= with PROC FORMAT to convert catalogs to UTF-8, see [“Converting Format Catalogs to UTF-8 Encoding” in *Moving and Accessing SAS Files*](#).

Note LIBRARY= can point to either library or a catalog. If only a libref is specified, SAS uses the catalog name FORMATS.

Tip You can use an output control data set as an input control data set in subsequent PROC FORMAT steps.

See [“Output Control Data Set” on page 923](#)

SAS Viya platform supports only UTF-8 encoding. For information about the encoding of your format catalogs in SAS Viya platform, see [Migrating Data to UTF-8 for the SAS Viya Platform](#) and [SAS Viya FAQ for Processing UTF-8 Data](#).

FMTLIB

prints information about informats or formats in the catalog that is specified in the LIBRARY= option. To get information about specific informats or formats, subset the catalog using the SELECT or EXCLUDE statement.

Note: FMTLIB is not supported for CASFMTLIB.

Interaction The PAGE option invokes FMTLIB.

Tips If your output from FMTLIB is not formatted correctly in the ODS LISTING destination, then try increasing the value of the LINESIZE= system option.

If you use a SELECT or EXCLUDE statement in your code without including a CNTLOUT= option, the FMTLIB option is automatically invoked and the FMTLIB report is generated. If necessary, this report can be suppressed by placing the following ODS EXCLUDE statement immediately before the PROC FORMAT statement:

```
ods exclude FORMAT;
```

Example [“Example 15: Printing the Description of Informats and Formats” on page 971](#)

LIBRARY=libref<.catalog>

specifies a SAS library or catalog that contains the informats or formats that you are creating in the PROC FORMAT step. The procedure stores these informats and formats in the catalog that you specify so that you can use them in subsequent SAS sessions or jobs.

Alias LIB=

Default If you omit the LIBRARY= option, then formats and informats are stored in the Work.Formats catalog. If you specify the LIBRARY=

option but do not specify a name for *catalog*, then formats and informats are stored in the *libref*.FORMATS catalog.

Note LIBRARY= can point to either a library or a catalog. If only a libref is specified, then SAS uses the catalog name FORMATS.

Tips SAS automatically searches Library.Formats. You might want to define and use the LIBRARY libref for your format catalog.

You can control the order in which SAS searches for format catalogs with the FMTSEARCH= system option. For more information, see [“FMTSEARCH= System Option” in SAS System Options: Reference](#).

See [“Storing Informats and Formats ” on page 868](#)

Example [“Example 3: Creating a Picture Format” on page 938](#)

LOCALE

specifies to create a format catalog that corresponds to the current SAS locale. The name of the catalog that SAS creates is the SAS library or catalog that is specified in the LIBRARY= option appended with the five-character POSIX locale value for the current SAS locale.

See For a list of POSIX locale values, see [“Locale-to-Encoding Mapping” in SAS National Language Support \(NLS\): Reference Guide](#).

Example If the SAS locale is German_Germany, the POSIX locale value is de_DE. Using the following PROC FORMAT statement, SAS creates the catalog mylib.formats_de_DE to store formats and informats created by this procedure:

```
proc format locale lib=mylib.formats;
```

MAXLABELN=*number-of-characters*

specifies the number of characters in the informatted or formatted value that you want to appear in the CNTLOUT= data set or in the output of the FMTLIB option. The FMTLIB option prints a maximum of 40 characters for the informatted or formatted value.

MAXSELEN=*number-of-characters*

specifies the number of characters in the start and end values that you want to appear in the CNTLOUT= data set or in the output of the FMTLIB option. The FMTLIB option prints a maximum of 16 characters for start and end values.

NOREPLACE

prevents a new informat or format from replacing an existing one of the same name. If you omit NOREPLACE, then the procedure warns you that the informat or format already exists and replaces it.

Note You can have a format and an informat of the same name.

PAGE

prints information about each format and informat in the catalog.

Interaction The PAGE option activates the FMTLIB option.

Tip In the ODS LISTING destination, the information about each format and informat appears on separate pages in the Output window.

EXCLUDE Statement

Excludes entries from processing by the FMTLIB and CNTLOUT= options.

Restrictions: Only one EXCLUDE statement can appear in a PROC FORMAT step. You cannot use a SELECT statement and an EXCLUDE statement within the same PROC FORMAT step. When the CASFMTLIB option is specified, the EXCLUDE statement ignores format libraries in CAS sessions and refers only to catalogs in the SAS session.

Syntax

EXCLUDE *entry(s)*;

Required Argument

entry(s)

specifies one or more catalog entries to exclude from processing. Catalog entry names are the same as the name of the informat or format that they store. Because informats and formats can have the same name, and because character and numeric informats or formats can have the same name, you must use certain prefixes when specifying informats and formats in the EXCLUDE statement. Follow these rules when specifying entries in the EXCLUDE statement:

- Precede names of entries that contain character formats with a dollar sign (\$).
- Precede names of entries that contain character informats with an at sign and a dollar sign (for example, @\$*entry-name*).
- Precede names of entries that contain numeric informats with an at sign (@).
- Specify names of entries that contain numeric formats without a prefix.

Details

Shortcuts to Specifying Names

You can use the colon (:) and hyphen (-) wildcard characters to exclude entries. For example, the following EXCLUDE statement excludes all formats or informats that begin with the letter a.

```
exclude a;;
```

In addition, the following EXCLUDE statement excludes all formats or informats that occur alphabetically between **apple** and **pear**, inclusive:

```
exclude apple-pear;
```

FMTLIB Output

If you use the EXCLUDE statement without either FMTLIB or CNTLOUT= in the PROC FORMAT statement, then the procedure invokes the FMTLIB option and you receive FMTLIB option output.

INVALUE Statement

Creates an informat for reading and converting raw data values.

See: [SAS Formats and Informats: Reference](#) for documentation on informats supplied by SAS.

Example: [“Example 12: Converting Raw Character Data to Numeric Values” on page 958](#)

Syntax

INVALUE <\$>*name* <(informat-options)> <value-range-set(s)>;

Summary of Optional Arguments

DEFAULT=*length*

specifies the default length of the informat.

FUZZ=*fuzz-factor*

specifies a fuzz factor for matching values to a range.

JUST

left-justifies all input strings before they are compared to ranges.

MAX=*length*

specifies a maximum length for the informat.

MIN=*length*

specifies a minimum length for the informat.

NOTSORTED

stores values or ranges in the order in which you define them.

REGEXP

REGEXPE

specifies that the preceding range is to be treated as a Perl regular expression.

UPCASE

upper cases all input strings before they are compared to ranges.

Control the input template.

value-range-set(s)

specifies the variable template for reading data.

Required Argument

name

names the informat that you are creating.

Restrictions A user-defined informat name cannot be the same as an informat name that is supplied by SAS.

"_C" and "_B"(both case insensitive) are reserved keys for format names. These keys cannot appear at the end of a format name. If format catalogs are migrated from SAS 9 with these suffixes, an error appears. The workaround is to recreate the format entry with a name that does not contain the _C or _B suffix .

Requirement The name must be a valid SAS name. A numeric informat name can be up to 31 characters in length; a character informat name can be up to 30 characters in length and cannot end in a number. If you are creating a character informat, then use a dollar sign (\$) as the first character. Adding the dollar sign to the name is why a character informat is limited to 30 characters.

Interaction The maximum length of an informat name is controlled by the [VALIDFMTNAME=](#) system option. See [SAS System Options: Reference](#) for details.

Tips Refer to the informat later by using the name followed by a period. However, do not use a period after the informat name in the INVALUE statement.

When SAS prints messages that refer to a user-written informat, the name is prefixed by an at sign (@). When the informat is stored, the at sign is prefixed to the name that you specify for the informat. The addition of the at sign to the name is why the name is limited to 31 or 30 characters. You need to use the at sign *only* when you are using the name in an EXCLUDE or SELECT statement; do not prefix the name with an at sign when you are associating the informat with a variable.

Optional Arguments

DEFAULT=length

specifies the default length of the informat. The value for DEFAULT= becomes the length of the informat if you do not give a specific length when you associate the informat with a variable.

Defaults For character informats, the length of the longest label

For numeric informats, 12 if you have numeric data to the left of the equal sign

For quoted strings, the length of the longest string

Note If you specify an invalid value for `DEFAULT=`, SAS ignores the value and writes an error to the log.

Tip As a best practice, if you specify an existing informat in a value-range set, always specify the `DEFAULT=` option.

FUZZ=fuzz-factor

specifies a fuzz factor for matching values to a range. If a number does not match or fall in a range exactly but comes within *fuzz-factor*, then the informat considers it a match. For example, the following `INVALUE` statement creates the `LEVELS.` informat, which uses a fuzz factor of .2:

```
invalue levels (fuzz=.2) 1='A'
                        2='B'
                        3='C' ;
```

`FUZZ=.2` means that if a variable value falls within .2 of a value on either end of the range, then the informat uses the corresponding formatted value to store the variable value. So the `LEVELS.` informat saves the value 2.1 as `B.`

Tips Specify `FUZZ=0` to save storage space when you use the `INVALUE` statement to create numeric informats.

Use a nonzero fuzz factor only with numbers that are very close but not an exact match. Ranges are stored internally in sorted order (unless the `NOTSORTED` option is used), in order to perform a binary search. When a fuzz-factor is added to the end of one range and subtracted from the beginning of the next range, and the ranges overlap, the results can be unpredictable. A value is placed in the first range that is a match in the binary search. The exclusion operator is insufficient to override this binary search algorithm. As a best practice, when you use the exclusion operator, set `FUZZ=0` or the `NOTSORTED` option.

A best practice is to use `FUZZ=0` when you use the `<` exclusion operator with numeric informats.

JUST

left-justifies all input strings before they are compared to ranges.

MAX=length

specifies a maximum length for the informat. When you associate the informat with a variable, you cannot specify a width greater than the `MAX=` value.

Default 40

Range 1–32767

Note If you specify an invalid value for `MAX=`, SAS ignores the value and writes an error to the log.

MIN=length

specifies a minimum length for the informat. If a CNTLIN= data set contains a value for MIN that rounds to 0 or less, SAS ignores the invalid value and substitutes 1 for it. When you specify the CNTLIN= data set in SAS code, SAS continues processing the code step and does not write an error to the log.

The following example specifies a data set that has a MIN value of -1 and uses CNTLIN= to call the data set on the subsequent PROC FORMAT statement.

```
data temp;
    fmtname='abc';
    start=1;
    label='xyz';
    min=-1;
run;

proc format cntlin=temp;
run;
```

If you specify an invalid value for MIN in code, SAS does not ignore the invalid value or substitute another value for it. SAS stops processing the code step and writes an error to the log.

The following example specifies min=-1 for the NEWABC data set, which causes SAS to stop processing and issue an error.

```
proc format;
    value newabc (min=-1) 1='yes';
run;
```

Default 1

Range 1-32767

NOTSORTED

stores values or ranges in the order in which you define them.

If you do not specify NOTSORTED, then values or ranges are stored in sorted order by default, and SAS uses a binary searching algorithm to locate the range that a particular value falls into. If you specify NOTSORTED, then SAS searches each range in the order in which you define them until a match is found.

Use NOTSORTED if one of the following is true:

- You know the likelihood of certain ranges occurring, and you want your informat to search those ranges first to save processing time.
- You want to preserve the order that you define ranges when you print a description of the informat using the FMTLIB option.
- You want to preserve the order that you define ranges when you use the ORDER=DATA option and the PRELOADFMT option to analyze class variables in PROC MEANS, PROC SUMMARY, or PROC TABULATE.

Do not use NOTSORTED if the distribution of values is uniform or unknown, or if the number of values is relatively small. The binary searching algorithm that SAS uses when NOTSORTED is not specified optimizes the performance of the search under these conditions.

SAS automatically sets the NOTSORTED option when you use the CPORT and CIMPORT procedures to transport informats or formats between operating environments with different standard collating sequences. This automatic setting of NOTSORTED can occur when you transport informats or formats between ASCII and EBCDIC operating environments. If this situation is undesirable, then do the following:

- Use the CNTLOUT= option in the PROC FORMAT statement to create an output control data set.
- Use the CPORT procedure to create a transport file for the control data set.
- Use the CIMPORT procedure in the target operating environment to import the transport file.
- In the target operating environment, use PROC FORMAT with the CNTLIN= option to build the formats and informats from the imported control data set.

REGEXP

REGEXPE

specifies that the preceding range is to be treated as a Perl regular expression. If you specify REGEXPE, the regular expression is expected to produce a modified result, as in using the substitute action.

During execution, all regular expressions are compiled and the input data is passed to the first expression to confirm a match. If there is a match, the corresponding label is used. If there is no match, the next range is compared. Ranges are not sorted and are processed in the order in which they were defined in the INVALUE statement or in the order in which they appear in the CNTLIN= data set.

The rules for regular expressions using the REGEXP option are the same as they are for the [PRXPARSE function](#) in the DATA step. The rules for the REGEXPE option are the same as they are for the [PRXCHANGE function](#).

Interaction If you are using a CNTLIN= data set, the HLO variable contains P for REGEXP and E for REGEXPE.

See [“Example 10: Creating an Informat Using Perl Regular Expressions” on page 953](#)

UPCASE

converts all raw data values to uppercase before they are compared to the possible ranges. If you use UPCASE, then make sure the values or ranges that you specify are in uppercase.

value-range-set(s)

specifies raw data and values that the raw data become. The *value-range-set(s)* can be one or more of the following:

value-or-range-1 < , *value-or-range-2* ... > = *informat* | [*existing-informat*]

The informat converts the raw data to the values of *informat* on the right side of the equal sign.

value-or-range

See [“Specifying Values or Ranges” on page 918](#).

informatted-value

is the value that you want the raw data in *value-or-range* to become. Use one of the following forms for *informatted-value*:

'character-string'

is a character string up to 32,767 characters long. Typically, *character-string* becomes the value of a character variable when you use the informat to convert raw data. Use *character-string* for *informatted-value* only when you are creating a character informat. If you omit the single or double quotation marks around *character-string*, then the INVALUE statement assumes that the quotation marks are there.

For hexadecimal literals, you can use up to 32,767 typed characters, or up to 16,382 represented characters at two hexadecimal characters per represented character.

number

is a number that becomes the informatted value. Typically, *number* becomes the value of a numeric variable when you use the informat to convert raw data. Use *number* for *informatted-value* when you are creating a numeric informat. The maximum for *number* depends on the host operating environment.

ERROR

treats data values in the designated range as invalid data. SAS assigns a missing value to the variable, prints the data line in the SAS log, and issues a warning message.

SAME

prevents the informat from converting the raw data as any other value. For example, the following GROUP. informat converts values 01 through 20 and assigns the numbers 1 through 20 as the result. All other values are assigned a missing value.

```
invalue group 01-20= _same_
              other= .;
```

existing-informat

is an informat that is supplied by SAS or an existing user-defined informat. The informat that you are creating uses the existing informat to convert the raw data that match *value-or-range* on the left side of the equal sign. If you use an existing informat, then enclose the informat name in square brackets (for example, [date9.]) or with parentheses and vertical bars (for example, ([date9.])). Do not enclose the name of the existing informat in single quotation marks.

Tip As a best practice, if you specify an existing informat in a value-range-set, always specify a default value by using the DEFAULT= option.

Examples

Example 1: Create a Character Informat for Raw Data Values

The \$GENDER. character informat converts the raw data values **F** and **M** to character values '1' and '2':

```
invalue $gender 'F'='1'
                'M'='2';
```

The dollar sign prefix indicates that the informat converts character data.

Example 2: Create Character and Numeric Values or a Range of Values

When you create numeric informats, you can specify character strings or numbers for *value-or-range*. For example, the TRIAL. informat converts any character string that sorts between **A** and **M** to the number 1 and any character string that sorts between **N** and **Z** to the number 2. The informat treats the unquoted range 1–3000 as a numeric range, which includes all numeric values between 1 and 3000:

```
invalue trial 'A'-'M'=1
              'N'-'Z'=2
              1-3000=3;
```

Example 3: Convert Input Strings to Uppercase and Left-justify

The EVALUATION. informat uses the UPCASE and JUST options to uppercase and left-justify all input strings before they are converted to numbers.

```
invalue evaluation (upcase just) 'O'=4
                                'S'=3
                                'E'=2
                                'C'=1
                                'N'=0;
```

Note: Options in the INVALUE statement must be specified in parentheses.

Example 4: Create an Informat Using _ERROR_ and _SAME_

The CHECK. informat uses _ERROR_ and _SAME_ to convert values of 1 through 4 and 99. All other values are invalid:

```
invalue check 1-4=_same_
              99=.
              other=_error_;
```

If you use a numeric informat to convert character strings that do not correspond to any values or ranges, then you receive an error message.

PICTURE Statement

Creates a template for printing numbers.

- Tips:**
- As a best practice, if you specify an existing format in a value-range-set, always specify a default value by using the [DEFAULT= option on page 912](#).
 - If you are using the DATATYPE= option, use the DEFAULT= option to set the default format width to be large enough to format these characters. Without setting the DEFAULT= option, the default width of a format is the width of the largest value to the right of the equation symbol.
 - The DEFAULT, FUZZ, MAX, MIN, MULTILABEL, NOTSORTED, and ROUND options are valid before the value range specification.
 - The DATATYPE, DECSEP, DIG3SEP, FILL, LANGUAGE, MULT, NOEDIT, and PREFIX options are valid in parentheses after the user-supplied value label.
 - The DATATYPE, DECSEP, DIG3SEP, FILL, LANGUAGE, MULT, NOEDIT, PREFIX, and ROUND options are valid only with the PICTURE statement.
- See:** [SAS Formats and Informats: Reference](#) and [SAS National Language Support \(NLS\): Reference Guide](#) for documentation about formats that are supplied by SAS.
- Examples:**
- [“Example 3: Creating a Picture Format” on page 938](#)
 - [“Example 5: Filling a Picture Format” on page 943](#)
 - [“Example 18: Creating a Format in a non-English Language” on page 978](#)

Syntax

```
PICTURE name <(format-options)>
<value-range-set-1 <(picture-1-options)>
<value-range-set-2 <(picture-2-options)>> ...>;
```

Summary of Optional Arguments

Control the attributes of each picture in the format

FILL=*'character'*

specifies a character that completes the formatted value.

MULTIPLIER=*n*

specifies a number to multiply the variable's value by before it is formatted.

NOEDIT

specifies that numbers are message characters rather than digit selectors.

PREFIX=*'prefix'*

specifies a character prefix to place in front of the formatted value.

Control the attributes of the format**DATATYPE=***DATE | TIME | DATETIME | DATETIME_UTIL*

enables the use of directives in the picture as a template to format date, time, or datetime values.

DECSEP=*'character'*

specifies the separator character for the fractional part of a number.

DEFAULT=*length*

specifies the default length of the picture.

DIG3SEP=*'character'*

specifies the three-digit separator character for a number.

FUZZ=*fuzz-factor*

specifies a fuzz factor for matching values to a range.

LANGUAGE=

specifies the language that is used for weekdays and months that you can substitute in a date, time, or datetime picture.

MAX=*length*

specifies a maximum length for the format.

MIN=*length*

specifies a minimum length for the format.

MULTILABELenables the assignment of labels to multiple *values-or-range* values that might have the same or overlapping values.**NOTSORTED**

stores values or ranges in the order in which you define them.

ROUND

rounds the value to the nearest integer before formatting.

Control the template for printing*value-range-set*

specifies one or more variable values and a template for printing those values.

Required Argument***name***

names the format that you are creating.

Restrictions A user-defined format cannot be the name of a format supplied by SAS.

"_C" and "_B"(both case insensitive) are reserved keys for format names. These keys cannot appear at the end of a format name. If format catalogs are migrated from SAS 9 with these suffixes, an error appears. The workaround is to recreate the format entry with a name that does not contain the _C or _B suffix .

Requirement The name must be a valid SAS name. A numeric format name can be up to 32 characters in length; a character format name can be up to 31 characters in length, not ending in a number. If you are creating a character format, you use a dollar sign (\$) as the first

character, which is why a character informat is limited to 31 characters. For information about SAS names, see “Rules for Most SAS Names” in *SAS Programmer’s Guide: Essentials*.

Interaction The maximum length of a format name is controlled by the `VALIDFMTNAME=` system option. See *SAS System Options: Reference* for details.

Tip Refer to the format later by using the name followed by a period. However, do not put a period after the format name in the `VALUE` statement.

Optional Arguments

DATATYPE=DATE | TIME | DATETIME | DATETIME_UTIL

enables the use of directives in the picture as a template to format date, time, or datetime values. Specify either `DATE`, `TIME`, `DATETIME`, or `DATETIME_UTIL` based on the directive that you use in the picture format. See the definition and list of [directives on page 896](#) in the description of *picture*.

Interaction `DATATYPE=DATETIME` results in datetime hours 00:00:00–23:59:59. `DATATYPE=DATETIME_UTIL` results in datetime hours between 00:00:01–24:00:00.

Tip If you format a numeric missing value, then the resulting label will be `ERROR`. Adding a clause to your program that checks for missing values can eliminate the `ERROR` label.

DEFAULT=length

specifies the default length of the picture. The value for `DEFAULT=` becomes the length of *picture* if you do not give a specific length when you associate the format with a variable.

Default The length of the longest picture value

Range 1–32767

Tip If you are using the `DATATYPE=` option, use the `DEFAULT=` option to set the default format width large enough to format these characters.

DECSEP='character'

specifies the separator character for the fractional part of a number.

Default . (a decimal point)

DIG3SEP='character'

specifies the three-digit separator character for a number.

Default , (a comma)

FILL='character'

specifies a character that completes the formatted value.

If the number of significant digits is less than the length of the format, then the format must complete, or fill, the formatted value:

- The format uses *character* to fill the formatted value if you specify zeros as digit selectors.
- The format uses zeros to fill the formatted value if you specify nonzero-digit selectors. The FILL= option has no effect.

If the picture includes other characters, such as a comma, which appear to the left of the digit selector that maps to the last significant digit placed, then the characters are replaced by the fill character or leading zeros.

Default ' ' (a blank)

Restrictions The FILL= option is not valid when you use a function to format a value.

Only single-byte characters are supported with the FILL option.

Interaction If you use the FILL= and PREFIX= options in the same picture, then the format places the prefix and then the fill characters.

Example [“Example 5: Filling a Picture Format” on page 943](#)

FUZZ=fuzz-factor

specifies a fuzz factor for matching values to a range. If a number does not match or fall in a range exactly but comes within *fuzz-factor*, on either end of the range, then the format considers it a match. For example, the following VALUE statement creates the LEVELS. format, which uses a fuzz factor of .2:

```
value levels (fuzz=.2) 1='A'
                    2='B'
                    3='C' ;
```

FUZZ=.2 means that if a variable value falls within .2 of a value on either end of the range, then the format uses the corresponding formatted value to print the variable value. The LEVELS. format formats the value 2.1 as B.

Default 1E-12 for numeric formats.

Tips Specify FUZZ=0 to save storage space when you use the VALUE statement to create numeric formats.

Use a nonzero fuzz factor only with numbers that are very close but not an exact match. If *fuzz-factor* is added to the end of one range and subtracted from the beginning of the next range, and the ranges overlap, the results can be unpredictable. A value is placed in the first range that is a match in a binary search.

A best practice is to use FUZZ=0 when you use the < exclusion operator with numeric formats.

LANGUAGE=

specifies the language that is used for weekdays and months that you can substitute in a date, time, or datetime picture. If you specify a language that is not supported or is invalid, English is used.

These are the valid values for the LANGUAGE= option:

Afrikaans	English	Macedonian	Spanish
Catalan	Finnish	Norwegian	Swedish
Croatian	French	Polish	Swiss_French
Czech	German	Portuguese	Swiss_German
Danish	Hungarian	Russian	
Dutch	Italian	Slovenian	

Defaults For single-byte character sets, the language that is specified by DFLANG= system option

For double-byte and UTF-8 character sets, the language that is specified by the LOCALE= system option

Tip To use a user-defined format in languages other than those that are supported by the LANGUAGE= option, set the LOCALE= system option to the locale for the language. In PROC FORMAT, do not specify the LANGUAGE= option. The language of a picture format is determined by the locale setting. For a list of locales, see [“Locale-to-Encoding Mapping” in SAS National Language Support \(NLS\): Reference Guide](#).

See [“DFLANG= System Option” in SAS National Language Support \(NLS\): Reference Guide](#)

MAX=length

specifies a maximum length for the format. When you associate the format with a variable, you cannot specify a width greater than the MAX= value.

Default 40

Range 1–32767

MIN=length

specifies a minimum length for the format.

Default 1

Range 1–32767

MULTILABEL

enables the assignment of labels to multiple *values-or-range* values that might have the same or overlapping values. The label is the formatted value that is determined by the picture definition on the right of the equal sign in a value-range-set. Here is an example of how MULTILABEL is used:

The following PICTURE statements show the two uses of the MULTILABEL option. In each case, number formats are assigned as labels. The first PICTURE statement assigns multiple labels to a single value. Multiple labels can also be assigned to a single range of values. The second PICTURE statement assigns labels to overlapping ranges of values. The MULTILABEL option enables the assignment of multiple labels to the overlapped values.

```

picture abc (multilabel)
    1000='9,999'
    1000='9999';

picture overlap (multilabel)
    /* without decimals */
    0-999='999'
    1000-9999='9,999'

    /* with decimals */
    0-9='9.999'
    10-99='99.99'
    100-999='999.9';

```

Only multilabel-enabled procedures such as PROC MEANS, PROC SUMMARY, and PROC TABULATE can use multiple labels. All other procedures and the DATA step recognize only the primary label.

The *primary label* for a given entry is the formatted value (based on the picture) that is assigned to the first value or range-of-values (left side of the equal sign) that matches or contains the entry when all values (on the left side of the equal sign) are ordered sequentially. Here is an example:

- In the first PICTURE statement, the primary label for 1000 is 1,000 because the picture 9,999 is the first value that is assigned to 1000. The secondary label for 1000 is 1000, based on the 9999 picture.
- In the second PICTURE statement, the primary label for 5 is 5.000 based on the 9.999 picture that is assigned to the range 0–9 because 0–9 is sequentially the first range of values that contain 5. The secondary label for 5 is 005 because the range 0–999 occurs in sequence after the range 0–9.

Consider carefully when you assign multiple labels to a value.

Unless you use the NOTSORTED option when you assign value-range-sets, SAS stores the value-range-sets in sorted order. This order can produce unexpected results when value-range-sets with the MULTILABEL format are processed. Here is an example:

In the second PICTURE statement, the primary label for 15 is 015, and the secondary label for 15 is 15.00 because the range 0–999 occurs in sequence before the range 10–99. If you want the primary label for 15 to use the 99.99 format, then you might want to change the range 10–99 to 0–99 in the PICTURE statement. The range 0–99 occurs in sequence before the range 0–999 and will produce the desired result.

Restriction The maximum number of labels that can be created for a single format or informat is 255.

MULTIPLIER=*n*

specifies a number to multiply the variable's value by before it is formatted. The value of the MULTIPLIER= option depends both on the result of the multiplication and on the digit selectors in the *picture* portion of the *value-range-set*. For example, the following PICTURE statement creates the MILLION. format, which formats the variable value 1600000 as \$1.6M:

```
picture million low-high='09.9M'
```

```
(prefix='$' mult=.00001);
```

1600000 is first multiplied by .00001, which equals 16. Note that there is a digit selector after the decimal. The value 16 is placed into the picture beginning on the right. The value 16 overlays 09.9, and results in 01.6. Leading zeros are dropped, and the final result is 1.6M.

If the value of low-high is equal to '000M', then the result would be 16M.

Alias	MULT=
Default	10^n , where n is the number of digits after the first decimal point in the picture. For example, suppose your data contains a value 123.456 and you want to print it using a picture of '999.999'. The format multiplies 123.456 by 10^3 to obtain a value of 123456, which results in a formatted value of 123.456.
Restrictions	The MULT= option is not valid when you use a function to format a value. The MULT= option is ignored when you specify the DATATYPE= option.
Examples	“Example 3: Creating a Picture Format” on page 938 “Example 4: Creating a Picture Format for Large Dollar Amounts” on page 940

NOEDIT

specifies that numbers are message characters rather than digit selectors. That is, the format prints the numbers as they appear in the picture. For example, the following PICTURE statement creates the MILES. format, which formats any variable value greater than 1000 as **>1000 miles**:

```
picture miles 1-1000='0000'
          1000<-high='>1000 miles' (noedit);
```

Restriction The NOEDIT= option is not valid when you use a function to format a value.

NOTSORTED

stores values or ranges in the order in which you define them. If you do not specify NOTSORTED, then values or ranges are stored in sorted order by default, and SAS uses a binary searching algorithm to locate the range that a particular value falls into. If you specify NOTSORTED, then SAS searches each range in the order in which you define them until a match is found.

Use NOTSORTED if one of the following is true:

- You know the likelihood of certain ranges occurring, and you want your format to search those ranges first to save processing time.
- You want to preserve the order that you define ranges when you print a description of the format using the FMTLIB option.

- You want to preserve the order that you define ranges when you use the ORDER=DATA option and the PRELOADFMT option to analyze class variables in PROC MEANS, PROC SUMMARY, or PROC TABULATE.

Do not use NOTSORTED if the distribution of values is uniform or unknown, or if the number of values is relatively small. The binary searching algorithm that SAS uses when NOTSORTED is not specified optimizes the performance of the search under these conditions.

SAS automatically sets the NOTSORTED option when you use the CPORT and CIMPORT procedures to transport informats or formats between operating environments with different standard collating sequences. This automatic setting of NOTSORTED can occur when you transport informats or formats between ASCII and EBCDIC operating environments. If this situation is undesirable, then do the following:

- Use the CNTLOUT= option in the PROC FORMAT statement to create an output control data set.
- Use the CPORT procedure to create a transport file for the control data set.
- Use the CIMPORT procedure in the target operating environment to import the transport file.
- In the target operating environment, use PROC FORMAT with the CNTLIN= option to build the formats and informats from the imported control data set.

PREFIX='prefix'

specifies a character prefix to place in front of the formatted value. The prefix is placed in front of the value's first significant digit. You must use zero-digit selectors or the prefix is not used.

Typical uses for PREFIX= are printing leading currency symbols and minus signs. For example, the PAY. format prints the variable value 25500 as \$25,500.00:

```
picture pay
  low-high='000,009.99' (prefix='$');
```

Default no prefix

Restriction The PREFIX= option is not valid when you use a function to format a value.

Interaction If you use the FILL= and PREFIX= options in the same picture, then the format places the prefix and then the fill characters.

Examples [“Example 3: Creating a Picture Format” on page 938](#)
 [“Example 5: Filling a Picture Format” on page 943](#)

CAUTION **If the picture is not wide enough to contain both the value and the prefix, then the format truncates or omits the prefix, which results in inaccurate data.**

ROUND

rounds the value to the nearest integer before formatting. Without the ROUND option, the format multiplies the variable value by the multiplier, truncates the

decimal portion (if any), and prints the result according to the template that you define. With the ROUND option, the format multiplies the variable value by the multiplier, rounds that result to the nearest integer, and then formats the value according to the template. Note that if the FUZZ= option is also specified, the rounding takes place after SAS has used the fuzz factor to determine which range the value belongs to.

Tip The ROUND option rounds a value of .5 to the next highest integer.

CAUTION **The picture must be wide enough for an additional digit if rounding a number adds a digit to the number.** For example, the picture for the number .996 could be '99' (prefix '.' mult=100). After rounding the number and multiplying it by 100, the resulting number is 100. When the picture is applied, the result is .00, an inaccurate number. In order to ensure accuracy of numbers when you round numbers, make the picture wide enough to accommodate larger numbers.

value-range-set

specifies one or more variable values and a template for printing those values. *value-range-set* has the following form:

value-or-range-1 <, *value-or-range-2*, ...>=*'picture'*

value-or-range

See [“Specifying Values or Ranges” on page 918](#).

picture

specifies a template for formatting values of numeric variables. The picture is a sequence of characters in single quotation marks. The maximum length for a picture is 40 characters. Pictures are specified with three types of characters: digit selectors, message characters, and directives. You can have a maximum of 16 digit selectors in a picture.

digit selectors

are numeric characters (0 through 9) that define positions for numeric values. A picture format with nonzero-digit selectors prints any leading zeros in variable values; picture digit selectors of 0 do not print leading zeros in variable values. If the picture format contains digit selectors, then a digit selector must be the first character in the picture.

Note This section uses 9s as nonzero-digit selectors.

message characters

are nonnumeric characters that are printed as specified in the picture. The following PICTURE statement contains both digit selectors (99) and message characters (*illegal day value*). Because the DAYS. format has nonzero-digit selectors, values are printed with leading zeros. The special range OTHER prints the message characters for any values that do not fall into the specified range (1 through 31).

```
picture days
    01-31='99'
    other='99-illegal day value';
```

Example The values 02 and 67 are printed as

67-illegal day value

directives

are special characters that you can use in the picture to format date, time, or datetime values.

Note: You can use directives only when you specify the [DATATYPE= option on page 889](#) in the PICTURE statement. Ensure that the value of the DATATYPE= option is appropriate for the type of directive that you want to use. If you use an inappropriate value, the data does not format. For example, for the %a directive, use DATATYPE=DATE.

The DFLANG datetime handler and the National Language (NL) datetime handler are the two methods to control character casing in user format handling. The DFLANG datetime handler is the default method under non-UTF8 and non-DBCS sessions. The NL datetime handler is the default method under UTF8 and DBCS sessions.

Note: The DFLANG datetime handler is used if you specify it for UTF8 and DBCS sessions. Otherwise, the NL datetime handler is used.

If you specify the DFLANG datetime handler, then month names are in all uppercase if you specify English (for example, JAN), they are in all lowercase if you specify French (for example, jan), and they are in mixed case if you specify German (for example, Jan).

If you specify the NL datetime handler, then month names are in mixed case if you specify English or German (Jan), and they are in lowercase if you specify French (for example, jan).

The DFLANG datetime handler is used only if a supported language is specified with the DFLANG= system option and all of the specified datetime elements are supported by the DFLANG format handler. Otherwise, the NL datetime handler is used.

The casing of the month names is determined by the DFLANG= system option and the LANGUAGE= argument of the PICTURE statement. Otherwise, the casing is determined by the language elements in the locale data. For example, if the language specified by the LOCALE= option is English, then only the first character is in uppercase because it is defined as mixed-case strings in the locale data. For more information, see [“DFLANG= System Option” in SAS National Language Support \(NLS\): Reference Guide](#).

Note: The DFLANG option supports only 22 languages. Use the DFLANG= option only for compatibility purposes. When you specify the DFLANG= option, support for the datetime directive is limited and some elements such as %l are not supported.

Specify the `LANGUAGE=` argument if you need to use it in the `FORMAT` procedure, or you need month names in all uppercase for a locale such as English.

If you mainly use the `EUR*` datetime formats and you need support for European languages only for compatibility, then you can use the `DFLANG=` option to specify the language.

Use the `LOCALE=` system option to set the locale for double-byte and UTF-8 character sets. The `%b` directive formats month names with only the first letter in uppercase when the locale is set with `LOCALE=`. For more information, see [“LOCALE System Option” in SAS National Language Support \(NLS\): Reference Guide](#).

If you specify a format element that is supported by only the NL datetime handler, then the NL format handler is used regardless of whether the `DFLANG=` option is used. `%b` also produces abbreviated month names in mixed case for English if there is an unsupported directive such as `%l`.

The default value for the `LANGUAGE=` argument in UTF-8 sessions is `LOCALE`. To format month names in all upper-case characters in UTF-8 sessions, specify the locale on the `LANGUAGE=` argument. For more information, see [“LANGUAGE=” on page 890](#).

The permitted directives are as follows:

%a

abbreviated weekday name (for example, Wed).

%A

full weekday name (for example, Wednesday).

%b

abbreviated month name (for example, JAN or Jan).

Tip For the English language, use the directive `%3B` to create an abbreviated month with only an uppercase initial letter (for example, Jan).

%<n>B

the full month name (for example, January) if *n* is not included in the directive. *n* specifies the number of characters that appear for the month name. In comparison, the `%b` directive writes a three-character month abbreviation in uppercase letters for some locales.

Restriction *n* is not supported in DBCS and Unicode SAS sessions.

Example `%3B` would write Oct for the month of October

%C

long month name with blank padding (January through December) (for example, December).

%d

day of the month.

Note To add a leading zero before a single-digit number, insert a 0 before the directive (for example, %0d).

%e

day of the month as a two-character decimal number with leading spaces (" 1"- "31") (for example, " 2").

%F

full weekday name with blank padding.

%G

year as a four-digit decimal number (for example, 2008). If the week that contains January 1 has four or more days in the new year, then it is considered week 1 in the new year. Otherwise, it is the last week of the previous year and the year is considered the previous year.

%H

hour (24-hour clock).

Note To add a leading zero before a single-digit number, insert a 0 before the directive (for example, %0H).

Tip When DATETYPE=DATETIME, SAS uses datetime hours 00:00–23:59. When DATETYPE=DATETIME_UTIL, SAS uses datetime hours 00:00:01–24:00:00 and 24:00:00 is midnight at the end of the day. The hour 00:00:00 is not in the hour range and if used, converts to 24:00:00 of the previous day. When you specify DATETIME, 00:00:00 is midnight of a new day and the value 24:00:00 is midnight of the next day.

Example [“Example 7: Change the 24–Hour Clock to 00:00:01–24:00:00” on page 947](#)

%I

hour (12-hour clock).

Alias %i

Note To add a leading zero before a single-digit number, insert a 0 before the directive (for example, %0I).

%j

day of the year as a decimal number (1–366), with leading zero.

Note To add a leading zero before a single-digit number, insert a 0 before the directive (for example, %0j).

%m

month (1–12).

Note To add a leading zero before a single-digit number, insert a 0 before the directive (for example, %0m).

%M

minute (0–59).

Note To add a leading zero before a single-digit number, insert a 0 before the directive (for example, %OM).

%n

number of days in a duration as a decimal number (maximum of 10 digits) (for example, 25).

Restriction This directive is not valid for DBCS and Unicode SAS sessions.

%o

month (1-12) with blank padding (for example, " 2").

%p

equivalent to either a.m. or p.m.

%q

abbreviated quarter of the year string such as 1, 2, 3, or 4.

Restriction "_C" and "_B" (both case insensitive) are reserved keys for format names. These keys cannot appear at the end of a format name. If format catalogs are migrated from SAS 9 with these suffixes, an error appears. The workaround is to recreate the format entry with a name that does not contain the _C or _B suffix .

%Q

quarter of the year string, such as Quarter1, Quarter2, Quarter3, or Quarter4.

%s

fractional seconds as decimal digits (for example, .39555). The number of digits formatted is the number of digits to the right of the decimal point that is specified when you use the format. SAS rounds fractional seconds to accommodate the number of digits specified for fractional seconds.

Restriction This directive is not valid for DBCS and Unicode SAS sessions.

Interaction The %s and %S directives cannot be used together.

Tips The %s directive displays both whole and fractional seconds.

To add a leading zero before a single-digit number, insert a 0 before the directive (for example, %0s).

%S

seconds (0–59), allowing for possible leap seconds.

Interaction The %s and %S directives cannot be used together.

Tips The %S directive displays whole seconds. It does not display fractional seconds.

To add a leading zero before a single-digit number, insert a 0 before the directive (for example, %0S).

Example 58 and 59.07

%u

weekday as a one-digit decimal number (1–7 (Monday - Sunday)) (for example, Sunday=7).

%U

week number of the year as a decimal number (0–53). Sunday is considered the first day of the week.

Note To add a leading zero before a single-digit number, insert a 0 before the directive (for example, %0U).

%V

week number (01–53) with the first Monday as the start day of the first week. Minimum days of the first week is 4.

Note To add a leading zero before a single-digit number, insert a 0 before the directive (for example, %0SV).

%w

weekday as a one-digit decimal number (0–6 (Sunday through Saturday)) (for example, Sunday=0).

Note To add a leading zero before a single-digit number, insert a 0 before the directive (for example, %0w).

%W

week number (0–53) with the first Monday as the start day of the first week.

Note To add a leading zero before a single-digit number, insert a 0 before the directive (for example, %0W).

%y

year without century (0–99) (for example, 93).

Note To add a leading zero before a single-digit number, insert a 0 before the directive (for example, %0y).

%Y

year with century as a four-digit decimal number (1970–2069) (for example, 1994).

%z

UTC time-zone offset.

%Z

time-zone name.

%%
the % character.

Tip Add code to your program to direct how you want missing values to be displayed.

Interaction If you specify LANGUAGE= and PICTURE= in the format definition, the format supports only English and the European languages. To use a user-defined format in languages other than those that are supported by the LANGUAGE= option, use the PICTURE= statement. Do not specify the LANGUAGE= option. The language of a picture format is determined by the locale setting.

Details

Building a Picture Format: Step by Step

This section shows how to write a picture format that eliminates leading zeros. In the SAMPLE data set, the default printing of the variable Amount has leading zeros on numbers between 1 and -1. The PICTURE statement defines two similar formats that eliminate leading zeros on numbers between 1 and -1. The difference between the two formats is that the NOZEROSR. format specifies the ROUND option to round numbers and the NOZEROS. format does not round numbers.

This program creates, sorts, and prints the sample data set:

```
data sample;
  input Amount;
  datalines;
-2.051
-.05
-.017
0
.093
.54
.556
6.6
14.63
0.996
-0.999
-45.00
;
run;

proc sort data=sample;
  by amount;
run;

proc print data=sample;
  title 'Default Printing of the Variable Amount';
run;
```

Default Printing of the Variable Amount

Obs	Amount
1	-45.000
2	-2.051
3	-0.999
4	-0.050
5	-0.017
6	0.000
7	0.093
8	0.540
9	0.556
10	0.996
11	6.600
12	14.630

Here is the PROC FORMAT step that creates the NOZEROSR. and NOZEROS. formats. Both formats eliminate leading zeros in the formatted values. The NOZEROSR. format specifies the ROUND option to round numbers. The NOZEROS. format does not perform rounding.

```
libname library 'SAS-library';

proc format;
  picture nozerosR (round fuzz=0)
    low - -1      = '000.00' (prefix='-')
    -1 < - < -.99 = '0.99'   (prefix='-' mult=100)
    -0.99 < -< 0  = '99'     (prefix='-' mult=100)
                0  = '9.99'
    0 < -< .99    = '99'     (prefix='.' mult=100)
    0.99 - < 1    = '0.99'   (mult=100)
    1 - high      = '09.99';

  picture nozeros (fuzz=0)
    low - -1      = '000.00' (prefix='-')
    -1 < - < -.99 = '0.99'   (prefix='-' mult=100)
    -0.99 < -< 0  = '99'     (prefix='-' mult=100)
                0  = '9.99'
    0 < -< .99    = '99'     (prefix='.' mult=100)
    0.99 - < 1    = '0.99'   (prefix='.' mult=100)
    1 - high      = '09.99';

run;
```

The following table explains how one value from each range is formatted. For an illustration of each step, see [Table 26.70 on page 906](#).

Table 26.2 Building a Picture Format

Step	Rules for Processing the PICTURE Statement	In This Example
1	Determine into which range the value falls and use that picture.	In the second range, the exclusion operator < appears on both sides of the hyphen and excludes -1 and -.99 from the range. The third range excludes 0 and .99. The fourth range excludes 1. Because exclusion operators are used, the FUZZ=0 option is specified.
2	Take the absolute value of the numeric value.	Because the absolute value is used, you need a separate range and picture for the negative numbers in order to prefix the minus sign.
3	Multiply the number by the MULT= value. If you do not specify the MULT= option, then the PICTURE statement uses the default. The default is 10^n , where n is the number of digit selectors to the right of the decimal in the picture. (Step 6 discusses digit selectors further.) ¹	Specifying a MULT= value is necessary for numbers between 0 and 1 and numbers between 0 and -1 because no decimal appears in the pictures for those ranges. Because MULT= defaults to 1, truncation of the significant digits results without a MULT= value specified. (Truncation is explained in the next step.) For the three ranges that do not have MULT= values specified, the MULT= value defaults to 100 because the corresponding picture has two digit selectors to the right of the decimal. After the MULT= value is applied, all significant digits are moved to the left of the decimal.
4	If the number is within 10^{-8} of a higher integer, round the number up. This operation is performed before the ROUND option is performed. The ROUND option is in effect. The format rounds the number after the decimal to the next highest integer if the number after the decimal is greater than or equal to .5.	Because the example uses MULT= values that ensured that all of the significant digits were moved to the left of the decimal, no significant digits are lost. The zeros are truncated. 205.1 is rounded to 205. 55.6 is rounded up to 56. 99.6 is rounded up to 100. Rounding is not performed on 5 and 660.
4a	When the ROUND option is not performed, the numbers after the decimal are truncated.	205.1 is truncated to 205. 55.6 is truncated to 55. 99.6 is truncated to 99.
5	Turn the number into a character string. If the	205 becomes the character string 00205. 5 becomes the character string 05.

Step	Rules for Processing the PICTURE Statement	In This Example
5a	number is shorter than the picture, then the length of the character string is equal to the number of digit selectors in the picture. Pad the character string with leading zeros. (The results are equivalent to using the <code>Zw.</code> format. <code>Zw.</code> is explained in the section on SAS formats in SAS Formats and Informats: Reference.	56 becomes the character string 56. 100 becomes the character string 100. 660 becomes the character string 0660. When the picture is longer than the numbers, the format adds a leading zero to the value. The format does not add leading zeros to the character string 56 and 100 because the corresponding picture has the same number of selectors.
		205 becomes the character string 00205. 5 becomes the character string 05. 55 becomes the character string 55. 99 becomes the character string 099. 660 becomes the character string 0660. When the picture is longer than the numbers, the format adds a leading zero to the value. The format does not add leading zeros to the character string 55 because the corresponding picture has the same number of selectors.
6	Apply the character string to the picture. The format maps only the rightmost n characters in the character string, where n is the number of digit selectors in the picture. Thus, it is important to make sure that the picture has enough digit selectors to accommodate the characters in the string. After the format takes the right-most n characters, it then maps those characters to the picture from left to right. Choosing a zero or nonzero-digit selector is important if the character string contains leading zeros. If one of the leading zeros in the character string maps to a nonzero-digit selector, then	00205 is mapped to 2.05. 05 is mapped to 05. 56 is mapped to 56. 100 is mapped to 1.00. 0660 is mapped to 6.60. The leading zero is dropped from the character strings 00205 and 0660 because the leading zero maps to a zero-digit selector in the picture.

Step	Rules for Processing the PICTURE Statement	In This Example
	it and all subsequent leading zeros and message characters become part of the formatted value. If all of the leading zeros map to zero-digit selectors, then none of the leading zeros or message characters become part of the formatted value. The format replaces the leading zeros in the character string with blanks.²	
6a		00205 is mapped to 2.05. 05 is mapped to 05. 55 is mapped to 55. 099 is mapped to 99. 0660 is mapped to 6.60. The leading zero is dropped from the character strings 00205, 099, and 0660, because the leading zero maps to a zero-digit selector in the picture. The period (.) message character in the 0.99 picture is dropped because the leading zero maps to a zero-digit selector. Because the period message character is dropped, the format definition for the range 0.99 – < 1 requires a prefix of ' ' in the NOZEROS. format to format a decimal number.
7	Prefix any characters that are specified in the PREFIX= option. You need the PREFIX= option because when a picture contains any digit selectors, the picture must begin with a digit selector. Thus, you cannot begin your picture with a decimal point, minus sign, or any other character that is not a digit selector.	The PREFIX= option reclaims the decimal point and the negative sign, as shown with the formatted values -2.05, -.05 and .56.

Step	Rules for Processing the PICTURE Statement	In This Example
7a		The PREFIX= option reclaims the decimal point and the negative sign, as shown with the formatted values -2.05, -.05, .55, and .99.

- 1 A decimal in a PREFIX= option is not part of the picture.
- 2 You can use the FILL= option to specify a character other than a blank to become part of the formatted value.

Table 26.3 Steps to Format Various Values

Step	Action	-2.051	-.05	.556	.996	6.6
1	Range	low – -1	-0.99 < - < 0	0 < - < .99	0.99 – < 1	1 – high
	Picture	000.00	99	99	0.99	09.99
2	Absolute value	2.051	.05	.556	.996	6.6
3	MULT=	2.051×10 ² =205.1	.05×100=5	.556×100=55.6	.996×100=99.6	6.6×10 ² =660
4	Round	205	5	56	100	660
4a	No Rounding	205	5	55	99	660
5	Character string, rounding	00205	05	56	100	0660
5a	Character string, no rounding	00205	05	55	099	0660
6	Template, rounding	2.05	05	56	1.00	6.60
6a	Template, no rounding	2.05	05	55	99	6.60
7	Prefix, rounding	prefix='-'	prefix='-.'	prefix=''	none	none

Step	Action	-2.051	-.05	.556	.996	6.6
7a	Prefix, no rounding	prefix='-'	prefix='-.'	prefix=''	prefix=''	none
	Formatt ed result, rounding	-2.05	-.05	.56	1.00	6.60
	Formatt ed results, no rounding	-2.05	-.05	.55	.99	6.60

The following PROC PRINT steps associates the NOZEROSR. format and the NOZEROS. format with the AMOUNT variable in SAMPLE. The first output shows the result of rounding.

```
proc print data=sample;
    format amount nozerosr.;
    title 'Formatting the Variable Amount';
    title2 'with the NOZEROSR. Format Using Rounding';
run;

proc print data=sample;
    format amount nozeros.;
    title 'Formatting the Variable Amount';
    title2 'with the NOZEROS. Format, No Rounding';
run;
```

**Formatting the Variable Amount
with the NOZEROSR. Format Using Rounding**

Obs	Amount
1	-45.00
2	-2.05
3	-1.00
4	-.05
5	-.02
6	.00
7	.09
8	.54
9	.56
10	1.00
11	6.60
12	14.63

**Formatting the Variable Amount
with the NOZEROS. Format, No Rounding**

Obs	Amount
1	-45.00
2	-2.05
3	-.99
4	-.05
5	-.01
6	.00
7	.09
8	.54
9	.55
10	.99
11	6.60
12	14.63

CAUTION

The picture must be wide enough for the prefix and the numbers. In this example, if the value -45.00 were formatted with NOZEROS., then the result would be 45.00 because it falls into the first range, low $- -1$, and the picture for that range is not wide enough to accommodate the prefixed minus sign and the number.

CAUTION

The picture must be wide enough for an additional digit if rounding a number adds a digit to the number. For example, the picture for the number .996 could be '99' (prefix '.' mult=100). After rounding the number and multiplying it by 100, the resulting number is 100. When the picture is applied, the result is .00, an inaccurate number. In order to ensure accuracy of numbers when you round numbers, make the picture wide enough to accommodate larger numbers.

Specifying No Picture

This PICTURE statement creates a *picture-name* format that has no picture:

```
picture picture-name;
```

Using this format has the effect of applying the default SAS format to the values.

SELECT Statement

Selects entries for processing by the FMTLIB and CNTLOUT= options.

Restrictions: Only one SELECT statement can appear in a PROC FORMAT step. You cannot use a SELECT statement and an EXCLUDE statement within the same PROC FORMAT step.

Example: [“Example 15: Printing the Description of Informats and Formats” on page 971](#)

Syntax

```
SELECT entry(s);
```

Required Argument

entry(s)

specifies one or more catalog entries for processing. Catalog entry names are the same as the name of the informat or format that they store. Because informats and formats can have the same name, and because character and numeric informats or formats can have the same name, you must use certain prefixes when specifying informats and formats in the SELECT statement. Follow these rules when specifying entries in the SELECT statement:

- Precede names of entries that contain character formats with a dollar sign (\$).

- Precede names of entries that contain character informats with an at sign and a dollar sign (for example, @\$entry-name).
- Precede names of entries that contain numeric informats with an at sign (@).
- Specify names of entries that contain numeric formats without a prefix.

Details

Shortcuts to Specifying Names

You can use the colon (:) and hyphen (-) wildcard characters to select entries. For example, the following SELECT statement selects all formats or informats that begin with the letter **a**.

```
select a;;
```

In addition, the following SELECT statement selects all formats or informats that occur alphabetically between **apple** and **pear**, inclusive:

```
select apple-pear;
```

How the FMTLIB and CNTLOUT= Options Affect Whether a Catalog Is Opened in Read or Update Mode

Using the FMTLIB and CNTLOUT= options in the SELECT statement indicates whether a catalog is opened for Read or Update mode. The following rules apply:

- If you use the SELECT statement and do not specify the FMTLIB option or the CNTLOUT= option, PROC FORMAT assumes that the catalog is opened in Update mode.
- If you use the SELECT statement and specify the FMTLIB option or the CNTLOUT= option, the catalog is opened for Read access.
- If you use the SELECT statement without the FMTLIB option or the CNTLOUT= option, and the SAS program does not have Write access to the catalog, the following error is written to the SAS log:

```
ERROR: User does not have appropriate authorization level
       for file libref.FORMATS.CATALOG.
```

VALUE Statement

Creates a format that specifies character strings to use to print variable values.

See: [SAS Formats and Informats: Reference](#) for documentation about SAS formats.

Examples: [“Example 8: Creating a Format for Character Values” on page 948](#)
[“Example 11: Writing a Format for Dates Using a Standard SAS Format and a Color Background” on page 955](#)
[“Example 17: Writing Ranges for Character Strings” on page 975](#)

Syntax

VALUE <\$>*name* <(format-options)> <value-range-set(s)>;

Summary of Optional Arguments

DEFAULT=*length*

specifies the default length of the format.

FUZZ=*fuzz-factor*

specifies a fuzz factor for matching values to a range.

MAX=*length*

specifies a maximum length for the format.

MIN=*length*

specifies a minimum length for the format.

MULTILABEL

enables the assignment of multiple labels or external values to internal values.

NOTSORTED

stores values or ranges in the order in which you define them.

value-range-set(s)

specifies the assignment of a value or a range of values to a formatted value.

Required Argument

name

names the format that you are creating. If you created a function using the FCMP procedure to use as a format, *name* is the function name without parenthesis.

Restrictions The name of a user-defined format cannot be the same as the name of a format that is supplied by SAS.

Format names cannot end in a number. For more information, see [“User-Defined Formats” in SAS Formats and Informats: Reference](#) and [“SAS Names” in SAS Programmer’s Guide: Essentials](#).

“_C” and “_B”(both case insensitive) are reserved keys for format names. These keys cannot appear at the end of a format name. If format catalogs are migrated from SAS 9 with these suffixes, an error appears. The workaround is to recreate the format entry with a name that does not contain the _C or _B suffix .

Requirement The name must be a valid SAS name. A numeric format name can be up to 32 characters in length. A character format name can be up to 31 characters in length. If you are creating a character format, then use a dollar sign (\$) as the first character.

Interaction	The maximum length of a format name is controlled by the VALIDFMTNAME= system option. See SAS System Options: Reference for details.
Tip	Refer to the format later by using the name followed by a period. However, do not use a period after the format name in the VALUE statement.
See	“Using a Function to Format Values” on page 921

Optional Arguments

DEFAULT=length

specifies the default length of the format. The value for DEFAULT= becomes the length of the format if you do not give a specific length when you associate the format with a variable.

Default	The length of the longest label that is assigned to the right of the equal sign
Range	1–32767
Tip	As a best practice, always specify the DEFAULT= option if you specify a format as a label.

FUZZ=fuzz-factor

specifies a fuzz factor for matching values to a range. If a number does not match or fall in a range exactly but comes within *fuzz-factor*, then the format considers it a match. For example, the following VALUE statement creates the LEVELS. format, which uses a fuzz factor of .2:

```
value levels (fuzz=.2) 1='A'
                      2='B'
                      3='C' ;
```

FUZZ=.2 means that if a variable value falls within .2 of a value on either end of the range, then the format uses the corresponding formatted value to print the variable value. So the LEVELS. format formats the value 2.1 as B.

Default 1E–12 for numeric formats and 0 for character formats.

Tips Specify FUZZ=0 to save storage space when you use the VALUE statement to create numeric formats.

Use a nonzero fuzz factor only with numbers that are very close but not an exact match. Ranges are stored internally in sorted order (unless the NOTSORTED option is used), in order to perform a binary search. When a fuzz-factor is added to the end of one range and subtracted from the beginning of the next range, and the ranges overlap, the results can be unpredictable. A value is placed in the first range that is a match in the binary search. The exclusion operator is insufficient to override this binary search algorithm. As a best practice, when you use the exclusion operator, set FUZZ=0 or the NOTSORTED option

A best practice is to use FUZZ=0 when you use the < exclusion operator with numeric formats.

MAX=length

specifies a maximum length for the format. When you associate the format with a variable, you cannot specify a width greater than the MAX= value.

Default 40

Range 1–32767

MIN=length

specifies a minimum length for the format.

Default 1

Range 1–32767

MULTILABEL

enables the assignment of multiple labels or external values to internal values. The following VALUE statements show the two uses of the MULTILABEL option. The first VALUE statement assigns multiple labels to a single internal value. Multiple labels can also be assigned to a single range of internal values. The second VALUE statement assigns labels to overlapping ranges of internal values. The MULTILABEL option allows the assignment of multiple labels to the overlapped internal values.

```
value one (multilabel)
  1='ONE'
  1='UNO'
  1='UN';

value agefmt (multilabel)
  15-29='below 30 years'
  30-50='between 30 and 50'
  51-high='over 50 years'
  15-19='15 to 19'
  20-25='20 to 25'
  25-39='25 to 39'
  40-55='40 to 55'
  56-high='56 and above';
```

Only multilabel-enabled procedures such as PROC MEANS, PROC SUMMARY, and PROC TABULATE can use multiple labels. All other procedures and the DATA step recognize only the primary label.

The primary label for a given entry is the external value that is assigned to the first internal value or range of internal values that matches or contains the entry when all internal values are ordered sequentially. Here is an example:

- In the first VALUE statement, the primary label for 1 is ONE because ONE is the first external value that is assigned to 1. The secondary labels for 1 are UNO and UN.
- In the second VALUE statement, the primary label for 33 is 25 to 39 because the range 25–39 is sequentially the first range of internal values

that contains 33. The secondary label for 33 is **between 30 and 50** because the range 30–50 occurs in sequence after the range 25–39.

Restriction The maximum number of labels that can be created for a single format is 255.

NOTSORTED

stores values or ranges in the order in which you define them. If you do not specify NOTSORTED, then values or ranges are stored in sorted order by default, and SAS uses a binary searching algorithm to locate the range that a particular value falls into. If you specify NOTSORTED, then SAS searches each range in the order in which you define them until a match is found.

Use NOTSORTED if one of the following is true:

- You know the likelihood of certain ranges occurring, and you want your format to search those ranges first to save processing time.
- You want to preserve the order that you define ranges when you print a description of the format using the FMTLIB option.
- You want to preserve the order that you define ranges when you use the ORDER=DATA option and the PRELOADFMT option to analyze class variables in PROC MEANS, PROC SUMMARY, or PROC TABULATE.

Do not use NOTSORTED if the distribution of values is uniform or unknown, or if the number of values is relatively small. The binary searching algorithm that SAS uses when NOTSORTED is not specified optimizes the performance of the search under these conditions.

SAS automatically sets the NOTSORTED option when you use the CPORT and CIMPORT procedures to transport formats between operating environments with different standard collating sequences. This automatic setting of NOTSORTED can occur when you transport formats between ASCII and EBCDIC operating environments. If this situation is undesirable, then do the following:

- Use the CNTLOUT= option in the PROC FORMAT statement to create an output control data set.
- Use the CPORT procedure to create a transport file for the control data set.
- Use the CIMPORT procedure in the target operating environment to import the transport file.
- In the target operating environment, use PROC FORMAT with the CNTLIN= option to build the formats from the imported control data set.

value-range-set(s)

specifies the assignment of a value or a range of values to a formatted value. The *value-range-set(s)* have the following form:

value-or-range-1 <, *value-or-range-2*, ...>=*formatted-value* | [*existing-format*] | [*functionname*]

The variable values on the left side of the equal sign prints as the character string on the right side of the equal sign. The maximum length of each *value-or-range* to the left of the equal sign is 32,767 characters.

value-or-range

For details about how to specify *value-or-range*, see “Specifying Values or Ranges” on page 918.

formatted-value

specifies a character string that becomes the printed value of the variable value that appears on the left side of the equal sign. Formatted values are always character strings, regardless of whether you are creating a character or numeric format.

Formatted values can be up to 32,767 characters. For hexadecimal literals, you can use up to 32,767 typed characters, or up to 16,382 represented characters at 2 hexadecimal characters per represented character. Some procedures, however, use only the first 8 or 16 characters of a formatted value.

Requirements You must enclose a formatted value in single or double quotation marks. The following example shows a formatted value that is enclosed in double quotation marks:

```
value $ score
    'M'="Male"
    'F'="Female";
```

If a formatted value contains a single quotation mark, then enclose the value in double quotation marks:

```
value sect
    1="Smith's class"
    2="Leung's class";
```

Tip Formatting numeric variables does not preclude the use of those variables in arithmetic operations. SAS uses stored values for arithmetic operations.

existing-format

specifies a format that is supplied by SAS or an existing user-defined format. The format that you are creating uses the existing format to convert the raw data that is a match for *value-or-range* on the left side of the equal sign.

Using an existing format can be thought of as nesting formats. A nested level of one means that if you are creating the format A with the format B as a formatted value, then the procedure has to use only one existing format to create A.

Requirement If you use an existing format, then enclose the format name in square brackets (for example, [date9.]) or with parentheses and vertical bars (for example, (|date9.|)). Do not enclose the name of the existing format in single quotation marks.

Tips Avoid nesting formats more than one level. The resource requirements can increase dramatically with each additional level.

As a best practice, if you specify an existing format in *value-range-set*, always specify a default value by using the [DEFAULT= option on page 912](#).

functionname

specifies a function that is supplied by SAS or an existing user-defined function.

Tips As a best practice, you should define the default length.

If you create a function using the FCMP procedure to use as a format, *name* is the function name without parenthesis.

See [“Using a Function to Format Values”](#)

Examples

Example 1: Create a Format to Print Postal Codes for Selected States

The \$STATE. character format prints the postal code for selected states:

```
value $state 'Delaware'='DE'
            'Florida'='FL'
            'Ohio'='OH';
```

The variable value **Delaware** prints as **DE**, the variable value **Florida** prints as **FL**, and the variable value **Ohio** prints as **OH**. Note that the \$STATE. format begins with a dollar sign.

Note: Range specifications are case sensitive. In the \$STATE. format above, the value **OHIO** would not match any of the specified ranges. If you are not certain what case the data values are in, then one solution is to use the UPCASE function on the data values and specify all uppercase characters for the ranges.

Example 2: Write Numeric Values as Character Values

The numeric format ANSWER. writes the values 1 and 2 as yes and no:

```
value answer 1='yes'
            2='no';
```

Example 3: Specifying No Ranges

This VALUE statement creates a *format-name* format that has no ranges:

```
value format-name;
```

Using this format has the effect of applying the default SAS format to the values.

Example 4: Create a Format Using Existing SAS Formats as Labels

This program creates the MYfmt. format to format dates based on the year.

Note: If the format-as-label or function-as-label is within square brackets, then it is referred to as nested. A format specification in square brackets is followed by at least a period, for example, [year.]. A function specification in square brackets is followed by parentheses, for example, [year()]. SAS uses the nested format to format the values in the range. For example, see low- '31DEC2011'd= [year4.] in the following example. For another example that uses nested formats, see [“Example 5: Create a Format for Missing and Nonmissing Values Using an Existing SAS Format as a Label” on page 917.](#)

```
data test;
  do Date='01jan2006'd to '31dec2013'd;
    do j=1 to rannor(0)*100;
      output;
    end;
  end;
run;
proc format;
  value MYfmt
    /* Format dates prior to 31DEC2011 using only a year. */
    low-'31DEC2011'd=[year4.]

    /* Format 2012 dates using the month and year. */
    '01jan2012'd-'31DEC12'd=[monyy7.]

    /* Format dates 01JAN2013 and beyond using the day, month, and
year. */
    '01JAN2013'd-high=[date9.]

    /* Catch missing values. */
    other='n/a';
run;

proc freq data=test;
  table date /missing;
  format date myfmt.;
run;
```

Example 5: Create a Format for Missing and Nonmissing Values Using an Existing SAS Format as a Label

This program formats missing numeric values with the label N/A, and formats nonmissing values with the existing SAS format 12.1.

```
proc format;
  value myfmt .='N/A' other=[12.1];
run;
```

Usage: FORMAT Procedure

Specifying Values or Ranges

As the syntax of the INVALUE, PICTURE, and VALUE statements indicates, you must specify values as *value-range-sets*. On the left side of the equal sign, you specify the values that you want to convert to other values. On the right side of the equal sign, you specify the values that you want the values on the left side to become. This section discusses the different forms that you can use for *value-or-range*, which represents the values on the left side of the equal sign. For details about how to specify values for the right side of the equal sign, see the “Required Arguments” section for the appropriate statement.

The INVALUE, PICTURE, and VALUE statements accept numeric values on the left side of the equal sign. In character informats, numeric ranges are treated as character strings. INVALUE and VALUE also accept character strings on the left side of the equal sign.

As the syntax shows, you can have multiple occurrences of *value-or-range* in each *value-range-set*, using a comma to separate the occurrences. Each occurrence of *value-or-range* is either one of the following:

value

a single value, such as 12 or 'CA'. For character formats and informats, enclose the character values in single quotation marks.

You can use the keyword OTHER= as a single value. OTHER= matches all values that do not match any other value or range. OTHER= includes missing values for both numeric and character user-defined formats. You cannot nest a user-defined format by using the format as the value of OTHER=, unless the format is a function that formats values.

If you specify a format that is too narrow to represent a value, then SAS tries to fit the longer label into the available space. SAS truncates character values on the right, and it sometimes reverts numeric values to the BESTw.d format. If you do not specify an adequate width for a format, then SAS prints asterisks in the output. In this example, the specified value is two characters, DK. The width of DK is not adequate for the five-character 99999 value, and SAS prints two asterisks in the output.

```
proc format;
    value fmtzip 99999="DK";

data testzip;
    zip=27609;
    output;
```

```

zip=99999;
output;
run;

proc print data=testzip;
  format Zip fmtzip.;
run;

```

The result of `zip=99999;` is that `**` appears in the first observation of the column labeled “Zip” in the output table.

One way to prevent this problem from occurring is to assign a width when you apply the format, as in this example.

```

proc format;
  value fmtzip 99999="DK";

data testzip;
  zip=27609;
  output;
  zip=99999;
  output;
run;

proc print data=testzip;
  format Zip fmtzip5.;
run;

```

The result of `format Zip fmtzip5.;` is that adequate space is specified for 27609 to appear correctly in the first observation of the column labeled “Zip” in the output table.

You can also prevent the problem by specifying the `DEFAULT` or `MIN` options when you create the format, as in this example.

```

proc format;
  value fmtzip (default=5) 99999="DK";

data testzip;
  zip=27609;
  output;
  zip=99999;
  output;
run;

proc print data=testzip;
  format Zip fmtzip5.;
run;

```

The result of including `(default=5)` in the value specification is that adequate space is specified for 27609 to appear correctly in the first observation of the column labeled “Zip” in the output table.

Adding blank spaces at the end of a specified value increases the width of the value. For example, the specified value `DK` is not wide enough for 99999. Adding three blank spaces to the value when you specify `DK` provides five spaces for the 99999 value.

Note: If you add blank spaces to a specified value, be sure to add enough blank spaces to accommodate the longest value that the format might handle.

For more information about how SAS formats widths, see [“Syntax ” in SAS Formats and Informats: Reference](#).

For more information about format values, see [“Using a Function to Format Values” on page 921](#). For examples, see [“Example 8: Creating a Format for Character Values” on page 948](#) and [“Example 20: Creating a Function to Use as a Format” on page 986](#).

range

a list of values (for example, 12–68 or 'A' - 'Z'). For ranges with character strings, be sure to enclose each string in single quotation marks. For example, if you want a range that includes character strings from A to Z, then specify the range as 'A' - 'Z', with single quotation marks around the A and around the Z.

If you specify 'A-Z', then the procedure interprets it as a three-character string with A as the first character, a hyphen (-) as the second character, and a Z as the third character.

In numeric user-defined informats, the procedure interprets an unquoted numeric range on the left side of a *value-range-set* as a numeric range. In a character user-defined informat, the procedure interprets an unquoted numeric range on the left side of a *value-range-set* as a character string. For example, in a character informat, the range 12–86 is interpreted as '12' - '86'.

You can use LOW or HIGH as one value in a range, and you can use the range LOW-HIGH to encompass all values. For example, the following are valid ranges:

```
low- 'ZZ'
35-high
low-high
other
```

In numeric ranges, LOW includes the lowest numeric value, excluding missing values. HIGH includes the largest value in the range. In character ranges, LOW includes missing values. OTHER includes missing values for both numeric and character formats.

You can use the less than (<) symbol to exclude values from ranges. If you are excluding the first value in a range, then put the < exclusion operator after the value. If you are excluding the last value in a range, then put the < exclusion operator before the value. For example, the following range does not include 0:

```
0<-100
```

Likewise, the following range does not include 100:

```
0-<100
```

TIP When you use the < exclusion operator to place values in ranges, use the option FUZZ=0 in the VALUE statement for numeric formats. This is not necessary for character formats because FUZZ=0 is the default.

If a value at the high end of one range also appears at the low end of another range, and you do not use the < exclusion operator, then PROC FORMAT assigns the value to the first range. For example, in the following ranges, the value **AJ** is part of the first range:

```
'AA' - 'AJ' = 1  'AJ' - 'AZ' = 2
```

In this example, to include the value **AJ** in the second range, use the < exclusion operator on the first range:

```
'AA' - < 'AJ' = 1  'AJ' - 'AZ' = 2
```

If you overlap values in ranges, then PROC FORMAT returns an error message unless, for the VALUE statement, the MULTILABEL option is specified. For example, the following ranges will cause an error: 'AA' - 'AK' = 1 'AJ' - 'AZ' = 2.

Each *value-or-range* can be up to 32,767 characters. If *value-or-range* has more than 32,767 characters, then the procedure truncates the value after it processes the first 32,767 characters.

Note: You do not have to account for every value on the left side of the equal sign. Those values are converted using the default informat or format. For example, the following VALUE statement creates the TEMP. format, which prints all occurrences of 98.6 as **NORMAL**:

```
value temp 98.6='NORMAL';
```

If the value were 96.9, then the printed result would be **96.9**.

Using a Function to Format Values

SAS provides a way to format a value by first performing a function on a value. By using a function to format values, you can create customized formats. For example, SAS provides four formats to format dates by quarters, YYQw., YYQxw., YYQRw., and YYQRxw. None of these formats satisfy your requirement to use Q1, Q2, Q3, or Q4. You can create a function using the FCMP procedure that creates the Q1, Q2, Q3, and Q4 values based on a date and a SAS format, and then use that function in the FORMAT procedure to create a format.

Here are the steps to create and use a function to format values:

- 1 Use the FCMP procedure to create the function.
- 2 Use the OPTIONS statement to make the function available to SAS by specifying the location of the function in the CMPLIB= system option.

- 3 Use the FORMAT procedure to create a new format.
- 4 Use the new format in your SAS program.

Here is an example:

```

/* Create a function that creates the value Qx from a formatted value.
*/

proc fcmp outlib=work.functions.smd;

    function qfmt(date) $;
        length qnum $4;
        qnum=put(date,yyq4.);
        if substr(qnum,3,1)='Q'
            then return(substr(qnum,3,2));
        else return(qnum);
    endsub;
run;

/* Make the function available to SAS. */

options cmplib=(work.functions);

/* Create a format using the function created by the FCMP procedure. */

proc format;
    value qfmt
        other=[qfmt()]; run;

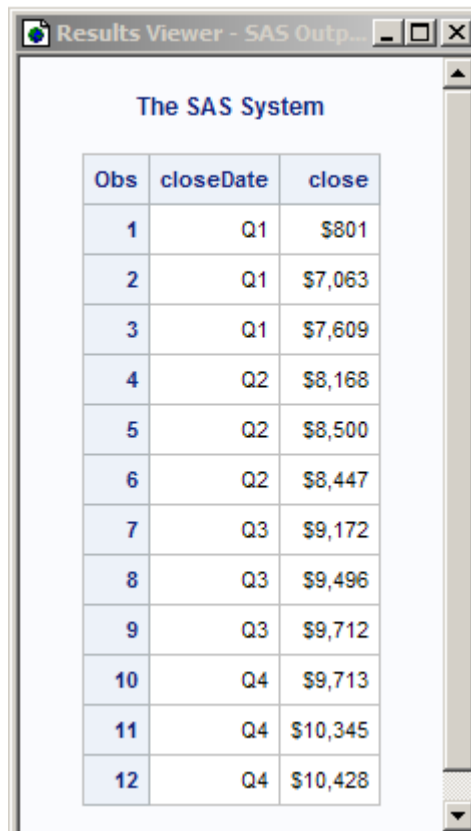
/* Use the format in a SAS program. */

data djia2013;
    input closeDate date7. close;
    datalines;
01jan13 800.86
02feb13 7062.93
02mar13 7608.92
01apr13 8168.12
01may13 8500.33
01jun13 8447.00
01jul13 9171.61
03aug13 9496.28
01sep13 9712.28
01oct13 9712.73
02nov13 10344.84
02dec13 10428.05
run;

proc print data=djia2013;
    format closedate qfmt. close dollar9.;
run;

```

Here is the output:

Output 26.1 Dow Jones Closings by Quarter


The screenshot shows a window titled "Results Viewer - SAS Outp...". Inside, the text "The SAS System" is displayed above a table. The table has three columns: "Obs", "closeDate", and "close". It contains 12 rows of data, grouped by quarter (Q1, Q2, Q3, Q4).

Obs	closeDate	close
1	Q1	\$801
2	Q1	\$7,063
3	Q1	\$7,609
4	Q2	\$8,168
5	Q2	\$8,500
6	Q2	\$8,447
7	Q3	\$9,172
8	Q3	\$9,496
9	Q3	\$9,712
10	Q4	\$9,713
11	Q4	\$10,345
12	Q4	\$10,428

You can use the function format as the value to OTHER=.

Results: FORMAT Procedure

Output Control Data Set

The output control data set contains information that describes informats or formats. Output control data sets have a number of uses. For example, an output control data set can be edited with a DATA step to programmatically change value ranges or can be subset with a DATA step to create new formats and informats. In addition, you can move formats and informats from one operating environment to another by creating an output control data set, using the CPORT procedure to create a transfer file of the data set. Then, use the CIMPORT and FORMAT procedures in the target operating environment to create the formats and informats there.

You create an output control data set with the `CNTLOUT=` option in the `PROC FORMAT` statement. You use output control data sets, or a set of observations from an output control data set, as an input control data set in a subsequent `PROC FORMAT` step using the `CNTLIN=` option.

Output control data sets contain an observation for every value or range in each of the informats or formats in the `LIBRARY=` catalog. The data set consists of variables that give either global information about each format and informat created in the `PROC FORMAT` step or specific information about each range and value.

The variables in the output control data set are as follows:

DATATYPE

enables the use of directives in a picture as a template to format date, time, or datetime values.

DECP

specifies the separator character for the fractional part of a number.

DEFAULT

specifies a numeric variable that indicates the default length for format or informat.

DIG3SEP

specifies the three-digit separator character for a number.

END

specifies a character variable that gives the range's ending value.

EEXCL

specifies a character variable that indicates whether the range's ending value is excluded. Valid values are as follows:

Y

specifies that the range's ending value is excluded.

N

specifies that the range's ending value is not excluded.

FILL

for picture formats, specifies a numeric variable whose value is the value of the `FILL=` option.

FMTNAME

specifies a character variable whose value is the format or informat name.

FUZZ

specifies a numeric variable whose value is the value of the `FUZZ=` option.

HLO

specifies a character variable that contains range information about the format or informat. The following valid values can appear in any combination:

F

specifies a standard SAS format or informat that is used with a value.

H

specifies that a range's ending value is HIGH.

- I**
specifies a numeric informat range.
- J**
specifies justification for an informat.
- L**
specifies that a range's starting value is LOW.
- M**
specifies that the MULTILABEL option is in effect.
- N**
specifies that the format or informat has no ranges, including no OTHER= range.
- O**
specifies that the range is OTHER.
- R**
specifies that the ROUND option is in effect.
- S**
specifies that the NOTSORTED option is in effect.
- U**
specifies that the UPCASE option for an informat be used.
- LABEL**
specifies a character variable whose value is associated with a format or an informat.
- LANGUAGE**
specifies the language that is used for weekdays and months that you can substitute in a date, time, or datetime picture. If you specify a language that is not supported or is invalid, English is used.
- LENGTH**
specifies a numeric variable whose value is the value of the LENGTH= option.
- MAX**
specifies a numeric variable whose value is the value of the MAX= option.
- MIN**
specifies a numeric variable whose value is the value of the MIN= option.
- MULT**
specifies a numeric variable whose value is the value of the MULT= option.
- NOEDIT**
for picture formats, specifies a numeric variable whose value indicates whether the NOEDIT option is in effect. Valid values are as follows:
 - 1**
specifies that the NOEDIT option is in effect.
 - 0**
specifies that the NOEDIT option is not in effect.

PREFIX

for picture formats, specifies a character variable whose value is the value of the PREFIX= option.

SEXCL

specifies a character variable that indicates whether the range's starting value is excluded. Valid values are as follows:

Y

specifies that the range's starting value is excluded.

N

specifies that the range's starting value is not excluded.

START

specifies a character variable that gives the range's starting value.

TYPE

specifies a character variable that indicates the type of format. Possible values are as follows:

C

specifies a character format.

I

specifies a numeric informat.

J

specifies a character informat.

N

specifies a numeric format (excluding pictures).

P

specifies a picture format.

This table specifies TYPE values for creating formats and informats using PROC FORMAT and the CNTLIN option. For example, in Scenario 1 if START=Numeric and LABEL=Numeric, then TYPE=I and you can use the INPUT function to create a numeric column. See [“Example 13: Creating a Format from a CNTLIN= Data Set”](#) for an example of creating a format using PROC FORMAT and the CNTLIN option.

Table 26.4 *TYPE Values*

Scenarios	Format/Informat	START	LABEL	TYPE
1	Format	Numeric	Character	N
2	Format	Character	Character	C
3	Informat	Character	Numeric	I
4	Informat	Character	Character	J

The following output shows an output control data set that contains information about all the informats and formats created in the FORMAT procedure examples.

Output 26.2 Output Control Data Set for PROC FORMAT Examples

Results Viewer - SAS Output

An Output Control Data Set

Obs	FMTNAME	START	END	LABEL	MIN	MAX	DEFAULT	LENGTH	FUZZ	PREFIX	MULT	FILL	NOEDIT	TYPE	SEXCL	EEXCL	HLO	DECSEP	DIG3SEP	DATATYPE	LANGUAGE
1	BENEFIT	LOW	14609	WORDDATE20.	1	40	20	20	1E-12		0.00		0	N	N	N	LF				
2	BENEFIT	14610	HIGH	** Not Eligible **	1	40	20	20	1E-12		0.00		0	N	N	N	H				
3	SALARY	LOW	HIGH	00,000,000.00	1	40	13	13	1E-12	\$	100.00	*	0	P	N	N	LH	.	.		
4	USCURRENCY	LOW	HIGH	000,000	1	40	7	7	1E-12	\$	1.61		0	P	N	N	LH	.	.		
5	CITY	BR1	BR1	Birmingham UK	1	40	14	14	0		0.00		0	C	N	N					
6	CITY	BR2	BR2	Plymouth UK	1	40	14	14	0		0.00		0	C	N	N					
7	CITY	BR3	BR3	York UK	1	40	14	14	0		0.00		0	C	N	N					
8	CITY	US1	US1	Denver USA	1	40	14	14	0		0.00		0	C	N	N					
9	CITY	US2	US2	Miami USA	1	40	14	14	0		0.00		0	C	N	N					
10	CITY	**OTHER**	**OTHER**	INCORRECT CODE	1	40	14	14	0		0.00		0	C	N	N	O				
11	EVALUATION	C	C	1	1	40	1	1	0		0.00		0	I	N	N					
12	EVALUATION	E	E	2	1	40	1	1	0		0.00		0	I	N	N					
13	EVALUATION	N	N	0	1	40	1	1	0		0.00		0	I	N	N					
14	EVALUATION	O	O	4	1	40	1	1	0		0.00		0	I	N	N					
15	EVALUATION	S	S	3	1	40	1	1	0		0.00		0	I	N	N					

You can use the SELECT or EXCLUDE statement to control which formats and informats are represented in the output control data set. For details, see [“SELECT Statement” on page 909](#) and [“EXCLUDE Statement” on page 879](#).

Input Control Data Set

You specify an input control data set with the CNTLIN= option in the PROC FORMAT statement. The FORMAT procedure uses the data in the input control data set to construct informats and formats. Thus, you can create informats and formats without writing INVALUE, PICTURE, or VALUE statements.

The input control data set must have these characteristics:

- For both numeric and character formats, the data set must contain the variables FMTNAME, START, and LABEL, which are described in [“Output Control Data Set” on page 923](#). The remaining variables are not always required.
- If you are creating a character format or informat, then you must either begin the format or informat name with a dollar sign (\$) or specify a TYPE variable with the value **c**.

If multiple formats are defined in the CNTLIN table, and they are of different types (Character AND Numeric), then the TYPE column must be added.
- If you are creating a PICTURE statement format, then you must specify a TYPE variable with the value **p**.
- If you are creating a format with ranges of input values, then you must specify the END variable. If range values are to be noninclusive, then the variables SEXCL and EEXCL must each have a value of **y**. Inclusion is the default.
- If you are creating a format with the CNTLIN= option, the value of LOW or OTHER for the START variable and the value of HIGH for the END variable are interpreted as their corresponding keywords of LOW, OTHER, or HIGH. This interpretation occurs whether you specify the values in uppercase or lowercase.

If you want the explicit values of LOW, OTHER, or HIGH to be used, then specify the HLO variable with the values as described below.

If you specify `START= 'LOW'`, and the HLO variable does not contain 'L', then the literal value of LOW is used. If you specify `START= 'OTHER'`, and the HLO variable does not contain 'O', then the literal value of OTHER is used. If you specify `END= 'HIGH'`, and the HLO variable does not contain 'H', then the literal value of HIGH is used.

You can create more than one format from an input control data set if the observations for each format are grouped together.

You can use a VALUE, INVALUE, or PICTURE statement in the same PROC FORMAT step with the CNTLIN= option. If the VALUE, INVALUE, or PICTURE statement is creating the same informat or format that the CNTLIN= option is creating, then the VALUE, INVALUE, or PICTURE statement creates the informat or format and the CNTLIN= data set is not used. However, you can create an informat or format with VALUE, INVALUE, or PICTURE and create a different informat or format with CNTLIN= in the same PROC FORMAT step.

For an example featuring an input control data set, see [“Example 13: Creating a Format from a CNTLIN= Data Set” on page 961](#).

Procedure Output

The FORMAT procedure prints output only when you specify the FMTLIB option or the PAGE option in the PROC FORMAT statement. The printed output is a table for each format or informat entry in the catalog that is specified in the LIBRARY= option. The output also contains global information and the specifics of each value or range that is defined for the format or informat. You can use the SELECT or EXCLUDE statement to control which formats and informats are represented in the FMTLIB output. For details, see [“SELECT Statement” on page 909](#) and [“EXCLUDE Statement” on page 879](#). For an example, see [“Example 15: Printing the Description of Informats and Formats” on page 971](#).

The FMTLIB output shown in the following output contains a description of the \$CITY. format, which is created in [“Example 8: Creating a Format for Character Values” on page 948](#), and the EVALUATION. informat, which is created in [“Example 12: Converting Raw Character Data to Numeric Values” on page 958](#).

Output 26.3 Output from PROC FORMAT with the FMTLIB Option

FORMAT NAME: \$CITY LENGTH: 14 NUMBER OF VALUES: 6 MIN LENGTH: 1 MAX LENGTH: 40 DEFAULT LENGTH: 14 FUZZ: 0		
START	END	LABEL (VER. V7 V8 12OCT2012:11:33:26)
BR1	BR1	Birmingham UK
BR2	BR2	Plymouth UK
BR3	BR3	York UK
US1	US1	Denver USA
US2	US2	Miami USA
OTHER	**OTHER**	INCORRECT CODE

INFORMAT NAME: @EVALUATION LENGTH: 1 MIN LENGTH: 1 MAX LENGTH: 40 DEFAULT LENGTH: 1 FUZZ: 0		
START	END	INVALUE(VER. 9.4 12OCT2012:11:34:18)
C	C	1
E	E	2

INFORMAT NAME: @EVALUATION LENGTH: 1 MIN LENGTH: 1 MAX LENGTH: 40 DEFAULT LENGTH: 1 FUZZ: 0		
START	END	INVALUE (CONT'D)
N	N	0
O	O	4
S	S	3

The fields are described below in the order in which they appear in the output, from left to right:

INFORMAT NAME or FORMAT NAME

the name of the informat or format. Informat names begin with an at-sign (@).

LENGTH

the length of the informat or format. PROC FORMAT determines the length in the following ways:

- For character informats, the value for LENGTH is the length of the longest raw data value on the left side of the equal sign.
- For numeric informats, the following is true:
 - LENGTH is 12 if all values on the left side of the equal sign are numeric.

- LENGTH is the same as the longest raw data value on the left side of the equal sign.
- For formats, the value for LENGTH is the length of the longest value on the right side of the equal sign.

In the output for \$CITY., the LENGTH is 14 because the longest picture is 14 characters.

In the output for @EVALUATION., the length is 1 because 1 is the longest raw data value on the left side of the equal sign.

NUMBER OF VALUES

the number of values or ranges associated with the informat or format.
NOZEROS. has 4 ranges, and EVAL. has 5.

MIN LENGTH

the minimum length of the informat or format. The value for MIN LENGTH is 1 unless you specify a different minimum length with the MIN= option.

MAX LENGTH

the maximum length of the informat or format. The value for MAX LENGTH is 40 unless you specify a different maximum length with the MAX= option.

DEFAULT LENGTH

the length of the longest value in the INVALUE or LABEL field, or the value of the DEFAULT= option.

FUZZ

the fuzz factor. For informats, FUZZ always is 0. For formats, the value for this field is STD if you do not use the FUZZ= option. STD signifies the default fuzz value.

START

the beginning value of a range. FMTLIB prints only the first 16 characters of a value in the START and END columns.

END

the ending value of a range. The exclusion sign (<) appears after the values in START and END, if the value is excluded from the range.

INVALUE

appears only for informats and contains the values that have informats. The SAS version specifies the version in which the informat is compatible. The date indicates the date in which the informat was created.

Note: If SAS displays version numbers V7 | V8, then the informat is compatible with those versions. If it is not compatible with earlier releases, the release that created the informat is shown. Version V9 supports long informat names (more than eight characters), and V7 | V8 do not.

LABEL

LABEL appears only for formats and contains either the formatted value or picture. The SAS version specifies the version in which the format is compatible. The date indicates the date in which the format was created.

Note: If SAS displays version numbers V7 | V8, then the format is compatible with those versions. If it is not compatible with earlier releases, the release that created the format is shown. Version V9 supports long format names (more than eight characters), and V7 | V8 do not.

For picture formats, such as NOZEROS., the LABEL section contains the PREFIX=, FILL=, and MULT= values. To note these values, FMTLIB prints the letters **P**, **F**, and **M** to represent each option, followed by the value. For example, in the LABEL section, **P-** . indicates that the prefix value is a hyphen followed by a period.

FMTLIB prints only 40 characters in the LABEL column.

Examples: FORMAT Procedure

Example 1: Create a Format Library in a CAS Session

Features: PROC FORMAT statement option
CASFMTLIB
CAS statement

Details

This example uses the CASFMTLIB option to create a format library in a CAS session. It associates the format library with a table in the WORK directory and assigns a CAS engine libref.

Program

```
cas casauto sessopts=(caslib="casuser");  
caslib _all_ assign;  
  
libname proclib cas;  
proc format casfmtlib='myformats';  
    value hospx
```

```

        1='New_York'
        2='Massachusetts_General'
        3='Los_Angeles'
        4='Mary_Fletcher';
run;

data clinicalTrial;
    input hospital treatment $ @@;
    severity=rannor(1323)*5 + 10;
    format hospital hospx.;
    cards;
3 B 3 B 3 C 3 C
1 A 1 A 1 A 1 B
1 B 1 B 1 C 1 C
1 C 1 D 1 D 1 D
2 A 2 A 2 A 2 B
2 B 2 B 2 C 2 C
2 C 2 D 2 D 2 D
3 A 3 A 3 A 3 B
3 C 3 D 3 D 3 D
4 A 4 A 4 A 4 B
4 B 4 B 4 C 4 C
4 C 4 D 4 D 4 D
;

data proclib.clinicalTrial;
    set work.clinicalTrial;
run;

proc regselect data=proclib.clinicalTrial;    class treatment hospital;
    model severity=treatment hospital;
run;

```

Program Description

Create a format library in a CAS session. Assign a library with the LIBNAME statement. PROC FORMAT creates a format named *hospx*. The CASFMTLIB option specifies the name of the format library *myformats* in the CAS session.

```

cas casauto sessopts=(caslib="casuser");
caslib _all_ assign;

libname proclib cas;
proc format casfmtlib='myformats';
    value hospx
        1='New_York'
        2='Massachusetts_General'
        3='Los_Angeles'
        4='Mary_Fletcher';
run;

```

Associate the HOSPX format with a column or variable.

```

data clinicalTrial;
    input hospital treatment $ @@;
    severity=rannor(1323)*5 + 10;

```

```

format hospital hospx.;
cards;
3 B 3 B 3 C 3 C
1 A 1 A 1 A 1 B
1 B 1 B 1 C 1 C
1 C 1 D 1 D 1 D
2 A 2 A 2 A 2 B
2 B 2 B 2 C 2 C
2 C 2 D 2 D 2 D
3 A 3 A 3 A 3 B
3 C 3 D 3 D 3 D
4 A 4 A 4 A 4 B
4 B 4 B 4 C 4 C
4 C 4 D 4 D 4 D
;

```

Send actions to the CAS session. The LIBNAME statement assigns a CAS engine libref that is used to identify the table in the REGSELECT procedure step.

```

data proclib.clinicalTrial;
    set work.clinicalTrial;
run;

proc regselect data=proclib.clinicalTrial;    class treatment hospital;
    model severity=treatment hospital;
run;

```

LOG

Example Code 26.1 Create a Format Library in a CAS Session, Part 1

```

1      OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;
56
57      cas casauto sessopts=(caslib="casuser");
NOTE: 'CASUSER(userid)' is now the active caslib.
NOTE: The CAS statement request to update one or more session options      for
session CASAUTO completed.
58      caslib _all_ assign;
NOTE: A SAS Library associated with a caslib can only reference library member
names that conform to SAS Library naming conventions.
NOTE: CASLIB CASUSER(userid) for session CASAUTO will be mapped to SAS Library
CASUSER.
NOTE: CASLIB Formats for session CASAUTO will be mapped to SAS Library FORMATS.
NOTE: CASLIB Models for session CASAUTO will be mapped to SAS Library MODELS.
NOTE: CASLIB Public for session CASAUTO will be mapped to SAS Library PUBLIC.
59
60      libname proclib cas;
NOTE: Libref PROCLIB was successfully assigned as follows:
Engine:          CAS
Physical Name: c0d2aee4-4628-5b42-b740-513df8d08070
61      proc format casfmtlib='myformats';
NOTE: Both CAS based formats and catalog-based formats will be written.
The CAS based formats will be written to the session
CASAUTO.
62      value hospx
63          1='New_York'
64          2='Massachusetts_General'
65          3='Los_Angeles'
66          4='Mary_Fletcher';
NOTE: Format library MYFORMATS added. Format search update using parameter
APPEND completed.
NOTE: Format HOSPX has been output.
67      run;

NOTE: PROCEDURE FORMAT used (Total process time):
real time          0.04 seconds
cpu time           0.00 seconds

68
69      data clinicalTrial;
70          input hospital treatment $ @@;
71          severity=rannor(1323)*5 + 10;
72          format hospital hospx.;
73          datalines;

NOTE: SAS went to a new line when INPUT statement reached past the end
of a line.
NOTE: The data set WORK.CLINICALTRIAL has 48 observations and 3 variables.
NOTE: DATA statement used (Total process time):
real time          0.00 seconds
cpu time           0.01 seconds

```

Example Code 26.2 Create a Format Library in a CAS Session, Part 2

```
88      data proclib.clinicalTrial;
89          set work.clinicalTrial;
90      run;

NOTE: There were 48 observations read from the data set WORK.CLINICALTRIAL.
NOTE: The data set PROCLIB.CLINICALTRIAL has 48 observations and 3 variables.
NOTE: DATA statement used (Total process time):
      real time           0.08 seconds
      cpu time            0.01 seconds

91
92      proc regselect data=proclib.clinicalTrial;
93          class treatment hospital;
94          model severity=treatment hospital;
95      run;

NOTE: The Cloud Analytic Services server processed the request in
      0.065319 seconds.
NOTE: PROCEDURE REGSELECT used (Total process time):
      real time           0.25 seconds
      cpu time            0.10 seconds

96
97
98      OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;
```

Creating a Format Library in a CAS Session

The output is divided into sections only for documentation appearances.

The REGSELECT Procedure

Number of Observations Read	48
Number of Observations Used	48

Class Level Information		
Class	Levels	Values
treatment	4	A B C D
hospital	4	Los_Angeles Mary_Fletcher Massachusetts_General New_York

Dimensions	
Number of Effects	3
Number of Parameters	9

Analysis of Variance					
Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	6	251.03651	41.83942	2.02	0.0845
Error	41	848.39550	20.69257		
Corrected Total	47	1099.43201			

Root MSE	4.54891
R-Square	0.22833
Adj R-Sq	0.11541
AIC	201.86300
AICC	205.55531
SBC	164.96141
ASE	17.67491

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Pr > t
Intercept	1	13.240102	1.737143	7.62	<.0001
treatment A	1	1.825309	1.857084	0.98	0.3314
treatment B	1	-1.711518	1.857084	-0.92	0.3621
treatment C	1	-2.357254	1.857084	-1.27	0.2115
treatment D	0	0	.	.	.
hospital Los_Angeles	1	-2.844992	1.857084	-1.53	0.1332
hospital Mary_Fletcher	1	-2.539727	1.857084	-1.37	0.1789
hospital Massachusetts_General	1	-4.498280	1.857084	-2.42	0.0199
hospital New_York	0	0	.	.	.

Task Timing		
Task	Seconds	Percent
Setup and Parsing	0.00	6.29%
Levelization	0.01	59.60%
Model Initialization	0.00	0.29%
SSCP Computation	0.00	16.01%
Model Fitting	0.00	1.69%
Cleanup	0.00	16.00%
Total	0.02	100.00%

Example 2: Create the Example Data Set

Details

Several examples in this section use the PROCLIB.STAFF data set. In addition, many of the informats and formats that are created in these examples are stored in Library.Formats. The output data set shown in [“Output Control Data Set” on page 923](#) contains a description of these informats and the formats.

The variables are about a small subset of employees who work for a corporation that has sites in the U.S. and Britain. The data contain the name, identification number, salary (in British pounds), location, and date of hire for each employee.

Program

```
libname proclib 'SAS-library';

data proclib.staff;
  infile datalines dlm='#';
  input Name & $16. IdNumber $ Salary
        Site $ HireDate date8.;
  format hiredate date8.;
  datalines;
Capalleti, Jimmy# 2355# 21163# BR1# 30JAN13
Chen, Len#       5889# 20976# BR1# 18JUN06
Davis, Brad#     3878# 19571# BR2# 20MAR04
Leung, Brenda#   4409# 34321# BR2# 18SEP94
Martinez, Maria# 3985# 49056# US2# 10JAN93
Orfali, Philip#  0740# 50092# US2# 16FEB03
Patel, Mary#     2398# 35182# BR3# 02FEB90
Smith, Robert#   5162# 40100# BR5# 15APR06
Sorrell, Joseph# 4421# 38760# US1# 19JUN11
Zook, Carla#     7385# 22988# BR3# 18DEC10
;
```

Program Description

```
libname proclib 'SAS-library';
```

Create the data set PROCLIB.STAFF. The INPUT statement assigns the names Name, IdNumber, Salary, Site, and HireDate to the variables that appear after the DATALINES statement. The FORMAT statement assigns the standard SAS format DATE7. to the variable HireDate.

```

data proclib.staff;
  infile datalines dlm='#';
  input Name & $16. IdNumber $ Salary
        Site $ HireDate date8.;
  format hiredate date8.;
  datalines;
Capalleti, Jimmy# 2355# 21163# BR1# 30JAN13
Chen, Len#        5889# 20976# BR1# 18JUN06
Davis, Brad#      3878# 19571# BR2# 20MAR04
Leung, Brenda#    4409# 34321# BR2# 18SEP94
Martinez, Maria#  3985# 49056# US2# 10JAN93
Orfali, Philip#   0740# 50092# US2# 16FEB03
Patel, Mary#      2398# 35182# BR3# 02FEB90
Smith, Robert#    5162# 40100# BR5# 15APR06
Sorrell, Joseph#  4421# 38760# US1# 19JUN11
Zook, Carla#      7385# 22988# BR3# 18DEC10
;

```

Example 3: Creating a Picture Format

Features:	PROC FORMAT statement options LIBRARY= PICTURE statement options MULT= PREFIX= LIBRARY libref LOW and HIGH keywords
Data set:	PROCIB.STAFF from Example 1

Details

This example uses a PICTURE statement to create a format that prints the values for the variable Salary in the data set PROCLIB.STAFF in U.S. dollars.

Program

```

libname proclib 'SAS-library-1';
libname library 'SAS-library-2';

options nodate pageno=1 linesize=80 pagesize=40;

proc format library=library;

  picture uscurrency low-high='000,000' (mult=1.61 prefix='$');
run;

```



```
proc print data=proclib.staff noobs label;

    label salary='Salary in U.S. Dollars';
    format salary uscurrency.;

    title 'PROCLIB.STAFF with a Format for the Variable Salary';
run;
```

Program Description

Assign two SAS library references (PROCLIB and LIBRARY). Assigning a library reference LIBRARY is useful in this case because if you use PROC FORMAT, then SAS automatically searches for informats and formats in any library that is referenced with the LIBRARY libref.

```
libname proclib 'SAS-library-1';
libname library 'SAS-library-2';
```

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=40;
```

Specify that user-defined formats will be stored in the catalog Library.Formats.

The LIBRARY= option specifies a SAS catalog that will contain the formats or informats that you create with PROC FORMAT. When you create the library named LIBRARY, SAS automatically creates a catalog named FORMATS inside LIBRARY.

```
proc format library=library;
```

Define the USCURRENCY. picture format. The PICTURE statement creates a template for printing numbers. LOW-HIGH ensures that all values are included in the range. The MULT= statement option specifies that each value is multiplied by 1.61. The PREFIX= statement adds a US dollar sign to any number that you format. The picture contains six digit selectors, five for the salary and one for the dollar sign prefix.

```
    picture uscurrency low-high='000,000' (mult=1.61 prefix='$');
run;
```

Print the PROCLIB.STAFF data set. The NOOBS option suppresses the printing of observation numbers. The LABEL option uses variable labels instead of variable names for column headings.

```
proc print data=proclib.staff noobs label;
```

Specify a label and format for the Salary variable. The LABEL statement substitutes the specific label for the variable in the report. In this case, “Salary in US Dollars” is substituted for the variable Salary for this print job only. The FORMAT statement associates the USCURRENCY. format with the variable name Salary for the duration of this procedure step.

```
    label salary='Salary in U.S. Dollars';
    format salary uscurrency.;
```

Specify the title.

```

title 'PROCLIB.STAFF with a Format for the Variable Salary';
run;

```

Output

Output 26.4 PROCLIB.STAFF with a Format for the Variable Salary

PROCLIB.STAFF with a Format for the Variable Salary				
Name	IdNumber	Salary in U.S. Dollars	Site	HireDate
Capalleti, Jimmy	2355	\$34,072	BR1	30JAN09
Chen, Len	5889	\$33,771	BR1	18JUN06
Davis, Brad	3878	\$31,509	BR2	20MAR04
Leung, Brenda	4409	\$55,256	BR2	18SEP94
Martinez, Maria	3985	\$78,980	US2	10JAN93
Orfali, Philip	0740	\$80,648	US2	16FEB03
Patel, Mary	2398	\$56,643	BR3	02FEB90
Smith, Robert	5162	\$64,561	BR5	15APR06
Sorrell, Joseph	4421	\$62,403	US1	19JUN11
Zook, Carla	7385	\$37,010	BR3	18DEC10

Example 4: Creating a Picture Format for Large Dollar Amounts

Features: PICTURE statement option
MULT

Format: BIGMONEY.

Details

This example uses the MULT option of the PICTURE statement to format dollars that displays M, B, or T to indicate millions, billions, and trillions of dollars,

respectively. The example uses exponential notation as well as decimal notation in the format definition.

TIP This example uses dollar values without cents and rounding is not necessary. If your dollar values include cents, you can use the ROUND option in the PICTURE statement to round values to the nearest dollar value. For more information, see “ROUND” on page 894.

Program

```
proc format;
  picture bigmoney (fuzz=0)
    1E06-<1000000000='0000 M' (prefix='$' mult=.000001)
    1E09-<1000000000000='0000 B' (prefix='$' mult=1E-09)
    1E12-<1000000000000000='0000 T' (prefix='$' mult=1E-012);
run;

data mult;
  do i=5 to 12;
    x=16**i;
    put x=comma20. x= bigmoney.;
  end;
run;
```

Program Description

Create the BIGMONEY format. The BIGMONEY. format defines three value-range sets to format millions, billions, and trillions of dollars. 1E06 is one million, 1E09 is one billion, and 1E12 is one trillion. The < exclusion operator indicates not to include the number that follows in the range. A best practice is to use the FUZZ=0 option when you use the exclusion operator to ensure accurate numbers. For a million dollars, the range is 1,000,000 to 999,999,999. The label that is specified on the right side of the equal sign uses 4 zeros as digit selectors. The zero-digit selector specifies not to print leading zeros. The first digit selector is necessary to print the \$ prefix symbol when the value is three digits. The value .000001 for the MULT= option is another way to write 1E-06, which is one millionth. Multiplying a value by the millionth, billionth, and trillionth multipliers return the number of millions, billions, and trillions of dollars.

```
proc format;
  picture bigmoney (fuzz=0)
    1E06-<1000000000='0000 M' (prefix='$' mult=.000001)
    1E09-<1000000000000='0000 B' (prefix='$' mult=1E-09)
    1E12-<1000000000000000='0000 T' (prefix='$' mult=1E-012);
run;
```

Generate large numbers to format as dollars.

```
data mult;
  do i=5 to 12;
```

```

        x=16**i;
        put x=comma20. x= bigmoney.;
    end;
run;

```

LOG

Example Code 26.3 Formatted Millions, Billions, and Trillions Dollar Amounts

```

x=1,048,576 x=$1 M
x=16,777,216 x=$16 M
x=268,435,456 x=$268 M
x=4,294,967,296 x=$4 B
x=68,719,476,736 x=$68 B
x=1,099,511,627,776 x=$1 T
x=17,592,186,044,416 x=$17 T
x=281,474,976,710,656 x=$281 T

```

Program

```

proc format;
    picture bigmoney (fuzz=0)
        1E06-<10000000000='0000.99 M' (prefix='$' mult=.0001)
        1E09-<10000000000000='0000.99 B' (prefix='$' mult=1E-07)
        1E12-<10000000000000000='0000.99 T' (prefix='$' mult=1E-010);
run;

data mult;
    do i=5 to 12;
        x=16**i;
        put x=comma20. x= bigmoney.;
    end;
run;

```

Program Description

In this program, the BIGMONEY. format is modified to display a more accurate number by adding decimal values.

Modify the BIGMONEY format. To display a more accurate number, the picture value and the MULT= value are modified. To display two decimal values, .99 is added to the picture. To calculate two decimal values, the value in the MULT= option is reduced from one millionth to one ten-thousandth. When 16^5 is multiplied by .0001, the results is 104.8576. The decimal values are truncated and the 104 is placed in the picture beginning on the right. The resulting formatted value is 1.04 M.

```

proc format;
    picture bigmoney (fuzz=0)
        1E06-<10000000000='0000.99 M' (prefix='$' mult=.0001)
        1E09-<10000000000000='0000.99 B' (prefix='$' mult=1E-07)

```

```
1E12-<10000000000000000='0000.99 T' (prefix='$' mult=1E-010);
run;
```

Generate large numbers to format as dollars.

```
data mult;
  do i=5 to 12;
    x=16**i;
    put x=comma20. x= bigmoney.;
  end;
run;
```

LOG

Example Code 26.4 More Precisely Formatted Large Dollar Amounts

```
x=1,048,576 x=$1.04 M
x=16,777,216 x=$16.77 M
x=268,435,456 x=$268.43 M
x=4,294,967,296 x=$4.29 B
x=68,719,476,736 x=$68.71 B
x=1,099,511,627,776 x=$1.09 T
x=17,592,186,044,416 x=$17.59 T
x=281,474,976,710,656 x=$281.47 T
```

Example 5: Filling a Picture Format

Features: PICTURE statement options
FILL=
PREFIX=

Details

This example does the following tasks:

- prefixes the formatted value with a specified character
- fills the leading blanks with a specified character
- shows the interaction between the FILL= and PREFIX= options

Program

```
data pay;
  input Name $ MonthlySalary;
```

```

        datalines;
Liu    1259.45
Lars   1289.33
Kim    1439.02
Wendy  1675.21
Alex   1623.73
        ;

proc format;
    picture salary low-high='00,000,000.00' (fill='*' prefix='$');
run;

proc print data=pay noobs;
    format monthllysalary salary.;

    title 'Printing Salaries for a Check';
run;

```

Program Description

Create the PAY data set. The PAY data set contains the monthly salary for each employee.

```

data pay;
    input Name $ MonthlySalary;
    datalines;
Liu    1259.45
Lars   1289.33
Kim    1439.02
Wendy  1675.21
Alex   1623.73
    ;

```

Define the SALARY. picture format and specify how the picture will be filled.

When FILL= and PREFIX= PICTURE statement options appear in the same picture, the format places the prefix and then the fill characters. The SALARY. format fills the picture with the fill character because the picture has zeros as digit selectors. The left-most comma in the picture is replaced by the fill character.

```

proc format;
    picture salary low-high='00,000,000.00' (fill='*' prefix='$');
run;

```

Print the PAY data set. The NOOBS option suppresses the printing of observation numbers. The FORMAT statement temporarily associates the SALARY. format with the variable MonthlySalary.

```

proc print data=pay noobs;
    format monthllysalary salary.;

```

Specify the title.

```

    title 'Printing Salaries for a Check';
run;

```

Output

Output 26.5 Printing Salaries for a Check

Printing Salaries for a Check

Name	Monthly Salary
Liu	****\$1,259.45
Lars	****\$1,289.33
Kim	****\$1,439.02
Wendy	****\$1,675.21
Alex	****\$1,623.73

Example 6: Create a Date or Time Format with Directives

Features: PICTURE statement option
DATATYPE=TIME
directives

Details

This example uses directives to format date, time, and datetime values. It also uses directives on more than one value-range pair.

Program

```
proc format;
  picture mytime (round)
    low-<86400='%H hours, %M minutes' (datatype=time) 1
    86400-high='%n days, %H hours, %M minutes' (datatype=time) ; 2
  /* 86400=number of seconds in one day */
run;

data test;
  input xtime;
  newtime=put(xtime,mytime.);
datalines;
```

```

12345
46987
86400
99999
172800
1012345
3333333
;

run;

proc print data=test;
run;

```

Program Description

Use directives with the keywords LOW, HIGH, and LOW-HIGH. The directives format the date, time, and datetime values.

```

proc format;
  picture mytime (round)
    low-<86400='%H hours, %M minutes' (datatype=time) 1
    86400-high='%n days, %H hours, %M minutes' (datatype=time) ; 2
  /* 86400=number of seconds in one day */
run;

```

- 1 The %H and %M directives specify that the hours and minutes are identified in the output. The DATATYPE option enables the use of directives in the picture.
- 2 The %n, %H, %M directives specify that the days, hours, and minutes are identified in the output.

Create the data set.

```

data test;
  input xtime;
  newtime=put(xtime,mytime.);
datalines;
12345
46987
86400
99999
172800
1012345
3333333
;

run;

```

Print the contents of the data set.

```

proc print data=test;
run;

```


Output 26.6 The Test data set with formatted values

The SAS System		
Obs	xtime	newtime
1	12345	3 hours, 25 minutes
2	46987	13 hours, 3 minutes
3	86400	1 days, 0 hours, 0 minutes
4	99999	1 days, 3 hours, 46 minutes
5	172800	2 days, 0 hours, 0 minutes
6	1012345	11 days, 17 hours, 12 minutes
7	3333333	38 days, 13 hours, 55 minutes

Example 7: Change the 24-Hour Clock to 00:00:01–24:00:00

Features: PICTURE statement option
DATATYPE=DATETIME_UTIL

Details

At times, you might need to express midnight as 24:00, or you need to use a datetime hour range 00:00:01–24:00:00. The hour value range for DATATYPE=DATETIME is 00:00:00–23:59:59. This example uses the option DATATYPE=DATETIME_UTIL to express hours in the range 00:00:01–24:00:00, and shows a date change if you use 00:00:00.

Program

```
proc format;
  picture hours (default=19)
    other='%Y-%0m-%0d %0H:%0M:%0S' (datatype=datetime_util);
run;

data _null_;
  x = '01jul2015:00:00:01'dt; put x=hours.;
  x = '01jul2015:00:00:00'dt; put x=hours.;
run;
```

Program Description

Use the DATATYPE=DATETIME_UTIL option to use the hour range 00:00:01–24:00:00.

```
proc format;
  picture hours (default=19)
    other='%Y-%0m-%0d %0H:%0M:%0S' (datatype=datetime_util);
run;
```

Compare Date Values. The first datetime value is in the range 00:00:01 and shows the day as July 1. The second datetime value is not in the range 00:00:01–24:00:00 and shows results as midnight of the previous day.

```
data _null_;
  x = '01jul2015:00:00:01'dt; put x=hours.;
  x = '01jul2015:00:00:00'dt; put x=hours.;
run;
```

Log

Example Code 26.5 Using Hour Range 00:00:01–24:00:00

```
x=2015-07-01 00:01:00
x=2015-06-30 24:00:00
```

Example 8: Creating a Format for Character Values

Features:	VALUE statement option OTHER
Data set:	PROCLIB.STAFF
Format:	USCURRENCY. from Example 2

Details

This example uses a VALUE statement to create a character format that prints a value of a character variable as a different character string.

Program

```
libname proclib 'SAS-library-1';
```

```

libname library 'SAS-library-2';

proc format library=library;
    value $city 'BR1'='Birmingham UK'
               'BR2'='Plymouth UK'
               'BR3'='York UK'
               'US1'='Denver USA'
               'US2'='Miami USA'
               other='INCORRECT CODE';
run;

proc print data=proclib.staff noobs label;
    label salary='Salary in U.S. Dollars';
    format salary uscurrency. site $city.;
    title 'PROCLIB.STAFF with a Format for the Variables';
    title2 'Salary and Site';
run;

```

Program Description

Assign two SAS library references (PROCLIB and LIBRARY). Assigning a library reference LIBRARY is useful in this case because if you use PROC FORMAT, then SAS automatically searches for informats and formats in any library that is referenced with the LIBRARY libref.

```

libname proclib 'SAS-library-1';
libname library 'SAS-library-2';

```

Create the catalog named Library.Formats, where the user-defined formats will be stored. The LIBRARY= option specifies a permanent storage location for the formats that you create. It also creates a catalog named FORMAT in the specified library. If you do not use LIBRARY=, then SAS temporarily stores formats and informats that you create in a catalog named Work.Formats.

```
proc format library=library;
```

Define the \$CITY. format. The special codes BR1, BR2, and so on, are converted to the names of the corresponding cities. The keyword OTHER specifies that values in the data set that do not match any of the listed city code values are converted to the value INCORRECT CODE.

```

    value $city 'BR1'='Birmingham UK'
               'BR2'='Plymouth UK'
               'BR3'='York UK'
               'US1'='Denver USA'
               'US2'='Miami USA'
               other='INCORRECT CODE';
run;

```

Print the PROCLIB.STAFF data set. The NOOBS option suppresses the printing of observation numbers. The LABEL option uses variable labels instead of variable names for column headings.

```
proc print data=proclib.staff noobs label;
```

Specify a label for the Salary variable. The LABEL statement substitutes the label “Salary in U.S. Dollars” for the name SALARY.

```
label salary='Salary in U.S. Dollars';
```

Specify formats for Salary and Site. The FORMAT statement temporarily associates the USCURRENCY. format with the variable SALARY and also temporarily associates the format \$CITY. with the variable SITE.

```
format salary uscurrency. site $city.;
```

Specify the titles.

```
title 'PROCLIB.STAFF with a Format for the Variables';
title2 'Salary and Site';
run;
```

Output

Output 26.7 PROCLIB.STAFF with Formatted Variables for Salary and Site

PROCLIB.STAFF with a Format for the Variables Salary and Site				
Name	IdNumber	Salary in U.S. Dollars	Site	HireDate
Capalleti, Jimmy	2355	\$34,072	Birmingham UK	30JAN09
Chen, Len	5889	\$33,771	Birmingham UK	18JUN06
Davis, Brad	3878	\$31,509	Plymouth UK	20MAR04
Leung, Brenda	4409	\$55,256	Plymouth UK	18SEP94
Martinez, Maria	3985	\$78,980	Miami USA	10JAN93
Orfali, Philip	0740	\$80,648	Miami USA	16FEB03
Patel, Mary	2398	\$56,643	York UK	02FEB90
Smith, Robert	5162	\$64,561	INCORRECT CODE	15APR06
Sorrell, Joseph	4421	\$62,403	Denver USA	19JUN11
Zook, Carla	7385	\$37,010	York UK	18DEC10

Example 9: Creating a Format for Missing and Nonmissing Variable Values

Features: VALUE statement

VALUE statement option
OTHER

Data set: EDUCATION

Details

The EDUCATION data set reports dropout rates and math scores for several states, and indicates a region for each state.

In this example, you use the VALUE statement to create the text value n/a for all math score missing values. All nonmissing math score values are formatted using the 5.1 format.

The example then prints the dropout rate and math scores for each state, by region.

Program

```
options obs=20;

proc format;
  value myfmt .='n/a' other=[5.1];
run;

proc sort data=education;
  by region;
run;

proc print data=education;
  by region;
  var state dropOutRate mathScore;
  format mathScore myfmt.;
run;
```

Program Description

Set the number of observations to print.

```
options obs=20;
```

Create a format for the Mathscore variable values. Use the VALUE statement to create the format MYFMT. for the Mathscore variable. When the program encounters a missing Mathscore value, the value is formatted as n/a. All other values for Mathscore are formatted using the 5.1 format.

```
proc format;
  value myfmt .='n/a' other=[5.1];
run;
```

Sort and print the data. Use PROC SORT to sort the data set by region. To print the data by region, specify the region variable in the PROC PRINT BY statement. To report the state, dropout rate, and math scores, use the VAR statement and specify the state, dropOutRate, and mathScore variables. Finally, use the FORMAT statement to tell SAS to format the mathScore variable using the MYFMT. format.

```
proc sort data=education;
    by region;
run;

proc print data=education;
    by region;
    var state dropOutRate mathScore;
    format mathScore myfmt.;
run;
```

Output

Output 26.8 Dropout Rates and Math Scores for Each State in a Region

The SAS System			
Region=MW			
Obs	State	DropoutRate	MathScore
1	Illinois	21.5	260.0
2	Indiana	13.8	267.0
3	Iowa	13.6	278.0
4	Kansas	17.9	n/a
Region=NE			
Obs	State	DropoutRate	MathScore
5	Connecticut	16.8	270.0
6	Delaware	28.5	261.0
7	Maine	22.5	n/a
8	Maryland	26.0	260.0

Region=SE			
Obs	State	DropoutRate	Math Score
9	Alabama	22.3	252.0
10	Arkansas	11.5	256.0
11	Florida	38.5	255.0
12	Georgia	27.9	258.0
13	Kentucky	32.7	256.0
14	Louisiana	43.1	246.0
Region=W			
Obs	State	DropoutRate	Math Score
15	Alaska	35.8	n/a
16	Arizona	31.2	259.0
17	California	32.7	256.0
18	Colorado	24.7	267.0
19	Hawaii	18.3	251.0
20	Idaho	21.8	272.0

Example 10: Creating an Informat Using Perl Regular Expressions

Features: INVALUE statement options
 REGEXP
 REGEXPE

Details

This example uses two Perl regular expressions to create an informat. The informat using the first expression verifies that the input is an integer and reads the integer. The second informat uses a regular expression that invokes substitution to read a number different from the input value.

Program

```
proc format;
```

```

        invalue isnum (default=5) '/'[0-9]/' (regex) = _same_ other=_error_;
        invalue x1to2x(default=5) 's/1/2/' (regexpe) = _same_ other=_same_;
run;

data _null_;
    input x:isnum. y:x1to2x.;
    put x= y=;
    datalines;
1 121
2 145
a 232
run;

```

Program Description

Create new informats. If the input is a decimal integer, the ISNUM. informat reads the number. Otherwise, SAS writes an error to the log. The X1TO2X. informat substitutes all 1s in the input value with a 2.

```

proc format;
    invalue isnum (default=5) '/'[0-9]/' (regex) = _same_ other=_error_;
    invalue x1to2x(default=5) 's/1/2/' (regexpe) = _same_ other=_same_;
run;

```

Read the data. The first two lines of data are valid. The first input value 121 is formatted as 222 because a 1 is substituted with a 2. The input value of 145 is formatted as 245 using the same substitution rule. The third line produces an error because the value for x is a character.

```

data _null_;
    input x:isnum. y:x1to2x.;
    put x= y=;
    datalines;
1 121
2 145
a 232
run;

```


LOG

```
1  proc format;
2      invalue isnum (default=5) '/'[0-9]/' (regexp) = _same_ other=_error_;
NOTE: Informat ISNUM has been output.
3      invalue xlto2x(default=5) 's/1/2/' (regexpe) = _same_ other=_same_;
NOTE: Informat X1TO2X has been output.
4      run;

NOTE: PROCEDURE FORMAT used (Total process time):
      real time          0.32 seconds
      cpu time           0.07 seconds

5
6  data _null_;
7      input x:isnum. y:xlto2x.;
8      put x= y=;
9      datalines;

x=1 y=222
x=2 y=245
NOTE: Invalid data for x in line 12 1-1.
x=. y=232
RULE:      +---+---1---+---2---+---3---+---4---+---5---+---6---+---7---
+---8---+---
12          a 232
x=. y=232 _ERROR_=1 _N_=3
```

Example 11: Writing a Format for Dates Using a Standard SAS Format and a Color Background

Features:	VALUE statement option HIGH PROC PRINT statement VAR statement STYLE option
Data set:	PROCLIB.STAFF
Format:	USCURRENCY. from Example 3
Format:	\$CITY. from Example 7

Details

This example uses an existing format that is supplied by SAS as a formatted value and color codes values based on dates.

Tasks include the following:

- creating a numeric format

- nesting formats
- writing a format using a standard SAS format
- formatting dates using a color scheme

Program

```
libname proclib 'SAS-library-1';
libname library 'SAS-library-2';

proc format library=library;

    value benefit
        low-'31DEC2008'd=[worddate20.]
        '01JAN2009'd-high=' ** Not Eligible **';

    value color
        low-'31DEC2008'd='light green'
        '01JAN2009'd-high='light red';
run;

proc print data=proclib.staff noobs label;
    var name idnumber salary site;
    var hiredate /style=[background=color.];

    label salary='Salary in U.S. Dollars';

    format salary uscurrency. site $city. hiredate benefit.;

    title 'PROCLIB.STAFF with a Format for the Variables';
    title2 'Salary, Site, and HireDate';
run;
```

Program Description

This program defines a format called BENEFIT., which differentiates between employees hired on or before 31DEC2008. The purpose of this program is to indicate any employees who are eligible to receive a benefit, based on a hire date on or before December 31, 2008. All other employees with a later hire date are listed as ineligible for the benefit.

Assign two SAS library references (PROCLIB and LIBRARY). Assigning a library reference LIBRARY is useful in this case because if you use PROC FORMAT, then SAS automatically searches for informats and formats in any library that is referenced with the LIBRARY libref.

```
libname proclib 'SAS-library-1';
libname library 'SAS-library-2';
```

Store the BENEFIT. format in the catalog Library.Formats. The LIBRARY= option specifies the permanent storage location LIBRARY for the formats that you create. If you do not use LIBRARY=, then SAS temporarily stores formats and informats that you create in the catalog Work.Formats.

```
proc format library=library;
```

Define the first range in the BENEFIT. format. This first range differentiates between the employees who were hired on or before 31DEC2008 and those who were hired after that date. The keyword LOW and the SAS date constant '31DEC2008'd create the first range, which includes all date values that occur on or before December 31, 2008. For values that fall into this range, SAS applies the WORDDATEw. format. For more information about SAS date constants, see [“Dates, Times, and Intervals” in SAS Programmer’s Guide: Essentials](#). For more information about the WORDDATE formats, see [“WORDDATEw. Format” in SAS Formats and Informats: Reference](#).

```
value benefit
  low-'31DEC2008'd=[worddate20.]
  '01JAN2009'd-high='  ** Not Eligible **';
```

Define the colors for the ranges. Using the same date ranges, employees who are eligible for a benefit based on the dates are color coded in light green. Employees who are not eligible for a benefit are color coded in a light red.

```
value color
  low-'31DEC2008'd='light green'
  '01JAN2009'd-high='light red';

run;
```

Print the data set PROCLIB.STAFF. The NOOBS option suppresses the printing of observation numbers. The LABEL option uses variable labels instead of variable names for column headings. The VAR statement names the variables to be printed. The second VAR statement uses the STYLE= option to name the color. format as the background color for the Hiredate variable.

```
proc print data=proclib.staff noobs label;
  var name idnumber salary site;
  var hiredate /style=[background=color.];
```

Specify a label for the Salary variable. The LABEL statement substitutes the label “Salary in U.S. Dollars” for the name SALARY.

```
label salary='Salary in U.S. Dollars';
```

Specify formats for Salary, Site, and Hiredate. The FORMAT statement associates the USCURRENCY. format (created in [“Example 3: Creating a Picture Format” on page 938](#)) with SALARY, the \$CITY. format (created in [“Example 8: Creating a Format for Character Values” on page 948](#)) with SITE, and the BENEFIT. format with HIREDATE.

```
format salary uscurrency. site $city. hiredate benefit.;
```

Specify the titles.

```
title 'PROCLIB.STAFF with a Format for the Variables';
title2 'Salary, Site, and HireDate';

run;
```

Output

Output 26.9 PROCLIB.STAFF with a Format for the Variables Salary, Site, and HireDate

PROCLIB.STAFF with a Format for the Variables Salary, Site, and HireDate				
Name	IdNumber	Salary in U.S. Dollars	Site	HireDate
Capalleti, Jimmy	2355	\$34,072	Birmingham UK	** Not Eligible **
Chen, Len	5889	\$33,771	Birmingham UK	June 18, 2006
Davis, Brad	3878	\$31,509	Plymouth UK	March 20, 2004
Leung, Brenda	4409	\$55,256	Plymouth UK	September 18, 1994
Martinez, Maria	3985	\$78,980	Miami USA	January 10, 1993
Orfali, Philip	0740	\$80,648	Miami USA	February 16, 2003
Patel, Mary	2398	\$56,643	York UK	February 2, 1990
Smith, Robert	5162	\$64,561	INCORRECT CODE	April 15, 2006
Sorrell, Joseph	4421	\$62,403	Denver USA	** Not Eligible **
Zook, Carla	7385	\$37,010	York UK	** Not Eligible **

Example 12: Converting Raw Character Data to Numeric Values

Features: INVALUE statement

Details

This example uses an INVALUE statement to create a numeric informat that converts numeric and character raw data to numeric data.

Program

```
libname proclib 'SAS-library-1';
libname library 'SAS-library-2';

proc format library=library;

    invalue evaluation '0'=4
```

```

        'S'=3
        'E'=2
        'C'=1
        'N'=0;

run;

data proclib.points;
    input EmployeeId $ (Q1-Q4) (evaluation.,+1);
    TotalPoints=sum(of q1-q4);
    datalines;
2355 S O O S
5889 2 2 2 2
3878 C E E E
4409 0 1 1 1
3985 3 3 3 2
0740 S E E S
2398 E E C C
5162 C C C E
4421 3 2 2 2
7385 C C C N
;

proc print data=proclib.points noobs;

    title 'The PROCLIB.POINTS Data Set';
run;

```

Program Description

This program converts quarterly employee evaluation grades, which are alphabetic, into numeric values so that reports can be generated that sum the grades up as points.

Set up two SAS library references, one named PROCLIB and the other named LIBRARY.

```

libname proclib 'SAS-library-1';
libname library 'SAS-library-2';

```

Store the EVALUATION. informat in the catalog Library.Formats.

```

proc format library=library;

```

Create the numeric informat EVALUATION.. The INVALUE statement converts the specified values. The letters O (Outstanding), S (Superior), E (Excellent), C (Commendable), and N (None) correspond to the numbers 4, 3, 2, 1, and 0, respectively.

```

    invalue evaluation 'O'=4
                      'S'=3
                      'E'=2
                      'C'=1
                      'N'=0;

run;

```

Create the PROCLIB.POINTS data set. The instream data, which immediately follows the DATALINES statement, contains a unique identification number

(EmployeeId) and bonus evaluations for each employee for each quarter of the year (Q1–Q4). Some of the bonus evaluation values that are listed in the data lines are numbers; others are character values. Where character values are listed in the data lines, the EVALUATION. informat converts the value O to 4, the value S to 3, and so on. The raw data values 0 through 4 are read as themselves because they are not referenced in the definition of the informat. Converting the letter values to numbers makes it possible to calculate the total number of bonus points for each employee for the year. TotalPoints is the total number of bonus points.

```
data proclib.points;
  input EmployeeId $ (Q1-Q4) (evaluation.,+1);
  TotalPoints=sum(of q1-q4);
  datalines;
2355 S O O S
5889 2 2 2 2
3878 C E E E
4409 0 1 1 1
3985 3 3 3 2
0740 S E E S
2398 E E C C
5162 C C C E
4421 3 2 2 2
7385 C C C N
;
```

Print the PROCLIB.POINTS data set. The NOOBS option suppresses the printing of observation numbers.

```
proc print data=proclib.points noobs;
```

Specify the title.

```
  title 'The PROCLIB.POINTS Data Set';
run;
```

Output

Output 26.10 The PROCLIB.POINT Data Set

The PROCLIB.POINTS Data Set					
EmployeeId	Q1	Q2	Q3	Q4	TotalPoints
2355	3	4	4	3	14
5889	2	2	2	2	8
3878	1	2	2	2	7
4409	0	1	1	1	3
3985	3	3	3	2	11
0740	3	2	2	3	10
2398	2	2	1	1	6
5162	1	1	1	2	5
4421	3	2	2	2	9
7385	1	1	1	0	3

Example 13: Creating a Format from a CNTLIN= Data Set

Features: PROC FORMAT statement option
CNTLIN=
Input control data set

Details

This example shows how to create a format from a SAS data set.

Here are the tasks:

- create a format from an input control data set
- create an input control data set from an existing SAS data set

Program To Create a Temporary Data Set

```

data scale;
  input begin: $char2. end: $char2. amount: $char2.;
  datalines;
0   3   0%
4   6   3%
7   8   6%
9  10   8%
11 16  10%
;

data ctrl;
  length label $ 11;

  set scale(rename=(begin=start amount=label)) end=last;

  retain fmtname 'PercentageFormat' type 'n';

  output;

  if last then do;
    hlo='O';
    label='***ERROR***';
    output;
  end;
run;

proc print data=ctrl noobs;

  title 'The CTRL Data Set';
run;

```

Program Description

Create a temporary data set named scale. The first two variables in the data lines, called BEGIN and END, will be used to specify a range in the format. The third variable in the data lines, called AMOUNT, contains a percentage that will be used as the formatted value in the format. Note that all three variables are character variables as required for PROC FORMAT input control data sets.

```

data scale;
  input begin: $char2. end: $char2. amount: $char2.;
  datalines;
0   3   0%
4   6   3%
7   8   6%
9  10   8%
11 16  10%
;

```

Create the input control data set CTRL and set the length of the LABEL variable. The LENGTH statement ensures that the LABEL variable is long enough to accommodate the label ***ERROR***.

```

data ctrl;
  length label $ 11;

```


Rename variables and create an end-of-file flag. The data set CTRL is derived from WORK.SCALE. RENAME= renames BEGIN and AMOUNT as START and LABEL, respectively. The END= option creates the variable LAST, whose value is set to 1 when the last observation is processed.

```
set scale(rename=(begin=start amount=label)) end=last;
```

Create the variables Fmtname and Type with fixed values. The RETAIN statement is more efficient than an assignment statement in this case. RETAIN retains the value of Fmtname and Type in the program data vector and eliminates the need for the value to be written on every iteration of the DATA step. Fmtname specifies the name PercentageFormat, which is the format that the input control data set creates. The Type variable specifies that the input control data set will create a numeric format.

```
retain fmtname 'PercentageFormat' type 'n';
```

Write the observation to the output data set.

```
output;
```

Create an “other” category. Because the only valid values for this application are 0–16, any other value (such as missing) should be indicated as an error to the user. The IF statement executes only after the DATA step has processed the last observation from the input data set. When IF executes, HLO receives a value of O to indicate that the range is OTHER, and LABEL receives a value of ***ERROR***. The OUTPUT statement writes these values as the last observation in the data set. HLO has missing values for all other observations.

```
if last then do;
  hlo='O';
  label='***ERROR***';
  output;
end;
run;
```

Print the control data set, CTRL. The NOOBS option suppresses the printing of observation numbers.

```
proc print data=ctrl noobs;
```

Specify the title.

```
title 'The CTRL Data Set';
run;
```

Output

Output 26.11 *The CTRL Data Set*

The CTRL Data Set					
label	start	end	fmtname	type	hlo
0%	0	3	PercentageFormat	n	
3%	4	6	PercentageFormat	n	
6%	7	8	PercentageFormat	n	
8%	9	10	PercentageFormat	n	
10%	11	16	PercentageFormat	n	
ERROR	11	16	PercentageFormat	n	O

Program To Create the Format

```
proc format library=work cntlin=ctrl;
run;

proc format library=work;
  invaluel evaluation 'O'=4
                        'S'=3
                        'E'=2
                        'C'=1
                        'N'=0;
run;

data points;
  input EmployeeId $ (Q1-Q4) (evaluation.,+1);
  TotalPoints=q1+q2+q3+q4;
  datalines;
2355 S O O S
5889 2 . 2 2
3878 C E E E
4409 0 1 1 1
3985 3 3 3 2
0740 S E E S
2398 E E C
5162 C C C E
4421 3 2 2 2
7385 C C C N
;

proc report data=work.points nowd headskip split='#';
  column employeeid totalpoints totalpoints=Pctage;
  define employeeid / right;
  define totalpoints / 'Total#Points' right;
  define pctage / format=PercentageFormat12. 'Percentage' left;
```

```

    title 'The Percentage of Salary for Calculating Bonus';
run;

```

Program Description

Store the created format in the catalog Work.Formats and specify the source for the format. The CNTLIN= option specifies that the data set CTRL is the source for the format PercentageFormat.

```

proc format library=work cntlin=ctrl;
run;

```

Create the numeric informat EVALUATION.. The INVALUE statement converts the specified values. The letters O (Outstanding), S (Superior), E (Excellent), C (Commendable), and N (None) correspond to the numbers 4, 3, 2, 1, and 0, respectively.

```

proc format library=work;
    invalue evaluation 'O'=4
                      'S'=3
                      'E'=2
                      'C'=1
                      'N'=0;

run;

```

Create the WORK.POINTS data set. The instream data, which immediately follows the DATALINES statement, contains a unique identification number (EmployeeId) and bonus evaluations for each employee for each quarter of the year (Q1–Q4). Some of the bonus evaluation values that are listed in the data lines are numbers; others are character values. Where character values are listed in the data lines, the Evaluation. informat converts the value O to 4, the value S to 3, and so on. The raw data values 0 through 4 are read as themselves because they are not referenced in the definition of the informat. Converting the letter values to numbers makes it possible to calculate the total number of bonus points for each employee for the year. TotalPoints is the total number of bonus points. The addition operator is used instead of the SUM function so that any missing value will result in a missing value for TotalPoints.

```

data points;
    input EmployeeId $ (Q1-Q4) (evaluation.,+1);
    TotalPoints=q1+q2+q3+q4;
    datalines;
2355 S O O S
5889 2 . 2 2
3878 C E E E
4409 0 1 1 1
3985 3 3 3 2
0740 S E E S
2398 E E C
5162 C C C E
4421 3 2 2 2
7385 C C C N
;

```

Generate a report for WORK.POINTS and associate the PERCENTAGEFORMAT. format with the TotalPoints variable. The DEFINE statement performs the association. The column that contains the formatted values of TotalPoints is using the alias Pctage. Using an alias enables you to print a variable twice, once with a format and once with the default format. For more information about the REPORT procedure, see [Chapter 51, “REPORT Procedure”](#).

```
proc report data=work.points nowd headskip split='#';
  column employeeid totalpoints totalpoints=Pctage;
  define employeeid / right;
  define totalpoints / 'Total#Points' right;
  define pctage / format=PercentageFormat12. 'Percentage' left;
  title 'The Percentage of Salary for Calculating Bonus';
run;
```

Output

Output 26.12 *The Percentage of Salary for Calculating Bonus*

The Percentage of Salary for Calculating Bonus

Employeeid	Total Points	Percentage
2355	14	10%
5889	.	***ERROR***
3878	7	6%
4409	3	0%
3985	11	10%
0740	10	8%
2398	.	***ERROR***
5162	5	3%
4421	9	8%
7385	3	0%

Example 14: Creating an Informat from a CNTLIN= Data Set

Features: PROC FORMAT statement option
CNTLIN=
Input control data set

Details

This example shows how to create an informat from a CNTLIN= data set.

Here are the tasks:

- create an informat from an input control data set
- create an input control data set from an existing SAS data set

Program

```
proc format;
  invalue mytest
    'abc'=1
    'xyz'=2
    other=3;
  invalue $chrtest
    'abc'='xyz'
    other='else';
run;

data _null_;
  input value:mytest. @@;
  put value=;
  datalines;
abc xyz ghi 4
run;

data _null_;
  input value:$chrtest. @@;
  put value=;
  datalines;
abc xyz ghi 4
run;

data temp;
  length start $8 type $1 hlo $1;
  fmtname='newtest'; type='i';
  start='abc'; label=1; hlo=' '; output;
  start='xyz'; label=2; hlo=' '; output;
  start=' '; label=3; hlo='O'; output;
run;

proc format cntlin=temp; run;

data temp;
  length start label $8 type $1 hlo $1;
  fmtname='$newchr'; type='j';
  start='abc'; label='xyz'; hlo=' '; output;
  start=' '; label='else'; hlo='O'; output;
run;
```

```

proc format cntlin=temp; run;

data _null_;
  input value:newtest. @@;
  put value=;
  datalines;
abc xyz ghi 4
run;

data _null_;
  input value:$newchr. @@;
  put value=;
  datalines;
abc xyz ghi 4
run;

data temp;
  length start label $8 hlo $1;
  fmtname='@new2test';
  start='abc'; label='1'; hlo=' '; output;
  start='xyz'; label='2'; hlo=' '; output;
  start=' '; label='3'; hlo='0'; output;
  fmtname='@$new2chr';
  start='abc'; label='xyz'; hlo=' '; output;
  start=' '; label='else'; hlo='0'; output;
run;

proc format cntlin=temp; run;

data _null_;
  input value:new2test. @@;
  put value=;
  datalines;
abc xyz ghi 4
run;

data _null_;
  input value:$new2chr. @@;
  put value=;
  datalines;
abc xyz ghi 4
run;

proc print data=temp noobs;
  title 'The CTRL Data Set';
run;

```

Program Description

Create informats with an INVALUE statement. Create a numeric informat and a character informat.

```

proc format;
  invalue mytest
    'abc'=1

```

```

'xyz'=2
other=3;
invalue $chrtest
'abc'='xyz'
other='else';
run;

```

Use the numeric informat with instream data. The code should produce 1, 2, 3, and 3 again as the results of VALUE.

```

data _null_;
  input value:mytest. @@;
  put value=;
  datalines;
abc xyz ghi 4
run;

```

Use the character informat with instream data. The code should produce xyz, else, else, and else as the results of VALUE.

```

data _null_;
  input value:$chrtest. @@;
  put value=;
  datalines;
abc xyz ghi 4
run;

```

Create the equivalent of the MYTEST numeric informat using a CNTLIN= data set, and use the informat name of NEWTEST. Specify that the FMTNAME variable has the value NEWTEST. Specify the value 'i' or 'l' for the TYPE variable to indicate that it is a numeric informat. Specify a value of O for other, or a blank value, for the HLO variable.

```

data temp;
  length start $8 type $1 hlo $1;
  fmtname='newtest'; type='i';
  start='abc'; label=1; hlo=' '; output;
  start='xyz'; label=2; hlo=' '; output;
  start=' '; label=3; hlo='O'; output;
run;

```

Use the character informat with instream data. The code should produce xyz, else, else, and else as the results of VALUE.

```

proc format cntlin=temp; run;

```

Create a CNTLIN= data set for the character informat. The informat is the equivalent to the \$CHRTEST informat that was created above. Specify the name \$NEWCHR for it. The value of TYPE should be 'j' or 'J' to indicate that it is a character informat.

```

data temp;
  length start label $8 type $1 hlo $1;
  fmtname='$newchr'; type='j';
  start='abc'; label='xyz'; hlo=' '; output;
  start=' '; label='else'; hlo='O'; output;
run;

```

Read in the CNTLIN= data set to create the character informat. This example uses *temp* as the value for CNTLIN. If you are saving the code example, specify a more descriptive name.

```
proc format cntlin=temp; run;
```

Create two DATA steps that are the same as the previously created versions. Use the informat names NEWTEST and \$NEWCHR for these versions.

```
data _null_;
  input value:newtest. @@;
  put value=;
  datalines;
abc xyz ghi 4
run;

data _null_;
  input value:$newchr. @@;
  put value=;
  datalines;
abc xyz ghi 4
run;
```

Show that the FMTNAME value can start with an @ to indicate that it is an informat, and that the type variable is not necessary. Numeric and character informats can be created in the same CNTLIN= data set. The label variable must be a character because character formats are being defined. Numeric values are saved as character strings that contain numeric values.

```
data temp;
  length start label $8 hlo $1;
  fmtname='@new2test';
  start='abc'; label='1'; hlo=' '; output;
  start='xyz'; label='2'; hlo=' '; output;
  start=' '; label='3'; hlo='0'; output;
  fmtname='@$new2chr';
  start='abc'; label='xyz'; hlo=' '; output;
  start=' '; label='else'; hlo='0'; output;
run;

proc format cntlin=temp; run;

data _null_;
  input value:new2test. @@;
  put value=;
  datalines;
abc xyz ghi 4
run;

data _null_;
  input value:$new2chr. @@;
  put value=;
  datalines;
abc xyz ghi 4
run;
```

Print the contents of the CNTLIN= data set. Label the table “The CTRL Data Set.”


```
proc print data=temp noobs;
  title 'The CTRL Data Set';
run;
```

Output 26.13 Contents of the CTRL Data Set

The CTRL Data Set			
start	label	hlo	fmtname
abc	1		@new2test
xyz	2		@new2test
	3	O	@new2test
abc	xyz		@\$new2chr
	else	O	@\$new2chr

Example 15: Printing the Description of Informats and Formats

Features: PROC FORMAT statement option
FMTLIB
SELECT statement

Format: [BENEFIT. from Example 4](#)

Informat: [EVALUATION. from Example 6](#)

Details

This example illustrates how to print a description of an informat and a format. The description shows the values that are read in and written.

Program

```
libname library 'SAS-library';
proc format library=library fmtlib;
  select @evaluation benefit;
  title 'FMTLIB Output for the BENEFIT. Format and the';
  title2 'EVALUATION. Informat';
run;
```

Program Description

Set up a SAS library reference named LIBRARY.

```
libname library 'SAS-library';
```

Print a description of EVALUATION. and BENEFIT. The FMTLIB option prints information about the formats and informats in the catalog that the LIBRARY= option specifies. LIBRARY=LIBRARY points to the Library.Formats catalog.

```
proc format library=library fmtlib;
```

Select an informat and a format. The SELECT statement selects EVALUATION. and BENEFIT., which were created in previous examples. The at sign (@) in front of EVALUATION. indicates that EVALUATION. is an informat.

```
select @evaluation benefit;
```

Specify the titles.

```
title 'FMTLIB Output for the BENEFIT. Format and the';
title2 'EVALUATION. Informat';
run;
```

Output

Output 26.14 FMTLIB Output for the BENEFIT Format and the EVALUATION Informat

**FMTLIB Output for the BENEFIT. Format and the
EVALUATION. Informat**

FORMAT NAME: BENEFIT LENGTH: 20 NUMBER OF VALUES: 2 MIN LENGTH: 1 MAX LENGTH: 40 DEFAULT LENGTH: 20 FUZZ: STD		
START	END	LABEL (VER. V7 V8 17SEP2012:15:53:27)
LOW 17898	HIGH 17897	[WORDDATE20.] ** Not Eligible **

INFORMAT NAME: @EVALUATION LENGTH: 1 MIN LENGTH: 1 MAX LENGTH: 40 DEFAULT LENGTH: 1 FUZZ: 0		
START	END	INVALUE(VER. 9.4 17SEP2012:16:10:24)
C	C	1
E	E	2
N	N	0
O	O	4
S	S	3

Example 16: Retrieving a Permanent Format

Features: PROC FORMAT statement option
LIBRARY=
FMTSEARCH= system option

Data set: [SAMPLE](#)

This example uses the LIBRARY= option and the FMTSEARCH= system option to store and retrieve a format stored in a catalog other than Work.Formats or Library.Formats.

Program

```
libname proclib 'SAS-library';
proc format library=proclib;
picture nozeros (fuzz=0)
    low - -1 = '000.00' (prefix='-')
    -1 < - < -.99 = '0.99' (prefix='-' mult=100)
    -0.99 < - < 0 = '99' (prefix='-' mult=100)
    0 = '0.99'
    0 < - < .99 = '99' (prefix='.' mult=100)
    0.99 - <1 = '0.99' (prefix='.' mult=100)
    1 - high = '00.99';
run;

options fmtsearch=(proclib);

data sample;
    input Amount;
    datalines;
-2.051
-.05
-.017
0
.093
.54
.556
6.6
14.63
0.996
-0.999
-45.00
;
run;

proc print data=sample;
    format amount nozeros.;

    title1 'Retrieving the NOZEROS. Format from PROCLIB.FORMATS';
```

```

    title2 'The SAMPLE Data Set';
run;

```

Program Description

Set up a SAS library reference named PROCLIB.

```
libname proclib 'SAS-library';
```

Store the NOZEROS. format in the PROCLIB.FORMATS catalog.

```
proc format library=proclib;
```

Create the NOZEROS. format. The PICTURE statement defines the picture format NOZEROS. See [“Details” on page 901](#).

```

picture nozeros (fuzz=0)
    low - -1 = '000.00' (prefix='-')
    -1 < - < -.99 = '0.99' (prefix='-' mult=100)
    -0.99 < - < 0 = '99' (prefix='-' mult=100)
    0 = '0.99'
    0 < - < .99 = '99' (prefix='.' mult=100)
    0.99 - <1 = '0.99' (prefix='.' mult=100)
    1 - high = '00.99';
run;

```

Add the PROCLIB.FORMATS catalog to the search path that SAS uses to find user-defined formats. The FMTSEARCH= system option defines the search path. The FMTSEARCH= system option requires only a libref. FMTSEARCH= assumes that the catalog name is FORMATS if no catalog name appears. Without the FMTSEARCH= option, SAS would not find the NOZEROS. format. For more information, see [“FMTSEARCH= System Option” in SAS System Options: Reference](#).

```
options fmtsearch=(proclib);
```

Create the sample data set.

```

data sample;
    input Amount;
    datalines;
-2.051
-.05
-.017
0
.093
.54
.556
6.6
14.63
0.996
-0.999
-45.00
;
run;

```

Print the SAMPLE data set. The FORMAT statement associates the NOZEROS. format with the Amount variable.

```
proc print data=sample;
  format amount nozeros.;
```

Specify the titles.

```
  title1 'Retrieving the NOZEROS. Format from PROCLIB.FORMATS';
  title2 'The SAMPLE Data Set';
run;
```

Output

Output 26.15 Retrieving the NOZEROS. Format from PROCLIB.FORMATS

Retrieving the NOZEROS. Format from PROCLIB.FORMATS The SAMPLE Data Set

Obs	Amount
1	-2.05
2	-.05
3	-.01
4	.00
5	.09
6	.54
7	.55
8	6.60
9	14.63
10	.99
11	-.99
12	45.00

Example 17: Writing Ranges for Character Strings

Features: VALUE statement

Data set: PROCLIB.STAFF

This example creates a format and shows how to use ranges with character strings.

Program

```
libname proclib'SAS-library';  
data train;  
    set proclib.staff(keep=name idnumber);  
run;  
proc print data=train noobs;  
    title 'The TRAIN Data Set without a Format';  
run;
```

Program Description

```
libname proclib'SAS-library';
```

Create the TRAIN data set from the PROCLIB.STAFF data set. PROCLIB.STAFF was created in [“Example 2: Create the Example Data Set” on page 937](#).

```
data train;  
    set proclib.staff(keep=name idnumber);  
run;
```

Print the data set TRAIN without a format. The NOOBS option suppresses the printing of observation numbers.

```
proc print data=train noobs;
```

Specify the title.

```
title 'The TRAIN Data Set without a Format';  
run;
```

Output

Output 26.16 The TRAIN Data Set without a Format

The TRAIN Data Set without a Format

Name	IdNumber
Capalleti, Jimmy	2355
Chen, Len	5889
Davis, Brad	3878
Leung, Brenda	4409
Martinez, Maria	3985
Orfali, Philip	0740
Patel, Mary	2398
Smith, Robert	5162
Sorrell, Joseph	4421
Zook, Carla	7385

Store the format in Work.Formats. Because the LIBRARY= option does not appear, the format is stored in Work.Formats and is available only for the current SAS session.

```
proc format;
```

Create the \$SKILLTEST. format. The \$SKILLTEST. format prints each employee's identification number and the skills test that they have been assigned. Employees must take either TEST A, TEST B, or TEST C, depending on their last name. The exclusion operator (<) excludes the last value in the range. Thus, the first range includes employees whose last name begins with any letter from A through D, and the second range includes employees whose last name begins with any letter from E through M. The tilde (~) in the last range is necessary to include an entire string that begins with the letter Z.

```
value $skilltest 'a'-'<'e','A'-'<'E'='Test A'
                'e'-'<'m','E'-'<'M'='Test B'
                'm'-'z~','M'-'Z~'='Test C';

run;
```

Generate a report of the TRAIN data set. The FORMAT= option in the DEFINE statement associates the \$SKILLTEST. format with the Name variable. The column that contains the formatted values of Name is using the alias Test. Using an alias enables you to print a variable twice, once with a format and once with the default format. For more information, see [Chapter 51, "REPORT Procedure"](#).

```
proc report data=train nowd headskip;
  column name name=test idnumber;
  define test / display format=$skilltest. 'Test';
```

```

define idnumber / center;
title 'Test Assignment for Each Employee';
run;

```

Output

Output 26.17 *Test Assignment for Each Employee*

Test Assignment for Each Employee

Name	Test	IdNumber
Capalleti, Jimmy	Test A	2355
Chen, Len	Test A	5889
Davis, Brad	Test A	3878
Leung, Brenda	Test B	4409
Martinez, Maria	Test C	3985
Orfali, Philip	Test C	0740
Patel, Mary	Test C	2398
Smith, Robert	Test C	5162
Sorrell, Joseph	Test C	4421
Zook, Carla	Test C	7385

Example 18: Creating a Format in a non-English Language

Features: PICTURE statement options
 DATATYPE=
 LANGUAGE=
 LOCALE= [system option](#)

Details

This example does the following tasks:

- Creates picture formats using directives for formatting date and datetime values by using the DATATYPE= statement option.

- Uses the LOCALE= system option to specify the locale for German.
- Prints date and datetime values to the SAS log in German using the picture formats.
- Prints a datetime value in French to the log by using the picture format that specifies LANGUAGE=French.

Program

```
proc format;
  picture mdy(default=8) other='%0d%0m%Y' (datatype=date);
  picture langtsda (default=50) other='%A, %d %B, %Y' (datatype=date);
  picture langtsdt (default=50) other='%A, %d,%B, %Y %H %M %S'
    (datatype=datetime);
  picture langtsfr (default=50) other='%A, %d %B, %Y %H %M %S'
    (datatype=datetime language=french);
  picture alltest (default=100)
    other='%a %A %b %B %d %H %I %j %m %M %p %S %w %U %y %%'
    (datatype=datetime);
run;
option locale = de_DE;

data _null_ ;
  a= 18903;
  b = 1633239000;
  put a= mdy.;
  put a= langtsda.;
  put b= langtsdt.;
  put b= langtsfr.;
  put b= alltest.;
run ;
```

Program Description

Create formats using the PICTURE statement. Each PICTURE statement specifies the date or datetime values to format using directives. %A prints a full weekday name. %B prints a full month name. %d prints the day of the month. %Y prints the year. %H prints the hour (24-hour clock). %M prints the minute. %S prints the seconds. The first three formats print the date or datetime in the language specified by the current value of the LOCALE= system option. The format LANGTSFT. prints the datetime in French. For the remaining directives, see the [PICTURE statement on page 887](#).

```
proc format;
  picture mdy(default=8) other='%0d%0m%Y' (datatype=date);
  picture langtsda (default=50) other='%A, %d %B, %Y' (datatype=date);
  picture langtsdt (default=50) other='%A, %d,%B, %Y %H %M %S'
    (datatype=datetime);
  picture langtsfr (default=50) other='%A, %d %B, %Y %H %M %S'
    (datatype=datetime language=french);
  picture alltest (default=100)
```

```

other='%a %A %b %B %d %H %I %j %m %M %p %S %w %U %y %%'
(datatype=datetime);

run;

```

Set the LOCALE= system option. de_DE is the locale value for Germany.

```
option locale = de_DE;
```

Print date and datetime values in German and French. The DATA step prints to the SAS log the date and datetime information for 3 October, 2011, 05:30:00 AM. All values are written in German except for the value of b when it is formatted using the LANGTSFR. format. The LANGTSFR. format prints the datetime value in French.

```

data _null_ ;
  a= 18903;
  b = 1633239000;
  put a= mdy.;
  put a= langtsda.;
  put b= langtsdt.;
  put b= langtsfr.;
  put b= alltest.;
run ;

```

The SAS Log Displaying Picture Format Output in German and French

```

1  proc format;
2      picture mdy(default=8) other='%0d%0m%Y' (datatype=date);
NOTE: Format MDY has been output.
3      picture langtsda (default=50) other='%A, %d %B, %Y' (datatype=date);
NOTE: Format LANGTSDA has been output.
4      picture langtsdt (default=50) other='%A, %d,%B, %Y %H %M %S'
5                                     (datatype=datetime);
NOTE: Format LANGTSDT has been output.
6      picture langtsfr (default=50) other='%A, %d %B, %Y %H %M %S'
7                                     (datatype=datetime language=french);
NOTE: Format LANGTSFR has been output.
8      picture alltest (default=100)
9          other='%a %A %b %B %d %H %I %j %m %M %p %S %w %U %y %%'
10             (datatype=datetime);
NOTE: Format ALLTEST has been output.
11  run;

NOTE: PROCEDURE FORMAT used (Total process time):
      real time          0.03 seconds
      cpu time           0.03 seconds

12
13  option locale = de_DE;
14
15  data _null_ ;
16      a= 18903;
17      b = 1633239000;
18      put a= mdy.;
19      put a= langtsda.;
20      put b= langtsdt.;
21      put b= langtsfr.;
22      put b= alltest.;
23  run ;

a=03102011
a=Montag, 3 Oktober, 2011
b=Montag, 3,Oktober, 2011 5 30 0
b=Lundi, 3 octobre, 2011 5 30 0
b=Mo Montag Okt Oktober 3 5 5 276 10 30 AM 0 2 40 11 %

```

Example 19: Creating a Locale-Specific Format Catalog

Features: PROC FORMAT LOCALE option
FMTSEARCH= system option

Details

This example demonstrates how to create a format in two languages, English and Romanian, and how to access the English and Romanian format catalogs to print a data set in the two languages. The example works best if the SAS session encoding is a latin 2 encoding that supports the Romanian locale.

Program

```
/*no locale information*/
proc format lib=work.formats;
value age low - 5 = 'baby'
6 - 12 = 'child'
13 - 15 = 'teen'
16 - 30 = 'youth'
31 - 50 = 'midlife'
51 - high = 'older';
run;

options locale=ro_RO;

proc format lib = work.formats locale;
value age low - 5 = 'Copil'
6 - 12 = 'Copil'
13 - 15 = 'Adolescent'
16 - 30 = 'Tineretului'
31 - 50 = 'Asta vrei'
51 - high = 'Mai vechi';
run;

options fmtsearch=(work/locale);

/* Set the locale back to English(US) */
options locale=en_US;

data datatst;
input age sex $;
attrib age format= age.;
cards;
5 M
6 F
12 M
13 F
15 M
16 F
30 M
35 F
51 M
100 F
;
```

```

run;
/* Use the English format catalog*/
title "Locale is English, Use the Original Format Catalog";
proc print data=datatst; run;

/* Use the Romanian format catalog*/
options locale=ro_RO;
title 'Locale is ro_RO, Use the Romanian Format Catalog';
proc print data=datatst;run;

```

Program Description

Create the AGE. format in English.

```

/*no locale information*/
proc format lib=work.formats;
value age low - 5 = 'baby'
6 - 12 = 'child'
13 - 15 = 'teen'
16 - 30 = 'youth'
31 - 50 = 'midlife'
51 - high = 'older';
run;

```

Change the locale and create the AGE. format in a locale-specific format catalog.

Using the LOCALE= system option, the locale is change to the Romanian locale. In the PROC FORMAT statement, the LOCALE option specifies to create a format catalog that corresponds to the current locale, which is for the Romanian language.

```

options locale=ro_RO;

proc format lib = work.formats locale;
value age low - 5 = 'Copil'
6 - 12 = 'Copil'
13 - 15 = 'Adolescent'
16 - 30 = 'Tineretului'
31 - 50 = 'Asta vrei'
51 - high = 'Mai vechi';
run;

```

Add the locale-specific format catalogs to the format search path. The FMTSEARCH= system option specifies the format catalog to search. Because you can create more than one locale-specific catalog, when /LOCALE is added to a libref in the search list, SAS searches for a catalog that is associated with the current locale.

```

options fmtsearch=(work/locale);

```

Create a data set and print it using the English format catalog. The LOCALE= system option sets the locale to English.

```

/* Set the locale back to English(US) */
options locale=en_US;

data datatst;
input age sex $;

```

```

attrib age format= age.;
cards;
5 M
6 F
12 M
13 F
15 M
16 F
30 M
35 F
51 M
100 F
;
run;
/* Use the English format catalog*/
title "Locale is English, Use the Original Format Catalog";
proc print data=datatst; run;

```

Print the data set Using the Romanian format catalog. Using the `LOCALE=` system option, the locale is set to the Romanian locale.

```

/* Use the Romanian format catalog*/
options locale=ro_RO;
title 'Locale is ro_RO, Use the Romanian Format Catalog';
proc print data=datatst;run;

```

Here is the data set printed using the English and Romanian format catalogs:

Output 26.18 A Data Set Printed Using an English and Romanian Format Catalog**Locale is English, Use the Original Format Catalog**

Obs	age	sex
1	baby	M
2	child	F
3	child	M
4	teen	F
5	teen	M
6	youth	F
7	youth	M
8	midlife	F
9	older	M
10	older	F

Locale is ro_RO, Use the Romanian Format Catalog

Obs	age	sex
1	Copil	M
2	Copil	F
3	Copil	M
4	Adolescent	F
5	Adolescent	M
6	Tineretului	F
7	Tineretului	M
8	Asta vrei	F
9	Mai vechi	M
10	Mai vechi	F

Example 20: Creating a Function to Use as a Format

Features:

- PROC FCMP statement
- CMPLIB= system option
- PROC FORMAT statement
- Function as a format feature

Details

This example creates a function that converts temperatures from Celsius to Fahrenheit and Fahrenheit to Celsius. The program uses the function as a function in one DATA step and then as a format in another DATA step.

Program

```
proc fcmp outlib=library.functions.smd;
  function ctof(c) $;
    return(cats(((9*c)/5)+32, 'F'));
  endsub;

  function ftoc(f) $;
    return(cats((f-32)*5/9, 'C'));
  endsub;

run;

options cmplib=(library.functions);

data _null_;
  f=ctof(100);
  put f=;
run;

proc format;
  value ctof (default=10) other=[ctof()];
  value ftoc (default=10) other=[ftoc()];
run;

data _null_;
  c=100;
  put c=ctof.;
  f=212;
  put f=ftoc.;
run;
```


Program Description

Create the functions that change temperature from Celsius to Fahrenheit and Fahrenheit to Celsius. The FCMP procedure creates the CTOF function to convert Celsius temperatures to Fahrenheit and the FTOC to convert Fahrenheit temperatures to Celsius.

```
proc fcmp outlib=library.functions.smd;
  function ctof(c) $;
    return(cats(((9*c)/5)+32, 'F'));
  endsub;

  function ftoc(f) $;
    return(cats((f-32)*5/9, 'C'));
  endsub;

run;
```

Access the function library. The CMPLIB system option enables the functions to be included during program compilation.

```
options cmplib=(library.functions);
```

Use the function as a function in a SAS program.

```
data _null_;
  f=ctof(100);
  put f=;
run;
```

Create user-defined formats using the functions. The name of the format is the name of the function. When you use a function as a format, you can nest the format as shown by the OTHER keyword.

```
proc format;
  value ctof (default=10) other=[ctof()];
  value ftoc (default=10) other=[ftoc()];
run;
```

Use the function as a format. This DATA step formats temperatures using a named PUT statement, where you assign a format to a variable in the PUT statement.

```
data _null_;
  c=100;
  put c=ctof.;
  f=212;
  put f=ftoc.;
run;
```

Output: Log

Example Code 26.6 The SAS Log After Creating a Function to Use as a Format

```

323 proc fcmp outlib=library.functions.smd;
324     function ctof(c) $;
325         return(cats(((9*c)/5)+32, 'F'));
326     endsub;
327
328     function ftoc(f) $;
329         return(cats((f-32)*5/9, 'C'));
330     endsub;
331
332 run;

NOTE: Function ftoc saved to library.functions.smd.
NOTE: Function ctof saved to library.functions.smd.
NOTE: PROCEDURE FCMP used (Total process time):
      real time          17.59 seconds
      cpu time           1.26 seconds

333
334 options cmplib=(library.functions);
335
336 data _null_;
337     f=ctof(100);
338     put f;
339 run;

f=212F
NOTE: DATA statement used (Total process time):
      real time          0.50 seconds
      cpu time           0.01 seconds

340
341 proc format;
342     value ctof (default=10) other=[ctof()];
NOTE: Format CTOF has been output.
343     value ftoc (default=10) other=[ftoc()];
NOTE: Format FTOC has been output.
344     run;

NOTE: PROCEDURE FORMAT used (Total process time):
      real time          0.00 seconds
      cpu time           0.00 seconds

345
346 data _null_;
347     c=100;
348     put c=ctof.;
349     f=212;
350     put f=ftoc.;
351 run;

c=212F
f=100C

```

Example 21: Using a Format to Create a Drill-down Table

Features: VALUE statement
 PROC PRINT FORMAT statement

Details

This example creates an HTML table that has population information about five U.S. states. The name of the state is a link to the state's website. The link is created using a user-defined format to format the state name. This example does the following:

- creates the data set that contains the state population information
- creates a user-defined format using the VALUE statement, where the value is an HTML link (<a>) element
- defines the name of the HTML file and the titles for the HTML file
- prints the HTML table using the user-defined format

Program

```
data mydata;
    format population comma12.0;
    label st='State';
    label population='Population';
    input st $ 1-2 population;
    year=2000;
    datalines;
VA 7078515
NC 8049313
SC 4012012
GA 8186453
FL 15982378
;
run;

proc format;
value $COMPND
'VA'='<a href=http://www.va.gov>VA</a>'
'NC'='<a href=http://www.nc.gov>NC</a>'
'SC'='<a href=http://www.sc.gov>SC</a>'
'GA'='<a href=http://www.ga.gov>GA</a>'
```

```

'FL'='<a href=http://www.fl.gov>FL</a>';
run;

ods html file="c:\mySAS\html\Drilldown.htm"
  (title="An ODS HTML Drill-down Table Using a User-defined Format in
the PRINT
  Procedure");

title h=.25in "Year 2000 U.S. Census Population";
title2 color=gray "An ODS HTML Drill-down Table Using a User-defined
Format in
the PRINT Procedure";
footnote color=gray "(Click the underlined text to drill down.)";

options nodate;
proc print data=mydata label noobs;
  var st population;
  format st $compnd. ;
run;

ods html close;
ods html;

```

Program Description

Create the data set. The mydata DATA step creates a data set that contains information about five U.S. state populations based on the census taken in the year 2000. The variables that are created assign data for the year of the census, the state abbreviations, and the state population.

```

data mydata;
  format population comma12.0;
  label st='State';
  label population='Population';
  input st $ 1-2 population;
  year=2000;
  datalines;
VA 7078515
NC 8049313
SC 4012012
GA 8186453
FL 15982378
;
run;

```

Create the \$COMPND. format. The \$COMPND. format formats each state as a link to the state's respective website.

```

proc format;
value $COMPND
'VA'='<a href=http://www.va.gov>VA</a>'
'NC'='<a href=http://www.nc.gov>NC</a>'
'SC'='<a href=http://www.sc.gov>SC</a>'
'GA'='<a href=http://www.ga.gov>GA</a>'
'FL'='<a href=http://www.fl.gov>FL</a>';
run;

```

Set up the table filename and table titles. The ODS HTML FILE= option names the directory and filename where SAS saves the HTML output.

```
ods html file="c:\mySAS\html\Drilldown.htm"
  (title="An ODS HTML Drill-down Table Using a User-defined Format in
the PRINT
Procedure");

title h=.25in "Year 2000 U.S. Census Population";
title2 color=gray "An ODS HTML Drill-down Table Using a User-defined
Format in
the PRINT Procedure";
footnote color=gray "(Click the underlined text to drill down.);"
```

Print the table and close and reopen the HTML destination. The PRINT procedure uses the format \$COMPND. to format the state name. The formatted name is a link to the state's respective website. The ODS HTML statements close and reopen the HTML destination so that future output does not overwrite the HTML file that you just created.

```
options nodate;
proc print data=mydata label noobs;
  var st population;
  format st $compnd. ;
run;

ods html close;
ods html;
```

Output

Output 26.19 Using a Format to Create Drill-down Text in an HTML Table

Year 2000 U.S. Census Population	
An ODS HTML Drill-down Table Using a User-defined Format in the PRINT Procedure	
State	Population
<u>VA</u>	7,078,515
<u>NC</u>	8,049,313
<u>SC</u>	4,012,012
<u>GA</u>	8,186,453
<u>FL</u>	15,982,378

(Click the underlined text to drill down.)

GATEWAY Procedure

Overview: GATEWAY Procedure	993
What Does the Gateway Procedure Do?	994
PROC Gateway Features	994
Concepts: GATEWAY Procedure	994
External Language Access and Required Libraries	994
Computational Resources	995
Displayed Output	995
ODS Tables	995
Syntax: GATEWAY Procedure	995
PROC GATEWAY Statement	996
ARGS Statement	996
OPTS Statement	997
SUBMIT Statement	998
ENDSUBMIT Statement	998
Python Syntax Statement	999
R Syntax Statement	1001
Usage: GATEWAY Procedure	1004
PROC GATEWAY Compared with Other SAS Procedures	1004
Converting Data Types	1005
Submitting Open Source Statements	1006
Selecting the Number of Sessions	1007
Reading and Writing CAS Tables	1007

Overview: GATEWAY Procedure

What Does the Gateway Procedure Do?

IMPORTANT PROC Gateway is available starting with the 2024.09 release.

The Gateway procedure enables the user to run multiple sessions of supported open source engines for parallel processing and high efficiency of data transfer between SAS and open source. Users can use Python and R to interact with large SAS tables that are securely stored in SAS.

PROC Gateway Features

The main features of PROC GATEWAY are as follows:

- Launch multiple Python or R inside the CAS workers.
- Read and write data from the CAS server efficiently.

Concepts: GATEWAY Procedure

External Language Access and Required Libraries

PROC GATEWAY uses the [External Languages Access Control Configuration](#) settings instead of environment variables to authorize the use of Python and R interpreters for your environment.

The following libraries must be installed in the Python and R environments in order to use the PROC GATEWAY:

- Python: must have pyarrow (8 or higher) and pandas libraries installed.
- R: must have arrow (8 or higher), jsonlite, and logger libraries installed

Computational Resources

The computational usage depends on what is being used in the Open Source code. Since Python and R sessions are launched inside of the CAS pod, they share resources with the CAS server. It is typically less efficient to write an entire CAS table to a single Open Source session because a single pod might not have enough resources to hold the entire table. There are methods that enable you to read and write tables in batches (“[new_table_reader](#)” Python and “[new_table_reader](#)” R) to avoid out of memory errors and crashes.

Displayed Output

When using PROC GATEWAY, any messages printed by the Open Source client are printed to the SAS log. You can print results from the Open Source client to the SAS log using “[return_result](#)” Function (Python) or “[return_result](#)” Function (R).

ODS Tables

By default, PROC GATEWAY does not produce any ODS tables. If you want PROC GATEWAY to return any table to the client, you must use the “[return_table](#)” Function (Python) or “[return_table](#)” Function (R).

Syntax: GATEWAY Procedure

```
PROC GATEWAY <proc-options>;  
ARGS <options>;  
OPTS <options>;  
SUBMIT;  
ENDSUBMIT;
```

PROC GATEWAY Statement

The GATEWAY procedure runs code that was written in another language such as Python or R

Restriction: PROC Gateway is available starting with the 2024.09 release.

Syntax

PROC GATEWAY *<options>*;

Optional Arguments

CODE_FILE=*"string"*

specifies the path to a file to be parsed and sent to the Python or R client. This argument is ignored if you use the [SUBMIT](#) and [ENDSUBMIT](#) statements.

LANG=PYTHON | PY | R

Default PYTHON

specifies the language engine to be called.

SINGLE

specifies to restrict the code execution to a single session in a single thread.

NTHREADS=*integer*

specifies the number of threads that are used to run the program. For distributed servers, this value specifies the number of threads on each worker to use for running the program. Leave this value unspecified or set to -1 to use all available threads.

ARGS Statement

Specifies a key pair list of arguments that is passed to the Open Source language.

Syntax

ARGS *<options>*

Optional Arguments

KEY=*"string"*

Alias K

Note It is recommended to always specify a unique KEY= value for each argument in the ARGS statement.

specifies the key name of the argument list.

NUMVAL=*number*

Aliases NVAL

NUM

N

specifies numeric argument value.

STRVAL=*"string"*

Aliases SVAL

STR

S

specifies the string argument value.

Details

The specified arguments can be accessed through `args` (Python) and `gw$args` (R).

OPTS Statement

Runs code that was written in another language such as Python or R.

Syntax

OPTS <options> ;

Optional Arguments

JITTER_TIMEOUT_MILLIS=*integer*

Default 60000

specifies the maximum variance between thread synchronization. Higher values are recommended when one thread or session has more work than another. Set to *-1* for no synchronization time out.

TIMEOUT_MILLIS=*integer*

Default 60000

specifies the maximum idle time out to time out when waiting for the external lang code written in the other language to be executed.

LOG_LEVEL=DEBUG | FATAL | WARN | INFO | NOTSET | ERROR

Default INFO

specifies the log level.

SUBMIT Statement

Identifies the beginning of a block of Python or R code. Enter Python or R statements between the SUBMIT and ENDSUBMIT statements.

Note: Each SUBMIT statement must have a corresponding ENDSUBMIT statement.

Syntax

SUBMIT;

Without Arguments

The SUBMIT statement in PROC GATEWAY has no arguments.

ENDSUBMIT Statement

Identifies the end of a block of Python or R statements.

Important: The ENDSUBMIT statement must be specified on a new line after your code block.

Syntax

ENDSUBMIT;

Without Arguments

The ENDSUBMIT statement in PROC GATEWAY has no arguments.

Python Syntax

The following Python functions and variables are available to use in PROC GATEWAY using the gateway module.

Details

read_table

Reads a table available on the CAS server to the Python client as a data frame or arrow table.

read_table(table: *dict*, repeat: *boolean*, as_data_frame: *boolean*, batch_size: *integer*)

table: dict

specifies a dictionary listing the table name, caslib and other parameters that can be passed to castables.

repeat: boolean

Default FALSE

specifies whether to read the table in all of the Python threads and sessions.

as_data_frame: boolean

Default TRUE

specifies whether to return the input table as a `pd.DataFrame`. When set to false, the input table is returned as an Apache Arrow table `pa.table`.

batch_size: integer

Default default_read_batch_row_count

specifies the number of rows to return per batch.

write_table

writes a table from the Python client to the CAS server.

write_table(df: *pd.DataFrame* | *pa.Table*, table: *dict*)

df: pd.DataFrame | pa.Table

specifies the input pandas data frame or Arrow table.

table: dict

specifies a dictionary listing the table name, caslib and other parameters that can be passed to castables.

Note It is recommended to always check how your table is being distributed to each of your Open Source sessions when writing tables.

new_table_reader

Creates a CAS table reader object.

new_table_reader(table: *dict*, repeat: *boolean*, batch_size: *integer*)

table: *dict*

specifies a dictionary listing the table name, caslib and other parameters that can be passed to castables

repeat: *boolean*

Default FALSE

specifies whether to read the table in all of the Python sessions.

Note: The returned object has the following methods: read_batch(), read_all() and read_all_pa().

new_table_writer

Creates a CAS batch writer object.

new_table_writer(table: *dict*)

table: *dict*

specifies a dictionary listing the table name, caslib, and other parameters that can be passed to castables.

Note: The batch writer object can use the write_batch(pd.DataFrame | pa.Table) method to write batch tables. The batch writer object can be closed by using the close() method.

return_table

Returns the table from the Python client as an ODS table. Tables from all python threads are binded together.

return_table(name: *str*, df: *pd.DataFrame* | *pa.Table*, label: *str*, title: *str*)

name: *str*

specifies the table name.

df: *pd.DataFrame* | *pa.Table*

specifies the input pandas data frame or Arrow table.

label: *str*

Default ""

specifies the table label.

title: *str*

Default ""

specifies the table title.

return_result

Prints the list of arguments to the SAS log.

return_result(name: *str*, **kwargs)

name: *str*

specifies the list name.

****kwargs**

Specifies the arguments to be returned. For example, `arg1='123'`, `arg2=.2`, `arg3=[0, 2]`.

Python Variables

The following variables are available for use in your Python sessions while using PROC GATEWAY.

Table 27.1 *Python Variables*

Variable Name	Variable Description
gateway.num_threads	The total number of threads.
gateway.num_workers	The total number of workers.
gateway.thread_id	The current session thread ID (starts with 0).
gateway.worker_id	The current session worker ID (starts with 0).
gateway.job_id	The current job ID.
gateway.user_id	The current user ID.
gateway.args	List of arguments passed by the “ARGS Statement” .

R Syntax

The following R functions and variables are available to use in your R sessions in PROC GATEWAY using the gateway package. If you are loading packages that mask R's gateway packages, you can access the functions using `gateway::function_name(...)` and the variables using `gateway::gw`.

Details

read_table

Reads a table available on the CAS server to the R client as a data frame or arrow table.

read_table(table= *list*, rep = *boolean*, as_data_frame = *boolean*, batch_size = *integer*)

table= *list*

specifies a list containing the table name, caslib and other parameters that can be passed to castables.

repeat= *boolean*

Default FALSE

specifies whether to read the whole table in each of the R sessions, otherwise it is distributed following CAS rules

as_data_frame= *boolean*

Default TRUE

Specifies whether to return the input table as a data.frame. When set to false the input table is returned as an Arrow Table object.

batch_size= *integer*

Default 8192

specifies the number of rows to return per batch.

write_table

Writes a table from the R client to the CAS server.

write_table(df= *data.frame* | *arrowTable*, table= *list*)

df= *data.frame* | *table*

specifies the input pandas data frame or Arrow table.

table= *list*

specifies a list containing the table name, caslib and other parameters that can be passed to castables.

new_table_reader

Creates a CAS table reader object.

Note: The returned object has the following methods read_batch() and read_table().

new_table_reader(table= *list*, repeat= *boolean*, batch_size= *integer*)

tabe= *list*

specifies a list containing the table name, caslib and other parameters that can be passed to castables.

repeat= *boolean*

Default FALSE

Specifies whether to read the table in all of the R sessions.

batch_size= *integer*

Default 8192

specifies the number of rows to return per batch.

new_table_writer

Creates a CAS batch writer object.

Note: The batch writer object can use the `write_batch(data.frame | Table)` method to write batch tables. The batch writer object can be closed by using the `close()` method.

new_table_writer(table= *list*)**tabe= *list***

specifies a list containing the table name, caslib and other parameters that can be passed to castables.

return_table

Returns table from the R client as and ODS table. Tables from all R threads are binded together.

return_table(df= data.frame | arrowTable, name= *str*, label= *str*, title= *str*)

df= *data.frame* | *table*

specifies the input pandas data frame or Arrow table.

name= *str*

specifies the table name.

label= *str*

Default ""

specifies the table label.

title= *str*

Default ""

specifies the table title.

return_result

Prints the list of arguments to the SAS log.

```
return_result(name= str, arg-1, ..., arg-n)

list= str
    specifies the name of the list.

arg-1, ..., arg-n
    Specifies the arguments to be returned. For example, arg1='123', arg2=.2,
    arg3=[0, 2].
```

R Variables

The following variables are available for use in your R sessions while using PROC GATEWAY.

Table 27.2 R Variables

Variable Name	Variable Description
gw\$num_threads	The total number of threads.
gw\$num_workers	The total number of workers.
gw\$thread_id	The current sessions thread ID (starts with 0).
gw\$worker_id	The current session worker ID (starts with 0).
gw\$job_id	The current job ID.
gw\$user_id	The current user ID.
gw\$args	The list of arguments passed by the “ ARGS Statement ”.

Usage: GATEWAY Procedure

PROC GATEWAY Compared with Other SAS Procedures

Unlike the PROC PYTHON, PROC GATEWAY does not launch an interactive Python or R session. It is designed for batch parallel processing that runs the code in all of the sessions at once and then closes it. Data is not recoverable once the session is

finished. Each of the Open Source sessions does not interact with each other, only with the CAS server.

Converting Data Types

PROC GATEWAY automatically converts certain data types when data is moved from an arrow table to a CAS table or from CAS table to an arrow table. Not all arrow data types are currently supported for CAS tables.

The following table lists the currently supported and non-supported data types.

Table 27.3 PROC GATEWAY Data Type Conversions

Supported	Arrow Data Type	CAS Data Type
Yes	BINARY	VARBINARY
Yes	BOOL	DOUBLE
Yes	DATE32	DATE
Yes	DATE64	DATETIME
Yes	DOUBLE	DOUBLE
Yes	FLOAT	DOUBLE
Yes	UINT8	INT32
Yes	UINT16	INT32
Yes	UINT32	INT64
Yes	UINT64	INT64
Yes	INT8	INT32
Yes	INT16	INT32
Yes	INT32	INT32
Yes	INT64	INT64
Yes	LARGE_BINARY	VARBINARY
Yes	LARGE_STRING	VARCHAR

Supported	Arrow Data Type	CAS Data Type
Yes	STRING	VARCHAR
Yes	TIME32	TIME
Yes	TIME64	TIME
Yes	TIMESTAMP	DATETIME
No	DECIMAL128	
No	DECIMAL256	
No	DENSE_UNION	
No	DICTIONARY	
No	DURATION	
No	EXTENSION	
No	FIXED_SIZE_BINARY	
No	FIXED_SIZE_LIST	
No	HALF_FLOAT	
No	LARGE_LIST	
No	LIST	
No	MAP	
No	NA	
No	SPARSE_UNION	
No	STRUCT	

Submitting Open Source Statements

PROC GATEWAY always requires a CAS session. PROC GATEWAY provides two methods of submitting Open Source language statements.

You can submit Open Source statements within the PROC GATEWAY invocation, between “[SUBMIT Statement](#)” and “[ENDSUBMIT Statement](#)”. The default language for PROC GATEWAY is Python. To submit R code, you must specify R for the [LANG](#) option. The following code executes a single R print statement.

```
proc gateway lang=R;
submit;
print(paste('hello world', gw$worker_id))
endsubmit;
run;
```

You can also submit Open Source statements using a path to a file containing your code using the [CODE_FILE](#) argument.

Note: The [CODE_FILE](#) argument is ignored if you use the [SUBMIT](#) and [ENDSUBMIT](#) statements.

```
PROC GATEWAY
code_file ="path/to/file/code.py";
run;
```

Selecting the Number of Sessions

You can set how many Python or R sessions are going to be run on each CAS worker with the [NTHREADS](#) option.

```
PROC GATEWAY nthreads=2;
submit;
print('Hello World from thread_id: {}'.format(thread_id))
endsubmit;
run;
```

You can also specify to run a single Python or R session by using the [SINGLE](#) option.

```
PROC GATEWAY single;
submit;
print('Hello World from thread_id: {}'.format(thread_id))
endsubmit;
run
```

Reading and Writing CAS Tables

PROC GATEWAY enables you to interact directly with CAS to read and write tables from each of the python sessions using the gateway embedded module. If you use multiple sessions, CAS will automatically distribute the data accross sessions when the client it reading, and it appends the tables together to the given table.

```
PROC GATEWAY single;
submit;
```

```

import pandas as pd
df = pd.read_csv('https://support.sas.com/documentation/onlinedoc/viya/
exampledatasets/hmeq.csv')
write_table(df, {'caslib': 'casuser', 'name': 'hmeq', 'replace': True})
df2 = read_table({'caslib': 'casuser', 'name': 'hmeq'})
print(df2.head())
endsubmit;
run;

```

For large tables, you can read and write the table in batches using the [“new_table_reader”](#) Python or [“new_table_reader”](#) R.

```

PROC GATEWAY single;
submit;
inspec = {'name': 'hmeq', 'caslib': 'casuser'}
outspec = {'name': 'hmeq_b', 'caslib': 'casuser', 'replace': True}
with new_table_writer(outspec) as w:
    with new_table_reader(inspec, batch_size = 500) as r:
        for df in r:
            lastbatchsize = df.shape[0]
            print(f'reading batch of size {lastbatchsize} and writing')
            w.write_batch(df)
w.close()
endsubmit;
run;

```

GROOVY Procedure

Overview: GROOVY Procedure	1009
What Does the GROOVY Procedure Do?	1009
Special Considerations	1010
Syntax: GROOVY Procedure	1010
PROC GROOVY Statement	1011
ADD Statement	1012
EVALUATE Statement	1013
EXECUTE Statement	1014
SUBMIT Statement	1015
ENDSUBMIT Statement	1016
CLEAR Statement	1017
Usage: GROOVY Procedure	1018
Special Variables	1018
Examples: GROOVY Procedure	1020
Example 1: Define Classes	1020
Example 2: Pass a Macro Variable to PROC GROOVY	1021

Overview: GROOVY Procedure

What Does the GROOVY Procedure Do?

Groovy is a dynamic language that runs on the Java Virtual Machine (JVM). PROC GROOVY enables SAS code to execute Groovy code on the JVM.

PROC GROOVY can run Groovy statements that are written as part of your SAS code, and it can run statements that are in files that you specify with PROC GROOVY commands. It can parse Groovy statements into Groovy Class objects,

and it can run these objects or make them available to other PROC GROOVY statements or Java DATA Step Objects. You can also use PROC GROOVY to update your CLASSPATH environment variable with additional CLASSPATH strings or filerefs to JAR files.

CAUTION

Do not run unsafe code. Running unsafe code may expose your environment to data loss, data corruption, security breaches, or system instability. It is your responsibility to ensure that all code you incorporate is secure, tested, and compliant with your organization's policies and applicable law.

Special Considerations

Groovy code that is submitted with PROC GROOVY runs as the process owner, and has the same access to resources (file system, network, and so on) as any process owner. Groovy code access to resources can cause problems when SAS code is running inside multiuser servers like the Stored Process Server. To give administrators some control over this functionality, PROC GROOVY runs only if the NOXCMD option is turned off. All SAS servers are shipped with the NOXCMD option turned on.

The use of a percent character (%) in the first byte of the text that is written by Java to the SAS log is reserved by SAS. If you need to write a percent character in the first byte of a Java text line, then you must immediately follow it with another percent character (%%).

PROC GROOVY does not support the THREADS | NOTTHREADS SAS system option. However, Groovy code that you submit with PROC GROOVY can use threaded processing in the JVM.

Syntax: GROOVY Procedure

Restriction: PROC GROOVY does not run if the NOXCMD option is turned on.

Interaction: When a SAS server is in a locked-down state, PROC GROOVY will not execute. For more information, see [“LOCKDOWN” in SAS Programmer's Guide: Essentials](#).

PROC GROOVY <CLASSPATH=*string-or-fileref*> <SASJAR= | *version*= | *range*=>;

ADD CLASSPATH=*string-or-fileref*<SASJAR=<*version*=>*range*=>;

EVALUATE <(LOAD)>

"Groovy statement string" <*argument(s)*>;

EXECUTE <(LOAD)>

Groovy filename | *fileref* <*argument(s)*>;

SUBMIT <(LOAD)> <*argument(s)*>;

Groovy statement(s)
ENDSUBMIT;
CLEAR;
QUIT;

Statement	Task	Example
PROC GROOVY	Enable SAS code to run Groovy code on the JVM	Ex. 1
ADD	Append the given classpath to the current CLASSPATH environment variable	
EVALUATE	Parse the Groovy statement in the quoted string and call the Run method on the Script	
EXECUTE	Read the contents of the file that is specified as either a quoted string path or as a fileref and call the Run method on the Script	
SUBMIT	Parse the Groovy statements between the SUBMIT and ENDSUBMIT commands and call the Run method on the Script	
ENDSUBMIT	End the Groovy statements that begin with the SUBMIT command	
CLEAR	Empty the binding and unload the Groovy classloader	

PROC GROOVY Statement

Enables SAS code to run Groovy code on the JVM.

Syntax

PROC GROOVY <CLASSPATH=*string-or-fileref*> <SASJAR= | *version*= | *range*=>;

Optional Arguments

CLASSPATH=*string-or-fileref*

specifies a quoted CLASSPATH string or a fileref to a specific JAR file that is to be added to the current classpath. This path is searched after the paths that are in the user's CLASSPATH environment variable.

Alias PATH=

SASJAR=<version=> | <range=>

specifies a quoted string that identifies a JAR in the Versioned JAR Repository (VJR) that should be added to the current classpath. The VERSION and RANGE values are optional. RANGE takes precedence over VERSION, as in the following example:

```
ADD SASJAR="sas.core";
ADD SASJAR="sas.core" version="903000.9.0.20100810190000_v930";
ADD SASJAR="sas.core" range="[0,909000]";
```

Note: SAS JAR files do not have a source compatibility guarantee across versions of SAS. Future versions of this JAR can change without notice. To ensure continued functionality, contact SAS Technical Support.

Details

PROC GROOVY uses the current user's CLASSPATH environment variable as the base for building its classpath. You can use the CLASSPATH and SASJAR options to add paths to the current classpath.

When a class is loaded, the paths are searched in the following order:

- 1 CLASSPATH environment variable when process started
- 2 paths added with the ADD CLASSPATH and ADD SASJAR statements in the order in which they were executed

ADD Statement

Appends the given classpath to the current CLASSPATH environment variable.

Syntax

ADD CLASSPATH=*string-or-fileref***<SASJAR=<version=>range=>;**

Required Arguments

CLASSPATH=*string-or-fileref*

specifies a quoted CLASSPATH string or a fileref to a specific JAR file that is to be added to the current classpath. This path is searched after the paths that are in the user's CLASSPATH environment variable.

Alias PATH=

SASJAR=<version=> | <range=>

specifies a quoted string that identifies a JAR in the Versioned JAR Repository (VJR) that should be added to the current classpath. The VERSION and RANGE values are optional. RANGE takes precedence over VERSION, as in the following example:

```
ADD SASJAR="sas.core";
ADD SASJAR="sas.core" version="903000.9.0.20100810190000_v930";
ADD SASJAR="sas.core" range="[0,909000]";
```

Note: SAS JAR files do not have a source compatibility guarantee across versions of SAS. Future versions of this JAR can change without notice. To ensure continued functionality, contact SAS Technical Support.

Details

The ADD statement appends the given classpath to the current CLASSPATH environment variable.

You must specify at least one CLASSPATH or one SASJAR. You can specify multiple CLASSPATHs or SASJARs.

EVALUATE Statement

Parses the Groovy statement that is provided in the quoted string into a groovy.lang.Script object and calls the Run method on the Script.

Alias: EVAL

Syntax

EVALUATE <(LOAD)>
"Groovy statement string" <argument(s)>;

Required Argument

Groovy statement string

specifies a Groovy statement string that is to be parsed by the EVALUATE statement.

Optional Arguments

(LOAD)

parses the Groovy statement into a groovy.lang.Script object, but does not run it. The arguments are aliases for each other.

Aliases (PARSEONLY)

(NORUN)

argument(s)

specifies arguments that are passed to the code that is being evaluated.

Details

The EVALUATE statement parses the Groovy statement that is provided in the quoted string into a `groovy.lang.Script` object and calls the `Run` method on the `Script`. If one of the `LOAD`, `PARSEONLY`, or `NORUN` options is present, then this statement parses the Groovy statement into a `Class` object but does not run it. Any classes that are defined by the Groovy code are then available for use by `PROC GROOVY` statements or by Java `DATA Step` Objects.

EVAL is an alias for the EVALUATE statement.

EXECUTE Statement

Reads the contents of the file that is specified as either a quoted string path or as a fileref.

Alias: EXEC

Syntax

EXECUTE <(LOAD)>

Groovy filename | *fileref* <*argument(s)*>;

Required Arguments

Groovy filename

specifies the name of the Groovy file that is to be parsed by the EXECUTE statement.

fileref

specifies the name of a fileref that is to be parsed by the EXECUTE statement.

Optional Arguments

(LOAD)

parses the Groovy statement in the specified Groovy file or fileref into a `groovy.lang.Script` object, but does not run it. The arguments are aliases for each other.

Aliases (PARSEONLY)

(NORUN)

argument(s)

specifies arguments that are passed to the code that is being executed.

Details

The EXECUTE statement reads the contents of the file that is specified as either a quoted string path or as a fileref. The contents are then parsed into a `groovy.lang.Script` object, and the `Run` method is called on the `Script`. If one of the `LOAD`, `PARSEONLY`, or `NORUN` options is present, then this statement parses the file contents into a `Class` object but does not run it. Any classes that are defined by the Groovy code are then available for use by `PROC GROOVY` statements or by Java `DATA Step Objects`.

EXEC is an alias for the EXECUTE statement.

Note: If you used an EXEC `PARSEONLY` statement to compile a file into a `Class`, then you must submit a `CLASS` statement so that changes to that file are honored by future EXEC `PARSEONLY` commands. If you do not submit the `CLEAR` statement, then any changes that you made to the file after you issued the EXEC `PARSEONLY` statement are not included by subsequent submissions of the EXEC `PARSEONLY` statement. You can use the `GroovyScriptEngine Class` if you need to use reloadable scripts.

SUBMIT Statement

Parses the Groovy statements that are between the `SUBMIT` and `ENDSUBMIT` commands into a `groovy.lang.Script` object and calls the `Run` method on the `Script`.

Syntax

```
SUBMIT <(LOAD)> <argument(s)>;
Groovy statement(s)
ENDSUBMIT;
```

Required Argument

Groovy statement(s)

specifies Groovy statements that are to be parsed by the `SUBMIT` statement into a `groovy.lang.Script` object.

Optional Arguments

(LOAD)

parses the Groovy statements into a `groovy.lang.Script` object, but does not run it. The arguments are aliases for each other.

Aliases (PARSEONLY)

(NORUN)

argument(s)

specifies arguments that are passed to the code that is being submitted.

Details

The SUBMIT statement parses the Groovy statements that are between the SUBMIT and ENDSUBMIT commands into a `groovy.lang.Script` object and calls the Run method on the Script. If one of the LOAD, PARSEONLY, or NORUN options is present, then this statement parses the Groovy statements into a Class object but does not run it. Any classes that are defined by the Groovy code are then available for use by PROC GROOVY statements or by Java DATA Step Objects.

Note:

- The ENDSUBMIT statement must be on a line by itself and preceded by only blank space.
 - Macro substitution is disabled between the SUBMIT and ENDSUBMIT commands.
 - PROC GROOVY with multi-line submit commands cannot be used inside a macro.
-

ENDSUBMIT Statement

Ends the Groovy statements that begin with the SUBMIT command.

Syntax

ENDSUBMIT;

Details

Ends the Groovy statements that begin with the SUBMIT statement.

Note: The ENDSUBMIT statement must be on a line by itself and preceded by only blank space.

CLEAR Statement

Empties the binding and unloads the Groovy classloader.

Alias: RESET

Syntax

CLEAR;

Details

The CLEAR statement empties the binding and unloads the Groovy classloader. When this statement is executed, any variables that are saved in the binding are rendered unavailable. Any classes that are loaded into the Groovy classloader are also rendered unavailable.

RESET is an alias for the CLEAR statement.

Note: Neither the CLEAR statement nor the RESET statement resets the System.Properties collection or the CLASSPATH.

Usage: GROOVY Procedure

Special Variables

PROC GROOVY has four special variables: BINDING, ARGS, EXPORTS, and SHELL. It makes these variables available to any Groovy code that it is running.

BINDING

The BINDING special variable is used to share the state of objects between executions of PROC GROOVY. It is populated by any variables that are created without scope or that are explicitly stored in the binding. BINDING also holds all of the other special variables that are discussed in this section. The binding can be cleared with the CLEAR statement.

Note: The BINDING special variable is available to any Groovy code that PROC GROOVY is running.

```
proc groovy;
    eval "a = 42";
    eval "binding.b = 84";
    eval "binding.setProperty( 'c', 168 )";
quit;
proc groovy;
    eval "println \"----> ${binding.getProperty('a')}\"";
    eval "println \"----> ${b}\"";
    eval "println \"----> ${binding.c}\"";
quit;
```

ARGS

Arguments are passed to Groovy code in the ARGS special variable in the binding.

Note: The ARGS special variable is available to any Groovy code that PROC GROOVY is running.

```
proc groovy;
    eval "args.each{ println \"----> ev ${it}\" }" "arg1" "arg2" "arg3";
```



```

exec "args.groovy" "arg1" "arg2" "arg3";

submit "arg1" "arg2" "arg3";
  args.each{
    println "----> su ${it}"
  }
endsubmit;
quit;

```

EXPORTS

The EXPORTS special variable contains a map in the binding. Adding a key or value pair to this map will create a SAS macro variable when PROC GROOVY ends. Groovy is case sensitive, but macros are not. If two keys exist in the map that differ only by their case, then the one that is exported into a SAS macro is not determined. You can also replace the EXPORTS variable in the binding with any object that inherits from `java.util.Map`. If you replace the variable, all of the key or value pairs in that object will be exported.

Note: The EXPORTS special variable is available to any Groovy code that PROC GROOVY is running.

```

proc groovy;
  eval "exports.fname = \"first name\"";
  eval "binding.exports.lname = \"last name\"";
  eval "exports.put('state', 'NC')";
quit;

data _NULL_;
  put "----> &fname &lname: &state";
run;

proc groovy;
  submit;
    exports = [fname:"first name", lname: "last name", state: "NC"]
  endsubmit;
quit;

data _NULL_;
  put "----> &fname &lname: &state";
run;

```

SHELL

The SHELL special variable in the binding is set to the `groovy.lang.GroovyShell` that was used to compile the current script. You must submit a CLEAR statement before changes that were made to the execution.groovy file in this example are reflected in subsequent runs of the code.

Note: The SHELL special variable is available to any Groovy code that PROC GROOVY is running.

```
proc groovy;
    eval "shell.run(
        new File("execution.groovy"),
        [] as String[] );";
quit;
```

Note: If you need Groovy scripts that will be reloaded automatically when they are modified, then create a new instance of the GroovyScriptEngine class.

Examples: GROOVY Procedure

Example 1: Define Classes

Features:	PROC GROOVY statement option CLASSPATH SUBMIT statement option PARSEONLY SUBMIT Statement ENDSUBMIT Statement
-----------	--

The following three examples show how to use PROC GROOVY to define a class.

Program

Groovy code is run by default. If your script does not have any executable code, then an error is returned. The following example defines a class, but it does not have any executable code, and an error is returned.

```
proc groovy classpath=cp;
    submit;
        class Speaker {
            def say( word ) {
                println "----> \"${word}\"";
            }
        }
    endsubmit;
```

```
quit;
```

Program

The following example shows how to define a class that can be run by including a main method.

```
proc groovy classpath=cp;
  submit;
    class Speaker {
      def Speaker() {
        println "----> ctor"
      }
      def main( args ) {
        println "----> main"
      }
    }
  endsubmit;
quit;
```

Program

The following example shows how to use the PARSEONLY option to avoid a run call. You can then use the new class in another execution of PROC GROOVY.

```
proc groovy classpath=cp;
  submit parseonly;
    class Speaker {
      def say( word ) {
        println "----> \"${word}\""
      }
    }
  endsubmit;
quit;

proc groovy classpath=cp;
  eval "s = new Speaker(); s.say( \"Hi\" )";
quit;
```

Example 2: Pass a Macro Variable to PROC GROOVY

Features:	SUBMIT Statement
	ENDSUBMIT Statement

The following example shows how to define a macro variable and pass it to PROC GROOVY.

```
%let _inzip = C:/path/example.zip;
proc groovy;
  submit "&_inzip.";
    def zipFile = new java.util.zip.ZipFile(new File(args[0]))
    zipFile.entries().each {
      println zipFile.getInputStream(it).text
    }
  endsubmit;
quit;
```

HADOOP Procedure

Overview: HADOOP Procedure	1023
What Does the HADOOP Procedure Do?	1023
Syntax: HADOOP Procedure	1024
PROC HADOOP Statement	1025
HDFS Statement	1026
MAPREDUCE Statement	1030
PIG Statement	1033
PROPERTIES Statement	1035
Usage: HADOOP Procedure	1036
Submitting Hadoop Distributed File System Commands	1036
Submitting MapReduce Programs	1036
Submitting Pig Language Code	1036
Submitting Configuration Properties	1037
Configuring Java Access or WebHDFS Access to Hadoop	1037
Semaphore Requirements When Using Hadoop	1037
Examples: HADOOP Procedure	1038
Example 1: Submitting HDFS Commands	1038
Example 2: Submitting HDFS Commands with Wildcard Characters	1039
Example 3: Submitting a MapReduce Program	1041
Example 4: Submitting Pig Language Code	1043

Overview: HADOOP Procedure

What Does the HADOOP Procedure Do?

The HADOOP procedure enables SAS to interact with Hadoop data by running Apache Hadoop code. Apache Hadoop is an open-source framework, written in

Java, that provides distributed data storage and processing of large amounts of data.

PROC HADOOP interfaces with the Hadoop JobTracker. This is the service within Hadoop that controls tasks to specific nodes in the cluster. PROC HADOOP enables you to submit the following:

- Hadoop Distributed File System (HDFS) commands
- MapReduce programs (deprecated)
- Pig language code (deprecated)

Syntax: HADOOP Procedure

- Restrictions:** This procedure is not supported on the CAS server.
- When SAS is in a locked-down state, PROC HADOOP is not available. Your server administrator can re-enable this procedure so that it is accessible in the lockdown state. When the FILENAME Hadoop access method is re-enabled by using the LOCKDOWN ENABLE_AMS= statement, PROC HADOOP is automatically re-enabled. For more information, see [“LOCKDOWN” in SAS Programmer's Guide: Essentials](#).
- Requirement:** Generally, configuration is required in order to connect to the Hadoop Cluster. Configuration is simplified if you access Hadoop through WebHDFS. For more information, see [“Configuring Java Access or WebHDFS Access to Hadoop” on page 1037](#).
- Tip:** The supported Hadoop distributions and versions are the same as for SAS/ACCESS Interface to Hadoop. For system requirements, see [“Requirements for SAS/ACCESS Interface to Hadoop” in System Requirements for the SAS Viya Platform](#).

PROC HADOOP <hadoop-server-options>;

HDFS <hadoop-server-options> <hdfs-command-options>;

MAPREDUCE <hadoop-server-options> <mapreduce-options>;

PIG <hadoop-server-options> <pig-code-options>;

PROPERTIES <configuration-properties>;

Statement	Task	Example
PROC HADOOP	Control access to the Hadoop server	Ex. 1, Ex. 2, Ex. 3, Ex. 4
HDFS	Submit Hadoop Distributed File System (HDFS) commands	Ex. 1, Ex. 2
MAPREDUCE	Submit MapReduce programs into a Hadoop cluster	Ex. 3

Statement	Task	Example
PIG	Submit Pig language code into a Hadoop cluster	Ex. 4

PROC HADOOP Statement

Controls access to the Hadoop server.

Examples: “Example 1: Submitting HDFS Commands” on page 1038
 “Example 2: Submitting HDFS Commands with Wildcard Characters” on page 1039
 “Example 3: Submitting a MapReduce Program” on page 1041
 “Example 4: Submitting Pig Language Code” on page 1043

Syntax

PROC HADOOP <*hadoop-server-options*>;

Summary of Optional Arguments

MAXWAIT=*wait-interval*

specifies the HTTP status response time when using WebHDFS.

PASSWORD=*'password'*

is the password for the user ID on the Hadoop server.

USERNAME=*'ID'*

is an authorized user ID on the Hadoop server.

VERBOSE

enables additional messages that are displayed on the SAS log.

WEBHDFSURL=*web-HDFS-URL*

specifies the URL to access the Hadoop distributed file system (HDFS) through a REST API.

Hadoop Server Options

These options control access to the Hadoop server and can be specified in all HADOOP procedure statements.

MAXWAIT=*wait-interval*

specifies the HTTP status response time when using WebHDFS.

Default 40000 milliseconds

Requirement The environment variable SAS_HADOOP_RESTFUL must be set to 1.

Tip If you receive a time-out message in the log, use MAXWAIT= to increase the wait period.

PASSWORD='password'

is the password for the user ID on the Hadoop server. The user ID and password are added to the set of options that are identified by CFG=.

Alias PASS=

Interaction To specify PASSWORD=, you must also specify USERNAME=.

USERNAME='ID'

is an authorized user ID on the Hadoop server. The user ID and password are added to the set of options that are identified by CFG=.

Alias USER=

VERBOSE

enables additional messages that are displayed on the SAS log. VERBOSE is a good error diagnostic tool. If you receive an error message when you invoke SAS, you can use this option to see whether you have an error in your system option specifications.

WEBHDFSURL=web-HDFS-URL

specifies the URL to access the Hadoop distributed file system (HDFS) through a REST API. For more information, see [“Configuring Java Access or WebHDFS Access to Hadoop” on page 1037](#).

Note This option was added in 2025.08.

Example webhdfsurl="https://my-gateway-host/my-gateway/my-api"

HDFS Statement

Submits Hadoop Distributed File System (HDFS) commands.

Restrictions: The HDFS statement supports only one operation per invocation. The CAT, CHMOD, and LS commands are available only when submitting HDFS commands through webHDFS. The RECURSE option and use of wildcards are permitted only when submitting HDFS commands through webHDFS.

Requirement: For configuration requirements, see [“Configuring Java Access or WebHDFS Access to Hadoop” on page 1037](#).

Examples: [“Example 1: Submitting HDFS Commands” on page 1038](#)
[“Example 2: Submitting HDFS Commands with Wildcard Characters” on page 1039](#)

Syntax

HDFS *<hadoop-server-options>* *<hdfs-command-options>*;

Summary of Optional Arguments

CAT=*'HDFS-file'* **< ONLY=*n* >** **< OUT=*'output-location'* >** **< RECURSE >** **< SHOW_FILENAME >**
 displays the contents of the specified file or files.

CHMOD=*'HDFS-file'* **PERMISSION**=**< '*>value<' >*** **< RECURSE >**
 changes file access permissions for one or more HDFS files.

COPYFROMLOCAL=*'local-file'* **OUT**=*'output-location'* **< DELETESOURCE >** **< OVERWRITE >** **< RECURSE >**
 copies the specified local file to an HDFS output location.

COPYTOLOCAL=*'HDFS-file'* **OUT**=*'output-location'* **< DELETESOURCE >** **< KEEP_CRC >** **< OVERWRITE >** **< RECURSE >**
 copies the specified HDFS file to a local output location.

DELETE=*'HDFS-file'* **< NOWARN >**
 deletes the specified HDFS file.

LS=*'HDFS-pathname'* **< OUT=*output-location* >** **< RECURSE >**
 lists the files in the specified HDFS pathname.

MKDIR=*'HDFS-pathname'*
 creates the specified HDFS pathname. Specify the complete HDFS pathname.

RENAME=*'HDFS-file'* **OUT**=*'output-location'*
 renames the specified HDFS file.

HDFS Command Options

These options support commands that interact with the HDFS. Include only one operation per HDFS statement.

CAT=*'HDFS-file'* **< ONLY=*n* >** **< OUT=*'output-location'* >** **< RECURSE >** **< SHOW_FILENAME >**

displays the contents of the specified file or files.

'HDFS-file'

specifies a pathname or a pathname and a filename. You can use wildcard characters to substitute for any other character or characters in the pathname or the filename. Use * to match one or more characters, or ? to match a single character.

ONLY=*n*

displays only the specified number of lines from the beginning of the file. For example, *only=10* displays the first ten lines of a file. This option is helpful to determine the contents of a file.

OUT='output-location'

specifies the output location for the contents, which can be an external file for your machine or a fileref that is assigned with the FILENAME statement. By default, the output location is the SAS log.

RECURSE

specifies to display the contents for all files in the specified pathname and all files that are in subdirectories. RECURSE has no effect if the specified HDFS file is not a directory.

SHOW_FILENAME

includes the name of the file in the output. For example, `hdfs cat='/tmp/*.txt' show_filename only=10 recurse;` displays in the SAS log the name of the file and the first ten lines of all .txt files that are found in the /tmp directory and all of its subdirectories.

CHMOD='HDFS-file' PERMISSION=<'>value<'> < RECURSE >

changes file access permissions for one or more HDFS files.

'HDFS-file'

specifies a pathname or a pathname and a filename. You can use a wildcard character to substitute for any character or characters in the pathname or filename. Use * to match any number of characters, or ? to match a single character.

PERMISSION=value

specifies a value that represents three levels of permissions, which are owner, group, and user. All three permission levels are required. You can specify the permissions in read, write, and execute (rwx) symbolic notation or octal notation.

- For the rwx symbolic notation, use nine characters. The first set of three characters represents what the owner can do, the second set represents what a group can do, and the third set represents what a user can do. For each set of three characters, the first position must be r or - (for read), the second position must be w or - (for write), and the third position must be x or - (for execute). For example, `permission=rwxr-xr-x` specifies that the owner has Read, Write, and Execute permission, group members have Read and Execute permission, and users have Read and Execute permission.
- For octal notation, use three digits. Each digit represents the permissions for owner, group, and user. Each digit must be from 0 to 7. The octal notation represents the same numeric value as the rwx symbolic notation. That is, 4 is r, 2 is w, 1 is x, and 0 is -. For example, `permission=755;` specifies that the owner has Read, Write, and Execute permission, group members have Read and Execute permission, and users have Read and Execute permission.

RECURSE

specifies to change the access permissions to all files and directories in the specified pathname and all files and directories that are in subdirectories. RECURSE has no effect if the specified HDFS file is not a directory. For example, `hdfs chmod='/tmp' permission=755 recurse;` changes the permissions to the specified directory and all files and subdirectories within the directory.

COPYFROMLOCAL='local-file' OUT='output-location' < DELETESOURCE > < OVERWRITE > < RECURSE >

copies the specified local file to an HDFS output location.

'local-file'

specifies the complete pathname and the filename. You can use a wildcard character to substitute for any other character or characters in the pathname or the filename. Use * to match any number of characters, or ? to match a single character.

OUT='output-location'

specifies the output location for the copied file, which is a complete HDFS pathname and the filename.

DELETESOURCE

deletes the input source file after a copy command.

OVERWRITE

specifies to overwrite an existing output location.

RECURSE

specifies to copy all the files in the specified pathname and all files that are in subdirectories. RECURSE has no effect if the specified file is not a directory.

COPYTOLOCAL='HDFS-file' OUT='output-location' < DELETESOURCE > < KEEP CRC > < OVERWRITE > < RECURSE >

copies the specified HDFS file to a local output location.

'HDFS-file'

specifies the complete pathname and the filename. You can use a wildcard character to substitute for any other character or characters in the pathname or the filename. Use * to match any number of characters, or ? to match a single character.

OUT='output-location'

specifies the output location for the copied file, which is an external file for your machine.

DELETESOURCE

deletes the input source file after a copy command.

KEEP CRC

saves the Cyclic Redundancy Check (CRC) file after the copy command to a local output location. The CRC file is saved to the same location that is specified in the OUT= option. The CRC file is used to ensure the correctness of the file being copied. By default, the CRC file is deleted.

OVERWRITE

specifies to overwrite an existing output location.

RECURSE

specifies to copy all the files in the specified pathname and all files that are in subdirectories. RECURSE has no effect if the specified HDFS file is not a directory.

DELETE='HDFS-file' < NOWARN >

deletes the specified HDFS file.

HDFS-file

specifies a pathname or a pathname and a filename. If you include the filename, then only that file is deleted. If you do not include a filename, then all the files in the specified pathname and all the files that are in subdirectories are deleted. You can use a wildcard character to substitute for any other character or characters in the pathname or the filename. Use * to match any number of characters, or ? to match a single character.

NOWARN

suppresses the warning message when there is an attempt to delete a file that does not exist.

LS='HDFS-pathname' < OUT=output-location >< RECURSE >

lists the files in the specified HDFS pathname. The output for each file consists of its permissions, User ID, User ID group, file size, creation date, creation time, and the filename.

HDFS-pathname

specifies a pathname. You can use a wildcard character to substitute for any character or characters in the pathname. Use * to match any number of characters, or ? to match a single character.

OUT=output-location

specifies the output location for the list of files, which can be an external file for your machine or a fileref that is assigned with the FILENAME statement. By default, the output location is the SAS log.

RECURSE

specifies to list the files in the specified pathname and all files that are in subdirectories. RECURSE has no effect if the specified file is not a directory.

Default The default output location is the SAS log.

MKDIR='HDFS-pathname'

creates the specified HDFS pathname. Specify the complete HDFS pathname.

RENAME='HDFS-file' OUT='output-location'

renames the specified HDFS file.

'HDFS-file'

specifies the pathname and the filename to rename.

OUT='output-location'

specifies the new HDFS pathname and filename.

MAPREDUCE Statement

Submits MapReduce programs into a Hadoop cluster.

Requirement: To submit MapReduce programs to a Hadoop server, the Hadoop configuration file must include the properties to run MapReduce (MR1) or MapReduce 2 (MR2) and YARN.

- Interaction: To submit MapReduce programs using the Java API, the Hadoop distribution JAR files must be copied to a physical location that is accessible to the SAS client. The SAS environment variable SAS_HADOOP_JAR_PATH must be set to the location of the Hadoop JAR files. For more information, see [“Configuring Java Access or WebHDFS Access to Hadoop” on page 1037](#).
- Note: Beginning in 2025.06, the MAPREDUCE statement is deprecated. This statement is still functional, but support is likely to be removed in a future update. It is recommended that you use Spark or Hive functionality as an alternative.
- Example: [“Example 3: Submitting a MapReduce Program” on page 1041](#)

Syntax

MAPREDUCE *<hadoop-server-options>* *<mapreduce-options>*;

Summary of Optional Arguments

COMBINE='class-name'

specifies the name of the combiner class in dot notation.

DELETERESULTS

specifies to delete the output directory, if it exists, before starting the MapReduce job.

GROUPCOMPARE='class-name'

specifies the name of the grouping comparator (GroupComparator) class in dot notation.

INPUT='HDFS-pathname'

specifies the HDFS pathname to the MapReduce input file.

INPUTFORMAT='class-name'

specifies the name of the input format class in dot notation.

JAR='external-file(s)'

specifies the locations of the JAR files that contain the MapReduce program and named classes.

MAP='class-name'

specifies the name of the map class in dot notation.

OUTPUT='HDFS-pathname'

when connecting to the Hadoop server, specifies a new HDFS pathname for the MapReduce output.

OUTPUTFORMAT='class-name'

specifies the name of the output format class in dot notation.

OUTPUTKEY='class-name'

specifies the name of the output key class in dot notation.

OUTPUTVALUE='class-name'

is the name of the output value class in dot notation.

PARTITIONER='class-name'

specifies the name of the partitioner class in dot notation.

REDUCE='class-name'

specifies the name of the reducer class in dot notation.

REDUCETASKS=integer

specifies the number of reduce tasks.

SORTCOMPARE='class-name'

specifies the name of the sort comparator class in dot notation.

MapReduce Options

COMBINE='class-name'

specifies the name of the combiner class in dot notation.

DELETERESULTS

specifies to delete the output directory, if it exists, before starting the MapReduce job.

GROUPCOMPARE='class-name'

specifies the name of the grouping comparator (GroupComparator) class in dot notation.

INPUT='HDFS-pathname'

specifies the HDFS pathname to the MapReduce input file.

INPUTFORMAT='class-name'

specifies the name of the input format class in dot notation.

JAR='external-file(s)'

specifies the locations of the JAR files that contain the MapReduce program and named classes. Include the complete pathname and the filename.

Requirement Enclose each location in single or double quotation marks.**MAP='class-name'**

specifies the name of the map class in dot notation. A map class contains elements that are formed by the combination of a key value and a mapped value.

OUTPUT='HDFS-pathname'

when connecting to the Hadoop server, specifies a new HDFS pathname for the MapReduce output.

Requirements You must specify the MapReduce output location.

Enclose the physical name in single or double quotation marks.

OUTPUTFORMAT='class-name'

specifies the name of the output format class in dot notation.

OUTPUTKEY='class-name'

specifies the name of the output key class in dot notation.

OUTPUTVALUE='class-name'

is the name of the output value class in dot notation.

PARTITIONER='class-name'

specifies the name of the partitioner class in dot notation. A partitioner class controls the partitioning of the keys of the intermediate map outputs.

REDUCE='class-name'

specifies the name of the reducer class in dot notation. The reduce class reduces a set of intermediate values that share a key to a smaller set of values.

REDUCETASKS=integer

specifies the number of reduce tasks.

SORTCOMPARE='class-name'

specifies the name of the sort comparator class in dot notation.

PIG Statement

Submits Pig language code into a Hadoop cluster.

Interaction: To submit Pig language code using the Java API, the Hadoop distribution JAR files must be copied to a physical location that is accessible to the SAS client. The SAS environment variable SAS_HADOOP_JAR_PATH must be set to the location of the Hadoop JAR files. For more information, see [“Configuring Java Access or WebHDFS Access to Hadoop” on page 1037](#).

Note: Beginning in 2025.06, the PIG statement is deprecated. This statement is still functional, but support is likely to be removed in a future update. It is recommended that you use Spark or Hive functionality as an alternative.

Example: [“Example 4: Submitting Pig Language Code” on page 1043](#)

Syntax

PIG *<hadoop-server-options>* *<pig-code-options>*;

Summary of Optional Arguments

CODE=fileref | 'external-file'

specifies the source that contains the Pig language code to execute.

DELETERESULTS

when connecting to the Hadoop server through the Oozie RESTful API, specifies to delete the existing output location before starting the Oozie job.

OUTPUT='HDFS-pathname'

when connecting to the Hadoop server through the Oozie RESTful API, specifies the existing output location to delete before starting the Oozie job.

PARAMETERS=fileref | 'external-file'

specifies the source that contains parameters to be passed as arguments when the Pig code executes.

REGISTERJAR='external-file(s)'

specifies the locations of the JAR files that contain the Pig scripts to execute.

REPLACE

when connecting to Hadoop through the Oozie RESTful API, specifies to delete any existing workflow and JAR file(s) in the Oozie application before copying new files to the working directory.

WORKINGDIR='HDFS-pathname'

when connecting to Hadoop through the Oozie RESTful API, specifies the HDFS pathname for the Oozie workflow application directory.

Pig Code Options

CODE=*fileref* | '*external-file*'

specifies the source that contains the Pig language code to execute.

fileref

is a SAS fileref that is assigned to the source file. To assign a fileref, use the FILENAME statement.

'*external-file*'

is the physical location of the source file. Specify the complete pathname and the filename.

Requirement Enclose the physical name in single or double quotation marks.

DELETERESULTS

when connecting to the Hadoop server through the Oozie RESTful API, specifies to delete the existing output location before starting the Oozie job.

Interaction Use the DELETERESULTS option with the OUTPUT= option.

OUTPUT='HDFS-pathname'

when connecting to the Hadoop server through the Oozie RESTful API, specifies the existing output location to delete before starting the Oozie job.

Requirement Enclose the physical name in single or double quotation marks.

Interaction Use the OUTPUT= option with the DELETERESULTS option.

PARAMETERS=*fileref* | '*external-file*'

specifies the source that contains parameters to be passed as arguments when the Pig code executes.

fileref

is a SAS fileref that is assigned to the source file. To assign a fileref, use the FILENAME statement.

'*external-file*'

is the physical location of the source file. Specify the complete pathname and the filename.

Requirement Enclose the physical name in single or double quotation marks.

REGISTERJAR='external-file(s)'

specifies the locations of the JAR files that contain the Pig scripts to execute. Specify the complete pathname and the filename.

Requirement Enclose each location in single or double quotation marks.

REPLACE

when connecting to Hadoop through the Oozie RESTful API, specifies to delete any existing workflow and JAR file(s) in the Oozie application before copying new files to the working directory.

WORKINGDIR='HDFS-pathname'

when connecting to Hadoop through the Oozie RESTful API, specifies the HDFS pathname for the Oozie workflow application directory.

Requirements When connecting to Hadoop through the Oozie RESTFUL API, this argument is required.

Enclose the *HDFS-pathname* name in single or double quotation marks.

PROPERTIES Statement

Submits configuration properties to the Hadoop server.

Alias: PROP

Syntax

PROPERTIES '*configuration-property-1*' < '*configuration-property-2*' > ...;

Required Argument

configuration-property

specifies any property that can be specified in a Hadoop configuration file.

Requirement Enclose each property in single or double quotation marks, and enclose each value in single or double quotation marks. Here is an example: prop

```
'mapred.job.tracker'='xxx.us.company.com:8021'
```

```
'fs.default.name'='hdfs://xxx.us.company.com:8020';
```

Usage: HADOOP Procedure

Submitting Hadoop Distributed File System Commands

The Hadoop Distributed File System (HDFS) is a distributed, scalable, and portable file system for the Hadoop framework. HDFS is designed to hold large amounts of data that is distributed across a network and to provide access to the data by many clients.

The PROC HADOOP HDFS statement submits HDFS commands to the Hadoop server. HDFS commands are like the Hadoop shell commands that interact with HDFS and manipulate files. For the list of HDFS commands, see [“HDFS Statement” on page 1026](#).

Submitting MapReduce Programs

MapReduce is a parallel processing framework that enables developers to write programs to process vast amounts of data. There are two types of key functions in the MapReduce framework:

- map function that separates the data to be processed into independent chunks
- reduce function that performs analysis on that data

The PROC HADOOP MAPREDUCE statement submits a MapReduce program into a Hadoop cluster. For more information, see [“MAPREDUCE Statement” on page 1030](#).

Note: Beginning in 2025.06, the MAPREDUCE statement is deprecated. This statement is still functional, but support is likely to be removed in a future update. It is recommended that you use Spark or Hive functionality as an alternative.

Submitting Pig Language Code

The Apache Pig language is a high-level programming language that creates MapReduce programs that are used with Hadoop.

The PROC HADOOP PIG statement submits Pig language code into a Hadoop cluster. For more information, see [“PIG Statement” on page 1033](#).

Note: Beginning in 2025.06, the PIG statement is deprecated. This statement is still functional, but support is likely to be removed in a future update. It is recommended that you use Spark or Hive functionality as an alternative.

Submitting Configuration Properties

Rather than specifying a Hadoop configuration file, you can submit configuration properties with the PROC HADOOP PROPERTIES statement. For more information, see [“PROPERTIES Statement” on page 1035](#).

Configuring Java Access or WebHDFS Access to Hadoop

Configuration for PROC HADOOP is the same as the configuration for the SAS/ACCESS Interface to Hadoop. For more information, see [“Configuring the Java HDFS Client” in SAS/ACCESS for Relational Databases: Reference](#).

Starting in 2025.08, when using WebHDFS, you can specify the REST URL in the WEBHDFSURL= option. When you use the WEBHDFSURL= option, you are not required to run the Hadoop tracer script to copy the Hadoop JAR and configuration files to the SAS client machine. Prior to 2025.08, those steps were required. In addition, you are no longer required to specify environment variables SAS_HADOOP_CONFIG_PATH or SAS_HADOOP_RESTFUL=1. The details are similar to SAS/ACCESS Interface to Hadoop. For more information, see [“Using WebHDFS” in SAS/ACCESS for Relational Databases: Reference](#).

If you have Apache Knox Gateway security enabled, specify the Knox Gateway endpoint in the WEBHDFSURL= option. You are no longer required to specify environment variable KNOX_GATEWAY_URL. For more information, see [“Using WebHDFS with Apache Knox Gateway Security” in SAS/ACCESS for Relational Databases: Reference](#).

Semaphore Requirements When Using Hadoop

Each time you query Hadoop, SAS requests a set of semaphore arrays from the operating system. (These arrays are sometimes referred to as semaphore sets.) SAS releases the semaphore arrays back to the operating system after the connection is closed or the query is complete.

SAS recommends that you configure the semaphore array limit on the Hadoop server to be at least 256. A better number is 512.

Note: The semaphore limit on the "maximum number of arrays" is distinct from the semaphore limit on the "maximum number of semaphores system wide". The Linux `ipcs -sl` command shows the typical default semaphore-related limits set on the server:

```
----- Semaphore Limits -----
max number of arrays = 512
max semaphores per array = 250
max semaphores system wide = 64000
max ops per semop call = 32
semaphore max value = 32767
```

Examples: HADOOP Procedure

Example 1: Submitting HDFS Commands

Features:

- SAS_HADOOP_CONFIG_PATH environment variable
- SAS_HADOOP_JAR_PATH environment variable
- PROC HADOOP statement
- HDFS statement
- OPTIONS statement
- SET system option

Details

This PROC HADOOP example submits HDFS commands to a Hadoop server. The statements create a directory, delete a directory, and copy a file from HDFS to a local output location.

Program

```
options set=SAS_HADOOP_CONFIG_PATH="configuration-files-pathname";
options set=SAS_HADOOP_JAR_PATH="JAR-files-pathname";
```

```
proc hadoop username='sasabc' password='sasabc' verbose;
    hdfs mkdir='/user/sasabc/new_directory';
    hdfs delete='/user/sasabc/temp2_directory';
    hdfs copytolocal='/user/sasabc/testdata.txt'
        out='C:\Users\sasabc\Hadoop\testdata.txt' overwrite;
run;
```

Program Description

Define the SAS_HADOOP_CONFIG_PATH environment variable and the SAS_HADOOP_JAR_PATH environment variable. The OPTIONS statements include the SET system option to define the environment variables. The environment variables set the location of the Hadoop cluster configuration files and the Hadoop JAR files so that the required files are available to the SAS session.

```
options set=SAS_HADOOP_CONFIG_PATH="configuration-files-pathname";
options set=SAS_HADOOP_JAR_PATH="JAR-files-pathname";
```

Execute the PROC HADOOP statement. The PROC HADOOP statement controls access to the Hadoop server by identifying the user ID and password on the Hadoop server. The statement specifies the VERBOSE option, which enables additional messages to be written to the SAS log.

```
proc hadoop username='sasabc' password='sasabc' verbose;
```

Create an HDFS pathname. The first HDFS statement specifies the MKDIR= option to create an HDFS pathname.

```
hdfs mkdir='/user/sasabc/new_directory';
```

Delete an HDFS file. The second HDFS statement specifies the DELETE= option to delete an HDFS file.

```
hdfs delete='/user/sasabc/temp2_directory';
```

Copy an HDFS file. The third HDFS statement specifies the COPYTOLOCAL= option to specify the HDFS file to copy, the OUT= option to specify the output location on the local machine, and the OVERWRITE option to specify that if the output location exists, write over it.

```
hdfs copytolocal='/user/sasabc/testdata.txt'
    out='C:\Users\sasabc\Hadoop\testdata.txt' overwrite;
run;
```

Example 2: Submitting HDFS Commands with Wildcard Characters

Features:

- SAS_HADOOP_CONFIG_PATH environment variable
- SAS_HADOOP_JAR_PATH environment variable
- SAS_HADOOP_RESTFUL environment variable

PROC HADOOP statement
HDFS statement
Wildcard characters
OPTIONS statement
SET system option

Details

This PROC HADOOP example submits HDFS commands to a Hadoop server. The statements display the contents of the specified files, change the permissions for one HDFS file, and list the files in a specified HDFS pathname.

Program

```
options set=SAS_HADOOP_CONFIG_PATH="configuration-files-pathname";
options set=SAS_HADOOP_JAR_PATH="JAR-files-pathname";
options set=SAS_HADOOP_RESTFUL 1;

proc hadoop username='sasabc' password='sasabc' verbose;
    hdfs cat='/user/sasabc/*';
    hdfs chmod='/user/sasabc/' permission=rwxr-xr-x;
    hdfs ls='/user/sasabc/*';
run;
```

Program Description

Define the SAS_HADOOP_CONFIG_PATH environment variable, the SAS_HADOOP_JAR_PATH environment variable, and the SAS_HADOOP_RESTFUL environment variable. The OPTIONS statements include the SET system option to define the environment variables. The first two environment variables set the location of the Hadoop cluster configuration files and the Hadoop JAR files so that the required files are available to the SAS session. The SAS_HADOOP_RESTFUL environment variable specifies to connect to the Hadoop server by using the WebHDFS REST API.

```
options set=SAS_HADOOP_CONFIG_PATH="configuration-files-pathname";
options set=SAS_HADOOP_JAR_PATH="JAR-files-pathname";
options set=SAS_HADOOP_RESTFUL 1;
```

Execute the PROC HADOOP statement. The PROC HADOOP statement controls access to the Hadoop server by identifying the user ID and password on the Hadoop server. The statement specifies the VERBOSE option, which enables additional messages to be written to the SAS log.

```
proc hadoop username='sasabc' password='sasabc' verbose;
```

Display the contents of HDFS files. The first HDFS statement specifies the CAT= option to display the contents of HDFS files. The wildcard character * specifies to match one or more characters. All files that are contained in the directory /user/sasabc/ are displayed in the SAS log.

```
hdfs cat='/user/sasabc/*';
```

Change the file access permissions. The second HDFS statement specifies the CHMOD= option to change the file access permissions for the specified HDFS pathname. The file access permissions provide the owner with Read, Write, and Execute permission, group members with Read and Execute permission, and users with Read and Execute permission.

```
hdfs chmod='/user/sasabc/' permission=rwxr-xr-x;
```

List the files in an HDFS pathname. The third HDFS statement specifies the LS= option to list the files in the specified HDFS pathname to the SAS log. The wildcard character * specifies to match one or more characters. All files that are contained in the directory /user/sasabc/ are displayed in the SAS log. The output for each file consists of its permissions, User ID, User ID group, file size, creation date, creation time, and the filename.

```
hdfs ls='/user/sasabc/*';
run;
```

Example 3: Submitting a MapReduce Program

Features:

- SAS_HADOOP_CONFIG_PATH environment variable
- SAS_HADOOP_JAR_PATH environment variable
- PROC HADOOP statement
- MAPREDUCE statement
- FILENAME statement

Details

This PROC HADOOP example submits a MapReduce program to a Hadoop server. This code runs the MapReduce job by using the Java API.

Note: Beginning in 2025.06, the MAPREDUCE statement is deprecated. This statement is still functional, but support is likely to be removed in a future update. It is recommended that you use Spark or Hive functionality as an alternative.

The example uses the Hadoop MapReduce application WordCount that reads a text input file, breaks each line into words, counts the words, and then writes the word counts to the output text file.

Program

```
options set=SAS_HADOOP_CONFIG_PATH="configuration-files-pathname";
options set=SAS_HADOOP_JAR_PATH="JAR-files-pathname";

proc hadoop username='sasabc' password='sasabc' verbose;
    mapreduce input='/user/sasabc/architectdoc.txt'
        output='/user/sasabc/outputtest'
        jar='pathname/WordCount.jar'
        outputkey='org.apache.hadoop.io.Text'
        outputvalue='org.apache.hadoop.io.IntWritable'
        reduce='org.apache.hadoop.examples.WordCount$IntSumReducer'
        combine='org.apache.hadoop.examples.WordCount$IntSumReducer'
        map='org.apache.hadoop.examples.WordCount$TokenizerMapper';
run;
```

Program Description

Define the SAS_HADOOP_CONFIG_PATH environment variable and the SAS_HADOOP_JAR_PATH environment variable. The OPTIONS statements include the SET system option to define the environment variables. The environment variables set the location of the Hadoop cluster configuration files and the Hadoop JAR files so that the required files are available to the SAS session.

```
options set=SAS_HADOOP_CONFIG_PATH="configuration-files-pathname";
options set=SAS_HADOOP_JAR_PATH="JAR-files-pathname";
```

Execute the PROC HADOOP statement. The PROC HADOOP statement controls access to the Hadoop server by identifying the user ID and password on the Hadoop server. The statement specifies the VERBOSE option, which enables additional messages to be written to the SAS log.

```
proc hadoop username='sasabc' password='sasabc' verbose;
```

Submit a MapReduce program. The MAPREDUCE statement includes several options. INPUT= specifies the HDFS pathname and the filename of the input Hadoop file named ArchitectDoc.txt.

```
mapreduce input='/user/sasabc/architectdoc.txt'
```

Create an HDFS pathname. OUTPUT= creates the HDFS pathname for the program output location named OutputTest.

```
output='/user/sasabc/outputtest'
```

Specify the JAR file. JAR= specifies the location of the JAR file that contains the MapReduce program named WordCount.jar.

```
jar='pathname/WordCount.jar'
```

Specify an output key class. OUTPUTKEY= specifies the name of the output key class org.apache.hadoop.io.Text.


```
outputkey='org.apache.hadoop.io.Text'
```

Specify an output value class. OUTPUTVALUE= specifies the name of the output value class org.apache.hadoop.io.IntWritable.

```
outputvalue='org.apache.hadoop.io.IntWritable'
```

Specify a reducer class. REDUCE= specifies the name of the reducer class org.apache.hadoop.examples.WordCount\$IntSumReducer.

```
reduce='org.apache.hadoop.examples.WordCount$IntSumReducer'
```

Specify a combiner class. COMBINE= specifies the name of the combiner class org.apache.hadoop.examples.WordCount\$IntSumReducer.

```
combine='org.apache.hadoop.examples.WordCount$IntSumReducer'
```

Specify a map class. MAP= specifies the name of the map class org.apache.hadoop.examples.WordCount\$TokenizerMapper.

```
map='org.apache.hadoop.examples.WordCount$TokenizerMapper';
run;
```

Example 4: Submitting Pig Language Code

Features:

- SAS_HADOOP_CONFIG_PATH environment variable
- SAS_HADOOP_JAR_PATH environment variable
- PROC HADOOP statement
- PIG statement
- FILENAME statement

Details

This PROC HADOOP example submits Pig language code into a Hadoop cluster. This code runs the Pig job by using the Java API.

Note: Beginning in 2025.06, the PIG statement is deprecated. This statement is still functional, but support is likely to be removed in a future update. It is recommended that you use Spark or Hive functionality as an alternative.

Here is the Pig language code, which is stored in a text file named sample_pig.txt.

```
A = LOAD '/user/sasabc/class.txt' USING PigStorage(' ')
AS (name,grade,age,height,weight);
B = FOREACH A GENERATE name, grade, some.custom.class.promotionDecision(grade) as promote;
store B into '/user/sasabc/pigout' USING PigStorage(',');
```

The Pig code references a data file named class.txt. The Pig code also calls a user-defined Java function named some.custom.class.promotionDecision() to

manipulate the data. The user has created that function and made it available in a JAR file.

Program

```
options set=SAS_HADOOP_CONFIG_PATH="configuration-files-pathname";
options set=SAS_HADOOP_JAR_PATH="JAR-files-pathname";

filename mycode 'pathname/sample_pig.txt';

proc hadoop username='sasabc' password='sasabc' verbose;
    pig code=mycode registerjar='pathname/gradeAnalysis.jar';
run;
```

Program Description

Define the SAS_HADOOP_CONFIG_PATH environment variable and the SAS_HADOOP_JAR_PATH environment variable. The OPTIONS statements include the SET system option to define the environment variables. The environment variables set the location of the Hadoop cluster configuration files and the Hadoop JAR files so that the required files are available to the SAS session.

```
options set=SAS_HADOOP_CONFIG_PATH="configuration-files-pathname";
options set=SAS_HADOOP_JAR_PATH="JAR-files-pathname";
```

Assign a file reference to the Pig language code. The FILENAME statement assigns the file reference MYCODE to the physical location of the file that contains the Pig language code that is named Sample_Pig.txt, which is shown above.

```
filename mycode 'pathname/sample_pig.txt';
```

Execute the PROC HADOOP statement. The PROC HADOOP statement controls access to the Hadoop server by identifying the user ID and password on the Hadoop server with the USERNAME= and PASSWORD= options. The statement specifies the VERBOSE option, which enables additional messages to be written to the SAS log.

```
proc hadoop username='sasabc' password='sasabc' verbose;
```

Execute the PIG statement. The CODE= option specifies the SAS fileref MYCODE, which is assigned to the physical location of the file that contains the Pig language code. The REGISTERJAR= option specifies the JAR file gradeAnalysis.jar, which the user has created, to provide user-defined functions.

```
    pig code=mycode registerjar='pathname/gradeAnalysis.jar';
run;
```

HTTP Procedure

Overview: HTTP Procedure	1045
What Does the HTTP Procedure Do?	1046
Syntax: HTTP Procedure	1046
PROC HTTP Statement	1047
DEBUG Statement	1056
HEADERS Statement	1058
SSLPARMS Statement	1059
Usage: HTTP Procedure	1060
Using Hypertext Transfer Protocol Secure (HTTPS)	1060
Using Authentication Other Than Basic	1061
Wire Logging	1062
Response Encodings	1062
Using URL-Encoding Data in PROC HTTP	1062
Uploading Multipart Data with the MULTI Option	1063
Using HEADERS= to Send Chunked Data	1064
PROC HTTP Response Status Reporting	1064
Macro Variables for Setting Global Values	1065
Examples: HTTP Procedure	1066
Example 1: A Simple GET Request	1066
Example 2: A Simple POST Request	1066
Example 3: Specify a Proxy in the HTTP Request	1067
Example 4: Specifying Input Data as a String	1068
Example 5: Specify a Proxy In a Macro Variable	1069
Example 6: A POST That Captures the Response Headers	1070
Example 7: A GET That Specifies HEADEROUT_OVERWRITE	1071
Example 8: A GET That Uses the HEADERS Statement	1073
Example 9: A Nonstandard Method	1074
Example 10: A PUT That Specifies EXPECT_100_CONTINUE	1074
Example 11: Create a Program That Captures Status Response Values	1076
Example 12: Use the DEBUG Statement with the LEVEL= Option	1079
Example 13: Specify Local Options for Encryption in UNIX	1082

Overview: HTTP Procedure

What Does the HTTP Procedure Do?

The HTTP procedure issues Hypertext Transfer Protocol (HTTP) requests.

PROC HTTP allows an open-ended set of methods. In addition to standard HTTP methods, PROC HTTP accepts any method that conforms to the HTTP/1.1 standard and that is recognized by the target web server. PROC HTTP also implements HTTP/1.1 features such as persistent connections, cookie caching, EXPECT_100_CONTINUE support, and it provides authentication type specification.

For web servers that support it, PROC HTTP uses connection caching and cookie caching by default. You can toggle the behavior of both types of caching and clear the caches within the procedure by specifying procedure arguments. Or you can turn cookie caching off by using a macro variable.

The authentication specification feature enables you to specify one or multiple authentication types for a request.

You can easily post form data as well as perform HTTP multipart requests. A QUERY= procedure option simplifies the process of submitting query parameters for the URL= argument. For more information, see [“Using URL-Encoding Data in PROC HTTP” on page 1062](#) and [“Uploading Multipart Data with the MULTI Option” on page 1063](#).

The procedure includes a DEBUG statement, response status macro variables, and the ability to specify a time-out period for requests. The SSLPARMS statement enables you to specify SSL options in the PROC HTTP request.

Syntax: HTTP Procedure

Restrictions:

This procedure is not supported on the CAS server.

When SAS is in a locked-down state, the HTTP procedure is enabled. A procedure can be re-enabled by using the ENABLE_AMS= option of the LOCKDOWN statement. The HTTP procedure is automatically re-enabled when the DS2 HTTP Package or FILENAME, URL access method is re-enabled. For more information, see [“LOCKDOWN” in SAS Programmer’s Guide: Essentials](#).

Note: Beginning in 2026.03, when FILENAME URL or PROC HTTP is in lockdown mode, the URL path can be controlled by a URL-allow list specified by SAS administrators.

Tip: PROC HTTP sets up macro variables with certain values after it executes each statement. These macro variables can be used inside a macro to test for HTTP errors. For more information, see [“PROC HTTP Response Status Reporting” on page 1064](#).

PROC HTTP URL=*URL-to-target* <*options*>;

DEBUG *options*;

HEADERS "HeaderName"="HeaderValue"
< "HeaderName-n"="HeaderValue-n" >;

SSLPARMS *host-specific-SSL-options*;

Statement	Task	Example
PROC HTTP	Issue HTTP requests	Ex. 1, Ex. 2, Ex. 3, Ex. 4, Ex. 5, Ex. 6, Ex. 7, Ex. 9, Ex. 10, Ex. 11
DEBUG	Display debugging messages	Ex. 12
HEADERS	Specify request headers for the HTTP request	Ex. 8

PROC HTTP Statement

Invokes a web service that issues requests.

Examples:

- “[Example 1: A Simple GET Request](#)” on page 1066
- “[Example 2: A Simple POST Request](#)” on page 1066
- “[Example 3: Specify a Proxy in the HTTP Request](#)” on page 1067
- “[Example 4: Specifying Input Data as a String](#)” on page 1068
- “[Example 5: Specify a Proxy In a Macro Variable](#)” on page 1069
- “[Example 6: A POST That Captures the Response Headers](#)” on page 1070
- “[Example 7: A GET That Specifies HEADEROUT_OVERWRITE](#)” on page 1071
- “[Example 8: A GET That Uses the HEADERS Statement](#)” on page 1073
- “[Example 9: A Nonstandard Method](#)” on page 1074
- “[Example 10: A PUT That Specifies EXPECT_100_CONTINUE](#)” on page 1074
- “[Example 11: Create a Program That Captures Status Response Values](#)” on page 1076
- “[Example 12: Use the DEBUG Statement with the LEVEL= Option](#)” on page 1079

Syntax

```

PROC HTTP URL="URL-to-target</redirect/n>"
  < METHOD=<">http-method<"> >
  <authentication-type-options>
  <caching-options>
  <header-options>
  <proxy-server-connection-options>
  <web-server-authentication-options>
  < EXPECT_100_CONTINUE >
  <FOLLOWLOC | NOFOLLOWLOC>
  < IN=<fileref | "string" | FORM (arguments) | MULTI <FORM | "type"> (inputs)>>
  <MAXREDIRECTS=n>
  < OUT=fileref >
  <QUERY=("parm1"="value1" "parm2"="value2" ...)>
;

```

Summary of Optional Arguments

EXPECT_100_CONTINUE

enables a client to determine whether the target server is willing to accept the request.

FOLLOWLOC

enables write methods to automatically follow URL redirections.

IN=*fileref* | "*string*" | FORM (*arguments*) | MULTI <FORM | "*type*" > (*inputs*)
specifies the input data.

MAXREDIRECTS=*n*

specifies the maximum number of redirects that are allowed.

METHOD=<">*http-method*<">

specifies an HTTP method.

NOFOLLOWLOC

prevents the GET method from following URL redirections.

OUT=*fileref-to-response-data*

specifies a *fileref* where output is written.

QUERY=(*<NOENCODE>"parm1"="value1" "parm2"="value2"*)

provides an alternate method for submitting query parameters for the URL= argument.

TIMEOUT=*integer*

specifies the number of seconds of inactivity to wait before canceling an HTTP request.

Authenticate to Web Server

WEBPASSWORD="*basic-authentication-password*"

specifies a password for basic authentication.

`WEBUSERNAME="basic-authentication-name"`

specifies a user name for basic authentication.

Connect to Proxy Server

`PROXYHOST="proxy-host-name"`

specifies the internet host name of an HTTP proxy server.

`PROXYPASSWORD="proxy-passwd"`

specifies an HTTP proxy server password.

`PROXYPORT=proxy-port-number`

specifies an HTTP proxy server port.

`PROXYUSERNAME="proxy-user-name"`

Disable Shared Connection and Cookie Caching

`CLEAR_CACHE`

specifies to clear both the shared connection and cookie caches before the HTTP request is executed.

`CLEAR_CONN_CACHE`

specifies to clear the shared connection cache before the HTTP request is executed.

`CLEAR_COOKIES`

specifies to clear the shared cookie cache before the HTTP request is executed.

`NO_CONN_CACHE`

disables connection caching for this procedure execution.

`NO_COOKIES`

specifies cached cookies will not be used for this procedure execution.

Specify Authentication Type

`AUTH_ANY`

specifies that any type of authentication can be used to authenticate to the connected server.

`AUTH_BASIC`

specifies to use user identity authentication to authenticate to the connected server.

`AUTH_NEGOTIATE`

specifies to use Kerberos or some other type of HTTP authentication to authenticate to the connected server.

`AUTH_NONE`

specifies not to use basic authentication or to negotiate authentication.

`OAuth_BEARER=token`

sends an OAuth access token along with the HTTP call.

`PROXY_AUTH_BASIC`

specifies to perform user identity authentication through a proxy server.

`PROXY_AUTH_NEGOTIATE`

specifies to perform Kerberos or some other type of HTTP authentication through a proxy server.

Specify HTTP Headers

`CT="content-type"`

specifies the HTTP content-type to be set in the request headers.

HEADERIN=*fileref-to-request-header-file*

specifies a fileref to a text file that contains one line per request header in the format *key:value*.

HEADEROUT_OVERWRITE

causes the response header to record only the last header block sent by the web server when a redirect occurs.

HEADEROUT=*fileref-to-response-header-file*

specifies a fileref to a text file to which the response headers are written in the format *key:value*.

Required Argument

URL="URL-to-target"

specifies a fully qualified URL path that identifies the endpoint for the HTTP request.

Note The URL that is passed to PROC HTTP is assumed to be URL encoded. To ensure correct encoding, use an appropriate connection class for the target web server. For example, use the AWSV4Signer class for Amazon Web Services. Or, encode reserved characters as described in [RFC3986](#).

Tip You do not have to specify the HTTP protocol in the URL. If you set just the path (for example, "httpbin.org"), the actual URL used is `http://httpbin.org`.

Optional Arguments

AUTH_ANY

When a user name and password are supplied, they are used to authenticate the connected server. Otherwise, any other form of authentication that is available is used. Specifying AUTH_ANY is equivalent to specifying AUTH_NEGOTIATE, and AUTH_BASIC on the procedure statement.

Default This is the default authentication type if an authentication type is not specified.

Tip Since there is a chance of more than one trip to the HTTP server, specify EXPECT_100_CONTINUE to prevent data from being uploaded multiple times.

AUTH_BASIC

specifies to use user identity authentication to authenticate the connected server. The user name and password are supplied with the WEBUSERNAME and WEBPASSWORD arguments.

AUTH_NEGOTIATE

specifies to use Kerberos or some other type of HTTP authentication to authenticate to the connected server. As long as your current user identity has permissions, authentication is established.

AUTH_NONE

specifies not to use basic authentication or to negotiate authentication.

The OAUTH_BEARER= procedure option can be used with NO_AUTH.

CLEAR_CACHE

specifies to clear both the shared connection and cookie caches before the HTTP request is executed.

CLEAR_CONN_CACHE

specifies to clear the shared connection cache before the HTTP request is executed.

CLEAR_COOKIES

specifies to clear the shared cookie cache before the HTTP request is executed.

CT="content-type"

used in conjunction with the HEADERIN= argument, specifies the HTTP content-type to be set in the request headers. The content-type describes the data contained in the body fully enough that the receiving user agent can present the data to the user.

Examples of content-type specifications are:

```
CT="Text/HTML; charset=ISO-8859-4"
```

```
CT="Text/plain; charset=us-ascii"
```

```
CT="Application/x-www-form-urlencoded"
```

Note This option is supported for compatibility with SAS 9.4 PROC HTTP programs. Use the [“HEADERS Statement” on page 1058](#) instead of CT= .

EXPECT_100_CONTINUE

enables a client that is sending a request message with a request body to determine whether the target server is willing to accept the request, based on the request headers. Use EXPECT_100_CONTINUE when you are sending large amounts of data and want to make sure that no unnecessary transfers of the data occur. For more information, see <http://www.w3.org/Protocols/rfc2616/rfc2616-sec8.html#sec8.2.3>.

Valid in HTTP requests that specify the IN= argument, most commonly with PUT.

Interaction This option is used in conjunction with the HEADEROUT= argument.

Example [“Example 10: A PUT That Specifies EXPECT_100_CONTINUE” on page 1074](#)

FOLLOWLOC

enables write methods to automatically follow URL redirections. By default, PROC HTTP methods that write data, like POST and PUT, terminate processing when they are redirected to an alternate location. When FOLLOWLOC is specified, PROC HTTP initially returns a 300-level response, then submits the POST or PUT again to the redirected location.

FOLLOWLOC is the default behavior for the GET method.

See [“NOFOLLOWLOC” on page 1055](#)

HEADERIN=*fileref-to-request-header-file*

specifies a fileref to a text file that contains one line per request header in the format *key:value*.

Note This option is supported for compatibility with SAS 9.4 PROC HTTP programs. Use the [“HEADERS Statement” on page 1058](#) instead of HEADERIN=

CAUTION **Do not specify both the HEADERS statement and the HEADERIN= argument.** The behavior that results from specifying both options together is not defined.

HEADEROUT=*fileref-to-response-header-file*

specifies a fileref to a text file to which the response headers are written in the format *key:value*.

Examples [“Example 6: A POST That Captures the Response Headers” on page 1070](#)

[“Example 10: A PUT That Specifies EXPECT_100_CONTINUE” on page 1074](#)

HEADEROUT_OVERWRITE

used in conjunction with the HEADEROUT= argument, causes the response header to record only the last header block sent by the web server when a redirect occurs.

Example [“Example 7: A GET That Specifies HEADEROUT_OVERWRITE” on page 1071](#)

IN=*fileref* | “*string*” | FORM (*arguments*) | MULTI <FORM | “*type*” > (*inputs*)

specifies the input data. There are multiple ways to submit input data:

fileref

specifies a fileref. The fileref is a pointer to data that exists in another location. A fileref is assigned with the FILENAME statement.

Example [“Example 2: A Simple POST Request” on page 1066](#)

“*string*”

specifies input data in a quoted string.

Example [“Example 4: Specifying Input Data as a String” on page 1068](#)

FORM (“*name1*”=“*value1*” “*name2*”=“*value2*” ...)

sends input data as a standard HTML form. The data is input as name=value pairs. The FORM option URL-encodes and delimits the pairs with an & (ampersand) as is standard for form submissions.

If a name=value pair should not be URL-encoded, you can specify a NOENCODE flag before the name=value pair.

See [“Using URL-Encoding Data in PROC HTTP” on page 1062](#)

MULTI <FORM | "type"> (inputs)

MULTI specifies a multipart upload.

FORM

specifies that the upload is performed using the specialized multipart type known as `multipart/form-data`. When FORM is used, then a special header named "content-disposition" is automatically generated and sent with each input.

This generated content-disposition header takes the form:

```
Content-Disposition: form-data;
  name="field-name";
  <filename=filename.ext">
```

"type"

indicates that the request is a generic multipart request. In a generic multipart request, the user specifies the value to include in the generated content-type header. For example, if the quoted value is "mixed", a header of `Content-Type: multipart/mixed` is sent. If the quoted value is "related", a header of `Content-Type: multipart/related` is sent.

(inputs)

specifies a comma-separated list of inputs, within parentheses.

An input in a form-data request has the syntax:

"Name"=value <FILENAME="filename" | NOFILENAME>

"Name"=value

(required) describes the basic content in the form. "Name" is the control name of the corresponding control.

Value can be a fileref or a quoted string. The fileref TEMP is allowed.

The **Filename=** field of the generated content-disposition header is automatically generated when the file name can be determined from the fileref.

FILENAME="filename"

(optional) specifies a file name for the **Filename=** field of the generated content-disposition header in the event that it cannot be automatically determined.

NOFILENAME

(optional) specifies to omit the **Filename=** field from the generated content-disposition header. This option enables you to use a fileref as input that is not a file, as far as the form is concerned.

An input for a generic multipart request has the syntax:

value <header>

value

is a fileref or a quoted string.

Header

(optional) specifies zero or more headers. Each header is a proper header in the form:

```
header="header-value"
```

For example:

```
header="Content-Type: application/json"
```

Note Headers must be specified after keywords in a part.

Requirement The IN= option is required when the POST and PUT methods are used.

Note The MULTI option does not include support for nested multipart uploads. See https://www.w3.org/Protocols/rfc1341/7_2_Multipart.html for more information about the multipart content-type.

Example “Uploading Multipart Data with the MULTI Option” on page 1063

MAXREDIRECTS=*n*

specifies the maximum number of redirects that are allowed. PROC HTTP automatically follows redirects coming from a 300-level response code, as long as the NOFOLLOWLOC option is not specified. This option enables you to limit the number of redirects that are performed.

Default 5

METHOD=<">http-method<">

specifies an HTTP method. Standard methods include HEAD, TRACE, GET, POST, PUT, and DELETE. However, the method is open-ended. Any method that conforms to the HTTP/1.1 standard and is recognizable by the target web server is acceptable. For information, see the [HTTP/1.1 specification](http://www.w3.org/Protocols/rfc1341/4_1_Methods.html) at www.w3.org.

Default If you omit the METHOD argument and do not specify the IN argument, the default method is GET. If you omit METHOD and do specify the IN argument, the default method is POST.

Examples “Example 1: A Simple GET Request” on page 1066

“Example 2: A Simple POST Request” on page 1066

“Example 9: A Nonstandard Method” on page 1074

NO_CONN_CACHE

disables connection caching for this HTTP request. The connection will be made with the specified connection parameters.

Note When you enable NO_CONN_CACHE, you forgo all benefits of cached connections, including cached authentication. Each call that uses authentication must re-authenticate, which can take time.

NO_COOKIES

specifies that cached cookies will not be used for this HTTP request. This option does not prevent cookies from being sent manually with the “Cookie” header.

NOFOLLOWLOC

prevents the GET method from following URL redirections.

NOFOLLOWLOC is the default behavior for HTTP methods that write data.

See [“FOLLOWLOC” on page 1051](#)

OAuth_Bearer=token

sends an OAuth access token along with the HTTP call. Valid token values are a string, a fileref, or the constant SAS_SERVICES. String values must be quoted.

For all token types, the argument sends an authorization header in the form:

Authorization: Bearer Value.

In addition to adding a header, OAuth_Bearer= disables other types of HTTP authentication. Any HTTP-Authenticate headers that come back will be ignored.

OUT=fileref-to-response-data

specifies a fileref that indicates where output is written.

Example [“Example 2: A Simple POST Request” on page 1066](#)

PROXY_AUTH_BASIC

specifies to perform user identity authentication through a proxy server. The user name and password are supplied with the PROXYUSERNAME and PROXYPASSWORD arguments.

PROXY_AUTH_NEGOTIATE

specifies to perform Kerberos or some other type of HTTP authentication through a proxy server. As long as your current user identity has permissions, authentication is established.

PROXYHOST="proxy-host-name"

specifies the internet host name of an HTTP proxy server. You can specify both the host name and the port number in the PROXYHOST argument in the form:

host-name:port-number

When this syntax is used, there is no need to specify the PROXYPORT argument.

Example [“Example 3: Specify a Proxy in the HTTP Request” on page 1067](#)

PROXYPASSWORD="proxy-passwd"

specifies an HTTP proxy server password.

Tips The password is required only if your proxy server requires credentials.

Encodings that are produced by PROC PWENCODE are supported.

PROXYPORT=proxy-port-number

optionally specifies an HTTP proxy server port. You are not required to specify PROXYPORT if you specified both the HTTP proxy server host name and port number in the PROXYHOST argument.

PROXYUSERNAME="proxy-user-name"

specifies an HTTP proxy server user name.

Tip The user name is required only if your proxy server requires credentials.

QUERY=(<NOENCODE>"parm1"="value1" "parm2"="value2")

provides an alternate method for submitting query parameters for the URL= argument.

The keyword NOENCODE can be specified before a name=value pair to indicate the pair should not be URL-encoded. NOENCODE applies only to the pair that it precedes. Other characters that normally are encoded are still encoded, like spaces.

Note When QUERY= is specified, if the input URL already has an existing query string, the entire generated replacement string uses an & instead of a ?.

See [“Using URL-Encoding Data in PROC HTTP” on page 1062](#)

TIMEOUT=integer

specifies the number of seconds of inactivity to wait before canceling an HTTP request. Use this option to prevent hangs if there is a chance that the server will not respond. The default value, 0 (zero), means no time-out period.

WEBPASSWORD="basic-authentication-password"

specifies a password for basic authentication.

Alias PASSWORD

Tip Encodings that are produced by PROC PWENCODE are supported.

WEBUSERNAME="basic-authentication-name"

specifies a user name for basic authentication.

Alias USERNAME

DEBUG Statement

Writes debugging information to the SAS log.

Supports: All HTTP methods

Example: [“Example 12: Use the DEBUG Statement with the LEVEL= Option” on page 1079](#)

Syntax

DEBUG *options*;

Optional Arguments

Level= 0 | 1 | 2 | 3

0

no debugging. This is the same as specifying PROC HTTP without the DEBUG statement.

1

displays request and response headers in the log.

Setting a debug level of 1 equates to setting the REQUEST_HEADERS and RESPONSE_HEADERS options in the DEBUG statement.

2

displays request data as well as level 1 messages in the log.

Setting a debug level of 2 equates to setting the REQUEST_HEADERS, RESPONSE_HEADERS, and REQUEST_BODY options in the DEBUG statement.

3

displays response data as well as level 2 messages in the log.

Setting a debug level of 3 equates to setting the REQUEST_HEADERS, RESPONSE_HEADERS, REQUEST_BODY, and RESPONSE_BODY options in the DEBUG statement.

CAUTION

Use level 3 with care. The system may become unstable when the response is binary data.

NO_REQUEST_BODY

suppresses the request body from the information displayed when a debug level of 2 or greater is specified.

NO_REQUEST_HEADERS

suppresses the request header from the information displayed when a debug level of 1 or greater is specified.

NO_RESPONSE_BODY

suppresses the response body from the information displayed when a debug level of 3 is specified.

NO_RESPONSE_HEADERS

suppresses the response header from the information displayed when a debug level of 1 or greater is specified.

OUTPUT_TEXT

displays the request body and response body as if they are text.

REQUEST_BODY

displays the request body in the log.

REQUEST_HEADERS

displays the request header in the log.

RESPONSE_BODY

displays the response body in the log.

RESPONSE_HEADERS

displays the response header in the log.

Details

You must specify at least one option in the DEBUG statement for debugging information to be written to the log.

You can control the amount of information that is printed with the LEVEL= option. Or, you can specify specific components that you want information about.

The options that represent individual components can be specified alone or they can be specified with the LEVEL= option to limit or expand the information that is displayed for a particular LEVEL= value. For example, you can specify RESPONSE_HEADERS to display debugging information for response headers only, or specify both RESPONSE_HEADERS and RESPONSE_BODY to display response information only. Or, you can specify LEVEL=3 to display all debugging information, and also specify NO_RESPONSE_BODY to omit the response body. When LEVEL= is used, a value of 1 or greater is required to display debugging information.

The HTTP request body and response body are written as binary by default. You can set the OUTPUT_TEXT option to print the information as text.

HEADERS Statement

Specifies request headers for the HTTP request.

Supports: All HTTP methods

Note: Use the HEADERS statement instead of the PROC HTTP CT= and HEADERIN= arguments.

Example: [“Example 8: A GET That Uses the HEADERS Statement” on page 1073](#)

Syntax

```
HEADERS "HeaderName"="HeaderValue" < "HeaderName-n"="HeaderValue-n" >;
```

Required Argument

"HeaderName"="HeaderValue"

is a name and value pair that represents a header name and its value. The HeaderName can be a standard header name or a custom header name. For information about header field definitions, see the [HTTP/1.1 specification](http://www.w3.org) at www.w3.org.

Note: Do not specify a colon (:) in the header name. The name=value pairs are automatically translated into the following form:

HeaderName : HeaderValue

Details

The HEADERS statement enables you to specify header values easily within the procedure request, instead of having to provide a fully formatted input file via a fileref. Use the HEADERS statement to specify the content-type and character set of the document that you are uploading when the values are different from the default values for the method.

Table 30.1 Default Content-Type for the POST and PUT Methods

HTTP Method	Default Content-Type
POST	application/x-www-form-urlencoded
PUT	application/octet-stream

SSLPARMS Statement

Sets SSL parameters for the PROC HTTP request.

Requirements: In order to use this feature, the Kubernetes administrator must provide a location that is mounted into the pod or pods where SAS is running in order for any path, value, or parameter set in SSLPARMS to be processed.

The SAS Viya platform requires the use of cert-manager, an X509 certificate management tool for Kubernetes, to provide certificates and private keys. For more information, see “Encryption Overview” in *SAS Viya Platform Encryption: Data in Motion*.

Work with the SAS Viya platform security team to ensure you are working within the SAS Security Framework.

Supports: All HTTP methods

Syntax

SSLPARMS *SSL-options*;

Required Argument

"SSLSystemOption"="OptionValue" <"SSLSystemOption"="OptionValue">

specifies one or more SAS system options for encryption that enable a secure client-server connection with the target server. For a listing of available system options, see "SAS System Options for Encryption" in [SAS Viya Platform Encryption: Data in Motion](#).

Note: Only system options that begin with "SSL" are supported.

SSL system options can be specified in either of the following ways:

"SSLSystemOption"="OptionValue"

SSLSystemOption="OptionValue"

Note SSL options are host-specific.

Details

Use the SSLPARMS statement to apply an SSL option locally instead of globally.

Usage: HTTP Procedure

Using Hypertext Transfer Protocol Secure (HTTPS)

HTTP Security: TLS and Data Encryption

Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL), enables web browsers and web servers to communicate over a secured connection by encrypting data. Both browsers and servers encrypt data before the data is transmitted. The receiving browser or server then decrypts the data before it is processed.

Note: All discussion of TLS is also applicable to the predecessor protocol, SSL.

Using HTTPS with PROC HTTP

Secure communication over HTTP (HTTPS) is provided globally by default on the full SAS Viya platform deployment. HTTPS is controlled by a Trusted Root CA bundle. A default Trusted Root CA bundle is laid down at installation.

PROC HTTP enables you to override the global communication security settings with local communication security settings with the SSLPARMS statement. In the SSLPARMS statement, specify SAS system options for encryption. The options that are set in the SSLPARMS statement apply only to the PROC HTTP request in which they are specified. For more information, see [“SSLPARMS Statement” on page 1059](#).

For information about setting SSL system options on the SAS Viya platform, see [“SAS System Options for Encryption” in SAS Viya Platform Encryption: Data in Motion](#).

Using Authentication Other Than Basic

PROC HTTP enables you to specify the authentication type. The ability to specify the authentication type is useful when you know which authentication type is required for a request to succeed in advance. Specifying the correct type, rather than requiring the procedure to negotiate it, optimizes procedure execution. For example, if you know that the server supports only Kerberos authentication, it is a good idea to specify the AUTH_NEGOTIATE argument.

If you do not specify the authentication type, the default type is AUTH_ANY. AUTH_ANY is equivalent to specifying AUTH_NEGOTIATE and AUTH_BASIC together in the request. If the server that you are connecting to supports the Kerberos authentication protocol, it usually is not necessary to specify a user name and password. As long as your current user identity has permissions, authentication is established.

EXPECT_100_CONTINUE support is provided to optimize requests that must make more than one trip to the server. This option prevents the data from being uploaded multiple times.

Note: When using authentication, do not enable NO_CONN_CACHE. When NO_CONN_CACHE is enabled, each call that uses authentication must re-authenticate, which can take time.

Wire Logging

Wire logging logs packets of information as they appear on the network. This information is normally referred to as a dump. Wire dumps enable you to see what information is being sent to the server and what information the server is sending back. Because you can see the raw data, wire dumps can be useful in debugging your programs.

Logger APP.TK.HTTPC is used to log HTTP-specific messages. The wire dumps that the logger generates can be enabled by setting logger APP.TK.HTTPC to the DEBUG level or higher. At the DEBUG level, the first 64 bytes of incoming and outgoing data is logged. At the TRACE level, all of the data is written to the log. Note that at the TRACE level, performance can be greatly diminished.

For more information, see “SAS Logging” in *SAS Logging: Configuration and Programming Reference*.

Response Encodings

Responses are not encoded to session encodings. You must supply the request with the encoding that you want to use, and set the content type.

Using URL-Encoding Data in PROC HTTP

You can post URL-encoded data with the FORM parameter in the IN= argument as follows. The FORM option enables you to submit a single-part form with PROC HTTP.

```
%let firstname=Mickey;
%let lastname=Mouse;
proc http url="http://httpbin.org/post"
  in = form ("firstname"="%&firstname"
            "lastname"="%&lastname");
run;
```

The QUERY= procedure option enables you to submit URL-encoded query parameters. You can add query parameters to a URL manually as shown below. However, when you modify the URL manually you are responsible for encoding any special characters and making sure that the & (ampersand) appears correctly and is not mistaken for a macro variable.

```
url="http://httpbin.org/GET?firstname=%&firstname
%nrstr(&lastname)=%&lastname";
```

With the new QUERY= procedure option, the above request can now be submitted as follows:

```
%let firstname=Mickey;
%let lastname=Mouse;
proc http url="http://httpbin.org/get"
  query = ("firstname"=&firstname"
          "lastname"=&lastname");
run;
```

The FORM and QUERY options URL-encode all of the data in the specified input string unless you specify the NOENCODE option before a name=value pair.

Uploading Multipart Data with the MULTI Option

An HTTP multipart request is commonly used by browsers and HTTP clients to upload files to an HTTP server. PROC HTTP supports generic multipart submissions and multipart form submissions.

An example of a generic multipart submission is uploading a file with metadata. A file is simply binary data to which you assign a human-readable name. The file's name and information about the file (metadata) are typically described with a JSON body like:

```
{
  "name": "Picture"
}
```

Here is an example of the code required to upload an image file and metadata using PROC HTTP:

```
filename image "profile.png";
filename meta TEMP;

data _null_;
  file meta recfm=f lrecl=1;
  put "{";
  put "\"name\": \"picture\", ";
  put "}";
run;

proc http
  url="httpbin.org/post"
  method="post"
  in = multi "related" ( meta header="Content-Type: application/json",
                        image header="Content-Type: image/png");
run;
```

The example defines filerefs for the metadata and the file to be uploaded. A DATA step creates the metadata content in JSON format. In PROC HTTP, the IN= argument specifies the MULTI keyword, a type of "related", and filerefs representing the message input (within parenthesis). The content type of each fileref is specified as a header within the message body.

Here is an example of how a multipart form might be submitted:

```
filename in 'your-input-file';
filename putHdr temp;
```

```

filename getHdf temp;

data _null_;
  file putHdr recfm=f lrecl=1;
  put "Content-Type: text/plain";
run;

data _null_;
  file getHdr recfm=f lrecl=1;
  put "Content-Type: text/plain";
run;

proc http url="your-web-location"
  in = multi form ( "file"=in)
  HeaderIn=putHdr
  HeaderOut=getHdr;
run;

```

A multipart form request enables you to send entire files for form submission. The example defines filerefs for the input file, and file input and output headers. In PROC HTTP, the IN= argument specifies the MULTI keyword with the FORM keyword and a name-value pair as the message input (within parenthesis). "File" is the control name of the corresponding control. The input file is the value of the pair. The content type for the uploaded file is specified in PROC HTTP's HeaderIn= and HeaderOut= arguments.

Use of the FORM keyword with the MULTI keyword generates the content type "multipart/form-data". w3.org recommends that the content type "multipart/form-data" be used for submitting forms that contain large files, non-ASCII data, and binary data.

Using HEADERS= to Send Chunked Data

To force chunked data to be sent, specify a header as follows:

```

proc http url="httpbin.org/post"
  in="hello";
  headers "Transfer-Encoding"="chunked";
  debug level=3;
run;

```

PROC HTTP Response Status Reporting

PROC HTTP writes response status values to the SAS log.

PROC HTTP also sets up macro variables with certain values after it executes each statement. These macro variables can be used inside a macro to test for HTTP errors. An HTTP error is an error that is encountered after a successful host connection has been made and the HTTP request has been successfully parsed by

the HTTP procedure. The macro variables do not store values for host connection errors or for PROC HTTP syntax errors. The macro variables are reset on each invocation of PROC HTTP.

SYS_PROCHTTP_STATUS_CODE

stores the status code of the HTTP request.

SYS_PROCHTTP_STATUS_PHRASE

stores the descriptive phrase that is associated with the status code.

Consider this example:

```
HTTP/1.1 200 OK
```

The **SYS_PROCHTTP_STATUS_CODE** macro variable stores the value 200. The **SYS_PROCHTTP_STATUS_PHRASE** macro variable stores OK, which indicates that this HTTP request was successful.

For more information, see [“Example 11: Create a Program That Captures Status Response Values” on page 1076](#).

Macro Variables for Setting Global Values

PROC HTTP produces two automatic macro variables to enable you to set or change default PROC HTTP settings.

PROCHTTP_PROXY=*proxy-server-name-and-port-number*;

sets a default proxy server for PROC HTTP requests. Once set, the specified proxy server establishes a proxy for all PROC HTTP requests in the SAS session, unless you specify the **PROXYHOST**= argument in the PROC HTTP request. The value that is specified in the procedure argument overrides the value that is specified in the macro variable. Specify the **PROXYHOST**= argument with a value that is different from the macro variable to use a different proxy server for a request. Specify **PROXYHOST**= without a value to disable proxy use for a request. For more information, see [“Example 5: Specify a Proxy In a Macro Variable” on page 1069](#).

PROCHTTP_NOCOOKIES= blank | *integer*;

provides global control of cookie caching for PROC HTTP requests. Omitting the macro variable or specifying the macro variable without a value enables cookie caching (cookie caching is on by default). To globally disable cookie caching, specify a nonzero value in the macro variable. Cookie caching can be disabled for a specific PROC HTTP request by specifying the **NO_COOKIES** procedure option in the PROC HTTP request along with the **CLEAR_COOKIES** or **CLEAR_CACHE** arguments.

The macro variables are set with the %LET statement. In the event that you disable cookie caching, you can delete the macro variable from the symbol table with the %SYMDEL statement.

Examples: HTTP Procedure

Example 1: A Simple GET Request

Features: METHOD= argument
 URL= Argument
 OUT= Argument

Details

This example makes a GET request to a server on the local network. GET is the simplest and most common request that you can make with PROC HTTP. GET is the default METHOD value when the IN argument is omitted for a PROC HTTP request, making the argument optional.

Program

```
filename resp TEMP;

proc http
  url="http://httpbin.org/get"
  out=resp;
run;
```

Example 2: A Simple POST Request

Features: METHOD= Argument
 IN= Argument
 OUT= Argument

Details

This example makes a simple POST request to a server on the local network. The file to upload is identified by a fileref in the IN argument. When the IN argument is specified, the default METHOD= value in all SAS releases is POST. The response and the output headers are written to filerefs.

Program

```
filename resp TEMP;
filename headout TEMP;
filename input TEMP;

data _null_;
  file input;
  put "this is some sample text";
run;

proc http
  url="http://httpbin.org/post"
  in=input
  out=resp
  headerout=headout;
run;
```

Example 3: Specify a Proxy in the HTTP Request

Features: PROXYHOST= Argument
 PROXYPORT= Argument

Details

This example makes the same request as in [“Example 2: A Simple POST Request” on page 1066](#), except that the call is sent to an external server and, therefore, requires the use of a proxy server. This example specifies both the name of the external server and the proxy port number in the PROXYHOST argument.

Program

```
filename out "u:\prochttp\Testware\ProxyTest_out.txt";
filename input TEMP;

data _null_;
  file input;
  put "this is some sample text";
run;

proc http
  url="http://httpbin.org/post"
  method="post"
  in=input
  out=out
  proxyhost="proxyhost.company.com: 889";
run;
```

Example 4: Specifying Input Data as a String

Features: IN= "string"

Details

The PROC HTTP IN= argument accepts a quoted input string or a fileref to submit input data. Specifying input in a string makes it easier to send text posts and form-based posts. This example submits the form that can be found at <http://httpbin.org/forms/post>. The response is written to a response file.

Program

```
filename resp TEMP;

proc http
  method="post"
  url="http://httpbin.org/post"
  in='custname=Sas+User&custtel=919-555-5555&custemail=sas.user%40
sas.com&size=medium&topping=cheese&delivery=12%3A00&comments=Dont
+Drop+It'
  out=resp;
run;

data _null_;
  infile resp;
  input;
```

```
put _infile_;
run;
```

This is the content of the Resp file:

```
{
  "args": {},
  "data": "",
  "files": {},
  "form": {
    "comments": "Dont Drop It",
    "custemail": "sas.user@sas.com",
    "custname": "Sas User",
    "custtel": "919-555-5555",
    "delivery": "12:00",
    "size": "medium",
    "topping": "cheese"
  },
  "headers": {
    "Accept": "*/*",
    "Content-Length": "133",
    "Content-Type": "application/x-www-form-urlencoded",
    "Host": "httpbin.org",
    "User-Agent": "SAS/9",
  },
  "json": null,
  "origin": "149.173.1.80, 104.129.194.85",
  "url": "http://httpbin.org/post"
}
```

Example 5: Specify a Proxy In a Macro Variable

Features: PROCHTTP_PROXY= Macro Variable
IN="string"

Details

This example makes the same request as in [“Example 3: Specify a Proxy in the HTTP Request” on page 1067](#), except the proxy server is specified in a macro variable and the input text is specified as a string in the IN argument. The PROCHTTP_PROXY macro variable specifies the proxy server’s internet host name and port number as one value. Because the proxy is set in the macro variable, it is available to all subsequent HTTP requests that are made in the SAS session. In this request, parameters to the POST are read from a string that is specified in the IN argument.

Program

```
%let PROCHTTP_PROXY=proxyhost.company.com:889;

filename out "u:\prochttp\Testware\ProxyTest_out.txt";
filename input TEMP;

proc http
  url="http://httpbin.org/post"
  method="post"
  in="text to write out"
  out=out;
run;
```

Example 6: A POST That Captures the Response Headers

Features: IN="string"
 HEADEROUT= Argument

Details

This example makes the same POST request as in [“Example 5: Specify a Proxy In a Macro Variable” on page 1069](#) but captures the response headers in a file called headerOut.txt.

Program

```
%let PROCHTTP_PROXY=proxyhost.company.com:889;

filename out "u:\prochttp\Testware\ProxyTest_out.txt";
filename hdrout "u:\prochttp\Testware\headerOut.txt";

proc http
  url="http://httpbin.org/post"
  method="post"
  in="text to write out"
  out=out
  headerout=hdrout;
run;
```

Example 7: A GET That Specifies HEADEROUT_OVERWRITE

Features: HEADEROUT argument
 HEADEROUT_OVERWRITE argument

Details

This example shows the effects of the HEADEROUT_OVERWRITE argument. The GET requests redirect twice before reaching their destination. HEADEROUT_OVERWRITE causes only the last output header to be recorded.

Example of Normal HEADEROUT Output after a Redirect

```
filename hdrs "u:\prochttp\Testware\GetHdr_out.txt";
filename out  "u:\prochttp\Testware\GetTest_out.txt";

proc http
  url="http://httpbin.org/redirect/2"
  method=GET
  headerout=hdrs
  out=out;
run;
```

This is the content of GetHdr_out.txt:

```

HTTP/1.1 302 FOUND
Server: nginx
Date: Mon, 20 Apr 2015 14:19:52 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 247
Connection: keep-alive
Location: /relative-redirect/1
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true

HTTP/1.1 302 FOUND
Server: nginx
Date: Mon, 20 Apr 2015 14:19:53 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 0
Connection: keep-alive
Location: /get
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true

HTTP/1.1 200 OK
Server: nginx
Date: Mon, 20 Apr 2015 14:19:53 GMT
Content-Type: application/json
Content-Length: 195
Connection: keep-alive
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true

```

Example of HEADEROUT Request with HEADEROUT_OVERWRITE

```

filename hdrs "u:\prochttp\Testware\GetHdr2_out.txt";
filename out "u:\prochttp\Testware\GetTest2_out.txt";

proc http
    url="http://httpbin.org/redirect/2"
    method=GET
    headerout=hdrs
    out=out
    HEADEROUT_OVERWRITE;
run;

```

This is the content of GetHdr2_out.txt:

```

HTTP/1.1 200 OK
Server: nginx
Date: Mon, 20 Apr 2015 14:22:48 GMT
Content-Type: application/json
Content-Length: 195
Connection: keep-alive
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true

```

Example 8: A GET That Uses the HEADERS Statement

Features: HEADERS Statement
GET Method

Details

The following is an example of a GET method request that specifies the HEADERS statement.

Program

```
filename resp TEMP;

proc http
  url="http://httpbin.org/headers"
  out=resp;
  headers
    "Accept"="application/json";
run;

data _null_;
  infile resp;
  input;
  put _infile_;
run;
```

The output looks like this:

```
"headers": {
  "Accept": "*/*,application/json",
  "Host": "httpbin.org",
  "User-Agent": "SAS/9",
}
```

Example 9: A Nonstandard Method

Features: METHOD Argument

Details

This example submits the MKCOL WEBDAV http method. Output is written to a temporary file named Resp. There are no input and output requirements for nonstandard methods. As long as the target server returns data and you have specified a valid OUT, data is written to your OUT fileref. Here, output is written to Resp.

Program

```
filename resp TEMP;

proc http
  url="http://hostname/directory/"
  method=MKCOL
  out=resp;
run;
```

Example 10: A PUT That Specifies EXPECT_100_CONTINUE

Features: EXPECT_100_CONTINUE Argument
 HEADEROUT= Argument

Details

This example specifies the EXPECT_100_CONTINUE header.

Program

```
filename resp TEMP;
filename hdrs TEMP;

proc http
  url="http://httpbin.org/put"
  method=PUT
  in='Some Put Data'
  out=resp
  headerout=hdrs
  EXPECT_100_CONTINUE;
run;

data _null_;
  infile hdrs;
  input;
  put _infile_;
run;

data _null_;
  infile resp;
  input;
  put _infile_;
run;
```

The output in the HDRS looks like this:

```
HTTP/1.1 100 Continue

HTTP/1.1 200 OK
Server: myserver/18.0
Date: Mon, 24 Nov 2014 20:18:29 GMT
Content-Type: application/json
Content-Length: 652
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
X-Cache: MISS from transproxy
Via: 1.1 vegur, 1.1 transproxy (squid)
Connection: keep-alive
```

The output in the Resp file looks like this:

```
{
  "args": {},
  "data": "Some Put Data",
  "files": {},
  "form": {},
  "headers": {
    "Accept": "*/*",
    "Content-Length": "13",
    "Content-Type": "application/octet-stream",
    "Host": "httpbin.org",
    "User-Agent": "SAS/9",
    "Xxpect": "100-continue",
  },
  "json": null,
  "origin": "149.173.1.80, 104.129.194.85",
  "url": "http://httpbin.org/put"
}
```

Example 11: Create a Program That Captures Status Response Values

Features: SYS_PROCHTTP_STATUS_CODE macro variable
 SYS_PROCHTTP_STATUS_PHRASE macro variable

Details

This example creates a simple macro program that tests the value set in the SYS_PROCHTTP_STATUS_CODE macro variable. The program specifies to print an error message when the result code does not match a specified value for the SYS_PROCHTTP_STATUS_CODE macro variable. The program also specifies to print the actual values returned by the SYS_PROCHTTP_STATUS_CODE and SYS_PROCHTTP_STATUS_PHRASE macro variables in any error messages.

The macro program is then invoked after various PROC HTTP requests to illustrate the results that you can expect under various circumstances. The macro program is invoked with a value of 200 in each execution. A value of 200 indicates successful completion of the HTTP request.

Program

The following code creates the macro program.

```
%macro prohttp_check_return(code);
%if %symexist(SYS_PROCHTTP_STATUS_CODE) ne 1 %then %do;
  %put ERROR: Expected &code., but a response was not received from
the HTTP Procedure;
```

```

%abort;
%end;
%else %do;
  %if &SYS_PROCHTTP_STATUS_CODE. ne &code. %then %do;
    %put ERROR: Expected &code., but received &SYS_PROCHTTP_STATUS_CODE.
    &SYS_PROCHTTP_STATUS_PHRASE.;
    %abort;%end;
  %end;
%mend;

```

Result for a Successful HTTP Request

Here is an example of a successful PROC HTTP request. The macro program specifies to print an error message and the return code and status values for any HTTP request that does not return the code value 200.

```

proc http url="httpbin.org/get";
run;
%prochttp_check_return(200);

```

Here is the log for the request. The request returned 200. For PROC HTTP requests whose return code matches the code value specified in the macro program, the values of the macro variables are ignored. There is no error message; therefore, no need to print the value of the status reporting macro variables.

```

173  proc http url="httpbin.org/get";
174  run;

NOTE: PROCEDURE HTTP used (Total process time):
      real time           3.40 seconds
      cpu time            0.03 seconds

NOTE: 200 OK

175
176  %prochttp_check_return(200);
177
178

```

Result for Failed Host Connection

The following is an example of a PROC HTTP request that cannot establish a host connection:

```

proc http url="foo.bar";
run;
%prochttp_check_return(200);

```

When a server connection cannot be made, the procedure returns an error message. However, the error message does not include a return code or a status phrase.

```
179 proc http url="foo.bar";
180 run;

ERROR: Host name resolution failed
NOTE: PROCEDURE HTTP used (Total process time):
      real time          2.28 seconds
      cpu time           0.00 seconds

NOTE: The SAS System stopped processing this step because of errors.
181
182 %prochttp_check_return(200);
ERROR: Expected 200, but a response was not received from the HTTP Procedure
ERROR: Execution terminated by an %ABORT statement.
183
```

Result for a Valid HTTP Error

The following is an example of a PROC HTTP request that connects but specifies an invalid secondary path:

```
proc http url="httpbin.org/get2";
run;
%prochttp_check_return(200);
```

When a server connection is made, but the HTTP request fails, the error message prints the actual return code and its corresponding status phrase.

```
184 proc http url="httpbin.org/get2";
185 run;

NOTE: PROCEDURE HTTP used (Total process time):
      real time          0.04 seconds
      cpu time           0.00 seconds

NOTE: 404 Not Found

186
187 %prochttp_check_return(200);
ERROR: Expected 200, but received 404 Not Found
ERROR: Execution terminated by an %ABORT statement.
188
```

Results for a Parsing Error

The following is an example of a PROC HTTP request that specifies an invalid argument:

```
proc http url="httpbin.org/get" foo;
```

```
run;
%prochttp_check_return(200);
```

When a server connection is made, but the PROC HTTP request cannot be parsed, the macro returns an error message. The message does not include a return code or status phrase.

```
189 proc http url="httpbin.org/get" foo;
      ---
      22
      202
ERROR 22-322: Syntax error, expecting one of the following: ;, AUTH_ANY, AUTH_BASIC,
AUTH_NEGOTIATE, AUTH_NONE, CLEAR_CACHE, CLEAR_CONN_CACHE,
CLEAR_COOKIES, CT, EXPECT_100_CONTINUE, FOLLOWLOC, HEADERIN, HEADEROUT,
HEADEROUT_OVERWRITE, IN, METHOD, NOFOLLOW, NOFOLLOWLOC,
NO_CONN_CACHE, NO_COOKIES, OAUTH_BEARER, OUT, PASSWORD, PROXYHOST,
PROXYPASSWORD, PROXYPORT, PROXYUSERNAME, PROXY_AUTH_BASIC,
PROXY_AUTH_NEGOTIATE,
PROXY_AUTH_NONE, TIMEOUT, URL, USERNAME, VERBOSE, WEBPASSWORD,
WEBUSERNAME.
ERROR 202-322: The option or parameter is not recognized and will be ignored.
190 run;
191
192 %prochttp_check_return(200);
ERROR: Expected 200, but a response was not received from the HTTP Procedure
NOTE: PROCEDURE HTTP used (Total process time):
      real time          0.07 seconds
      cpu time           0.07 seconds

ERROR: Execution terminated by an %ABORT statement.
NOTE: The SAS System stopped processing this step because of errors.
```

Example 12: Use the DEBUG Statement with the LEVEL= Option

Features: DEBUG statement
 LEVEL= statement option
 Statement output

Details

This example shows the information returned by the DEBUG statement Level= option.

Debug Level 1

A debug level of 1 displays the HTTP request and response headers.

```
proc http
  in = "*****testing prohttp*****"
  method="post"
  url="http://httpbin.org/post";
  debug level = 1;
run;
```

The following information is displayed in the log:

```
> POST /post HTTP/1.1
> User-Agent: SAS/9
> Host: httpbin.org
> Accept: */*
> Connection: Keep-Alive
> Content-Length: 45
> Content-Type: application/x-www-form-urlencoded
>
< HTTP/1.1 200 OK
< Connection: keep-alive
< Server: myserver/19.9.0
< Date: Tue, 07 Aug 2018 19:12:34 GMT
< Content-Type: application/json
< Content-Length: 418
< Access-Control-Allow-Origin: *
< Access-Control-Allow-Credentials: true
< Via: 1.1 vegur
<
```

Debug Level 2

A debug level of 2 displays the HTTP request and response headers and the HTTP request body.

```
proc http
  in = "*****testing prohttp*****"
  method="post"
  url="http://httpbin.org/post";
  debug level = 2;
run;
```

The following information is displayed in the log. The request body is written as binary by default. Use the OUTPUT_TEXT DEBUG statement option if you want to write the information as text.

```

> POST /post HTTP/1.1
> User-Agent: SAS/9
> Host: httpbin.org
> Accept: */*
> Connection: Keep-Alive
> Content-Length: 45
> Content-Type: application/x-www-form-urlencoded
>
> 000000000A9FC4C0: 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 74 65 73
*****tes
> 000000000A9FC4D0: 74 69 6E 67 20 70 72 6F 63 68 74 74 70 2A 2A 2A ting
prochttp***
> 000000000A9FC4E0: 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A *****
< HTTP/1.1 200 OK
< Connection: keep-alive
< Server: myserver/19.9.0
< Date: Thu, 06 Sep 2018 14:21:26 GMT
< Content-Type: application/json
< Content-Length: 418
< Access-Control-Allow-Origin: *
< Access-Control-Allow-Credentials: true
< Via: 1.1 vegur
<

```

Debug Level 3

A debug level of 3 displays the request header, response header, request body, and response body.

```

proc http
  in = "*****testing prochttp*****"
  method="post"
  url="http://httpbin.org/post";
  debug level = 3;
run;

```

The following information is displayed in the log. The request and response body are written as binary by default. Use the OUTPUT_TEXT DEBUG statement option if you want to write the information as text.

```

> POST /post HTTP/1.1
> User-Agent: SAS/9
> Host: httpbin.org
> Accept: */*
> Connection: Keep-Alive
> Content-Length: 45
> Content-Type: application/x-www-form-urlencoded
>
> 00000000A9FC4C0: 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 74 65 73 *****tes
> 00000000A9FC4D0: 74 69 6E 67 20 70 72 6F 63 68 74 74 70 2A 2A ting prochtpp**
> 00000000A9FC4E0: 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A *****
< HTTP/1.1 200 OK
< Connection: keep-alive
< Server: myserver/19.9.0
< Date: Thu, 06 Sep 2018 14:25:41 GMT
< Content-Type: application/json
< Content-Length: 418
< Access-Control-Allow-Origin: *
< Access-Control-Allow-Credentials: true
< Via: 1.1 vegur
<
< 00000000A9FC5B6: 7B 0A 20 20 22 61 72 67 73 22 3A 20 7B 7D 2C 20 { . "args": {},
< 00000000A9FC5C6: 0A 20 20 22 64 61 74 61 22 3A 20 22 22 2C 20 0A . "data": " , .
< 00000000A9FC5D6: 20 20 22 66 69 6C 65 73 22 3A 20 7B 7D 2C 20 0A "files": {}, .
< 00000000A9FC5E6: 20 20 22 66 6F 72 6D 22 3A 20 7B 0A 20 20 20 20 "form": { .
< 00000000A9FC5F6: 22 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 74 65 *****te
< 00000000A9FC606: 73 74 69 6E 67 20 70 72 6F 63 68 74 74 70 2A 2A sting prochtpp**
< 00000000A9FC616: 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 22 3A *****":
< 00000000A9FC626: 20 22 22 0A 20 20 7D 2C 20 0A 20 20 22 68 65 61 " . }, . "hea
< 00000000A9FC636: 64 65 72 73 22 3A 20 7B 0A 20 20 20 22 41 63 ders": { . "Ac
< 00000000A9FC646: 63 65 70 74 22 3A 20 22 2A 2F 2A 22 2C 20 0A 20 20 20 20 cept": "*/" , .
< 00000000A9FC656: 20 20 20 22 43 6F 6E 6E 65 63 74 69 6F 6E 22 3A "Connection":
< 00000000A9FC666: 20 22 63 6C 6F 73 65 22 2C 20 0A 20 20 20 20 22 "close", . "
< 00000000A9FC676: 43 6F 6E 74 65 6E 74 2D 4C 65 6E 67 74 68 22 3A Content-Length":
< 00000000A9FC686: 20 22 34 35 22 2C 20 0A 20 20 20 20 22 43 6F 6E "45", . "Con
< 00000000A9FC696: 74 65 6E 74 2D 54 79 70 65 22 3A 20 22 61 70 70 tent-Type": "app
< 00000000A9FC6A6: 6C 69 63 61 74 69 6F 6E 2F 78 2D 77 77 77 2D 66 lication/x-www-f
< 00000000A9FC6B6: 6F 72 6D 2D 75 72 6C 65 6E 63 6F 64 65 64 22 2C 2C orm-urlencoded",
< 00000000A9FC6C6: 20 0A 20 20 20 20 22 48 6F 73 74 22 3A 20 22 68 . "Host": "h
< 00000000A9FC6D6: 74 74 70 62 69 6E 2E 6F 72 67 22 2C 20 0A 20 20 ttpbin.org", .
< 00000000A9FC6E6: 20 20 22 55 73 65 72 2D 41 67 65 6E 74 22 3A 20 "User-Agent":
< 00000000A9FC6F6: 22 53 41 53 2F 39 22 0A 20 20 7D 2C 20 0A 20 20 "SAS/9" . }, .
< 00000000A9FC706: 22 6A 73 6F 6E 22 3A 20 6E 75 6C 6C 2C 20 0A 20 "json": null, .
< 00000000A9FC716: 20 22 6F 72 69 67 69 6E 22 3A 20 22 31 32 2E 37 "origin": "12.7
< 00000000A9FC726: 30 2E 31 30 31 2E 31 31 34 22 2C 20 0A 20 20 22 0.101.114", . "
< 00000000A9FC736: 75 72 6C 22 3A 20 22 68 74 74 70 3A 2F 2F 68 74 url": "http://ht
< 00000000A9FC746: 74 70 62 69 6E 2E 6F 72 67 2F 70 6F 73 74 22 0A tpbin.org/post".
< 00000000A9FC756: 7D 0A } .

```

Example 13: Specify Local Options for Encryption in UNIX

Features:

- SSLPARMS statement
- SSLCERTLOC= system option
- SSLPVTKEYLOC= system option
- SSLPVTKEYPASS= system option

SSLPKCS12LOC= system option
 SSLPKCS12PASS= system option

Details

The following is an example of a PROC HTTP request that specifies encryption options that specify a local certificate on UNIX. On UNIX, the system options that you specify depend on whether the certificate is in PEM or DER format. This example shows the code to send PEM and DER certificates for user John Doe.

SAS PROC HTTP supports secure API communication using TLS client certificate authentication, also known as mutual TLS (mTLS). This is useful when a web service requires both server and client to authenticate using X.509 certificates. You can configure PROC HTTP to send client certificates using either PEM files or PKCS#12 archives by specifying appropriate SSL system options and the SSLPARMS statement.

Note: The paths to the certificates are relative to the location where the web server is running. Therefore, the certificates must be accessible on disk.

For more information about the SSL system options in SAS Viya, see [SAS System Options for Encryption](#) in *SAS Viya Platform Encryption: Data in Motion*.

SSLPARMS Arguments for a PEM Certificate

The SSLCERTLOC= and SSLPVTKEYLOC= system options specify the location of a PEM certificate. SSLPVTKEYPASS= is specified only if needed.

```
proc http
  method=GET
  url="https://internal.site.com/";
  sslparms
    sslcertloc="/users/johndoe/certificates/server.pem"
    sslpvtkeyloc="/users/johndoe/certificates/serverkey.pem"
    sslpvtkeypass="password"
  ;
run;
```

SSLPARMS Arguments for a DER Certificate

The SSLPKCS12LOC= and SSLPKCS12PASS= system options specify the location of a DER certificate.

Example Code 30.1 *System Options for a DER-Encoded Certificate*

```
proc http
```

```
method=GET
url=https://internal.site.com/;
sslparms
  sslpkcs12loc="/users/server/certificates/server.p12"
  sslpkcs12pass="password"
;
run;
```

IMPORT Procedure

Overview: IMPORT Procedure	1085
What Does the IMPORT Procedure Do?	1085
Format Catalog Encodings in SAS Viya Platform	1088
Support for the VARCHAR Data Type	1088
External File Interface (EFI) Macro Variables	1089
Syntax: IMPORT Procedure	1090
PROC IMPORT Statement	1092
DATAROW Statement	1099
DBENCODING Statement	1100
DELIMITER Statement	1100
FMTLIB Statement	1101
GETNAMES Statement	1101
GETSHEETNAMES Statement	1103
GUESSINGROWS Statement	1103
META Statement	1104
VARNAMEROW Statement	1104
Examples: IMPORT Procedure	1105
Example 1: Importing a Delimited File	1105
Example 2: Importing a Specific Delimited File Using a Fileref	1109
Example 3: Importing a Tab-Delimited File	1112
Example 4: Importing a Comma-Delimited File with a CSV Extension	1116

Overview: IMPORT Procedure

What Does the IMPORT Procedure Do?

The IMPORT procedure reads data from an external data source and writes it to a SAS data set. Included are these external data sources:

- delimited files
- JMP files
- Microsoft Excel worksheets
- Paradox files
- SPSS files
- Stata files

In delimited files, a delimiter (such as a blank, comma, or tab) separates columns of data values. If you license SAS/ACCESS Interface to PC Files, additional external data sources can include Microsoft Access database files, Microsoft Excel files, and Lotus spreadsheets.

You can import data from JMP 7 or later files, and JMP variables can be up to 255 characters long. You can also import value labels to a SAS format catalog. Extended attributes are now used automatically, and the META= statement is no longer supported.

The CAS engine also supports JMP, Excel (.xlsx and .xls), SPSS, and Stata files for importing to a CAS table in memory on the CAS server and for exporting CAS tables to the PC file format. To use the CAS engine to import or export CAS data, a connection to the CAS server must be made. When importing into CAS tables, all CHAR data types or strings in the PC file formats are converted to the VARCHAR data type in a CAS table.

When you run the IMPORT procedure, it reads the input file and writes the data to the specified SAS data set. By default, IMPORT procedure expects the variable names to appear in the first row. The procedure scans the first 20 rows to count the variables, and it attempts to determine the correct informat and format for each variable. You can use the IMPORT procedure's statements to do the following:

- indicate how many rows SAS scans for variables to determine the type and length (GUESSINGROWS=)
- indicate at which row SAS begins to read the data (DATAROW=)
- modify whether SAS extracts the variable names (GETNAMES=).

You can also use these same statements to change the default values.

When the IMPORT procedure reads a delimited file, it uses one of these methods to import the data:

- generated DATA step code
- generated SAS/ACCESS code

You can customize the results with options and statements that are specific to the input data source. The IMPORT procedure generates the specified output SAS data set or table and writes information about the import to the SAS log. The log displays the DATA step code or the SAS/ACCESS code that is generated by the IMPORT procedure.

If you need to revise your code after the procedure runs, issue the RECALL command (or press F4) to recall the generated DATA step. At this point, you can add or remove options from the INFILE statement and customize the INFORMAT, FORMAT, and INPUT statements to your data.

If you use this method and modify an informat, also modify the format for that same variable. The informat and format for a given variable also must be of the same type (either character or numeric). In addition, if the type is character, the assigned format should be as long as the variable to avoid truncation when the data is displayed. For example, if a character variable is 400 characters long but has a format of \$char50, then only the first 50 characters are shown when the data is displayed.

To recall your PROC IMPORT code, issue a second RECALL command (or press F4 again).

Note: By default, the IMPORT procedure reads delimited files as varying record-length files. If your external file has a fixed-length format, use a SAS DATA step with an INFILE statement that includes the RECFM=F and LRECL= options. For more information, see the INFILE statement, RECFM= option in [SAS DATA Step Statements: Reference](#).

TIP Sharing Delimited Files Across Hosts: When a delimited file is read into SAS using the IMPORT procedure, each row must end with a host-specific, end-of-line delimiter. If you share delimited files that were created on one host with another host, the default end-of-line delimiters will not match. When this occurs, you must specify the new host's end-of-line delimiter for your files.

- Linux: The default end-of-row delimiter is Linefeed (LF). To read a file that is native to Windows, use a FILENAME statement with the TERMSTR=CRLF option. For more information, see the FILENAME statement in [SAS DATA Step Statements: Reference](#).

PROC IMPORT uses the NLNUM informat instead of the COMMA informat. When you import a file that contains values such as 14,000.01 that have commas, the COMMA informat removes the commas and other non-numeric characters from the numerical values. Removing these characters can cause interpretation errors in the values. NLNUM prevents these errors by using the specified value of the LOCALE system option to interpret numerical values that have commas.

For example, to enter the numerical equivalent of fourteen thousand and one hundredth, a person specifying LOCALE=English_UnitedStates would enter 14,000.01. A person specifying LOCALE=French_France would enter 14.000,01. NLNUM interprets either input value correctly and writes the correct value based on the specified locale. If you read in 14.000,01 with NLNUM and LOCALE=French_France, store it in a data set, and then write it with NLNUM and LOCALE=English_UnitedStates, it is displayed as 14,000.01.

If you want to import a character value such as AAFFGG_06JAN2016:10:04:26 into a SAS data set, SAS might try to read it as a date.

For more information, see:

- [“COMMAw.d Informat” in SAS Formats and Informats: Reference](#)
- [“NLNUMw.d Informat” in SAS National Language Support \(NLS\): Reference Guide](#)

- “Overview of Locale Concepts for NLS” in *SAS National Language Support (NLS): Reference Guide*

Format Catalog Encodings in SAS Viya Platform

SAS Viya platform supports only the UTF-8 encoding.

For more information about the encodings of format catalogs, see [Migrating Data to UTF-8 for the SAS Viya Platform](#) and [SAS Viya FAQ for Processing UTF-8 Data](#).

TIP

- Use the XLSX engine to read UTF-8 data.
- Do not use the `DBMS=xls` option to import spreadsheets that contain UTF-8 data.
- Use Microsoft Excel to convert your .xls spreadsheets to .xlsx spreadsheets before you import them with the `DBMS=xlsx` option.

Support for the VARCHAR Data Type

PROC IMPORT supports the VARCHAR data type in CAS. VARCHAR stores a character variable that can have a varying length. The length that you specify for the variable represents the maximum number of characters that you want to store.

The VARCHAR data type is similar to the CHAR data type. CHAR variables have a length that is measured in terms of bytes. VARCHAR variables have a length that is measured in terms of characters rather than bytes. For information about using VARCHAR, see “Data Types Supported in the CAS DATA Step” in [SAS Cloud Analytic Services: User's Guide](#).

In the following example, the CAS engine is used with the LENGTH statement to create a VARCHAR variable and a CHAR variable. The VARCHAR variable, X, has a length of 30 and the CHAR variable, Y, also has a length of 30.

```
libname mycas cas;
data mycas.string;
  length x varchar(30);
  length y $30;
  x = 'abc'; y = 'def';
run;
proc contents data=mycas.string; run;
```

Here is the output that the code produces.

The CONTENTS Procedure			
Data Set Name	MYCAS.STRING	Observations	1
Member Type	DATA	Variables	2
Engine	CAS	Indexes	0
Created	09/11/2016 13:48:30	Observation Length	48
Last Modified	09/11/2016 13:48:30	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64		
Encoding	utf-8 Unicode (UTF-8)		

Engine/Host Dependent Information	
Data Limit	100MB

Alphabetic List of Variables and Attributes			
#	Variable	Type	Len
1	x	Varchar	30
2	y	Char	30

External File Interface (EFI) Macro Variables

External File Interface (EFI) macro variables can be used to control how PROC IMPORT reads and interprets character values.

EFI_ALLCHARS=YES | NO

Default NO

Notes The value of EFI_ALLCHARS applies to all PROC IMPORT steps that follow.

The AllChars SAS Registry option has the same effect as the EFI_ALLCHARS macro variable.

Example %let EFI_ALLCHARS=YES;

specifies whether all columns default to a CHAR data type to avoid misinterpretation of a SAS datetime. Beginning in SAS 9.4TS1M4, you can specify the EFI macro variable, EFI_ALLCHARS, which causes PROC IMPORT to define all of the variables as character. Here is an example of how to use EFI_ALLCHARS with PROC IMPORT:

```
%let EFI_ALLCHARS=YES;
proc import datafile='c:\temp\class.csv' out=class dbms=csv replace;
run;
%let EFI_ALLCHARS=NO;
```

Reset the value of EFI_ALLCHARS after the PROC IMPORT step. Otherwise, all PROC IMPORT steps that follow EFI_ALLCHARS=YES define all variables as character.

EFI_MISSING_NUMERICS=YES | NO

Default NO

Note The MissingNumerics SAS Registry option has the same effect as the EFI_MISSING_NUMERICS macro variable.

Example %let EFI_MISSING_NUMERICS=YES;

specifies whether a single period is numeric or character. When MissingNumerics is set to No, the single period is interpreted as a character.

EFI_NOQUOTED_DELIMITER=YES | NO

Default NO

Note The QuotedDelimiter SAS Registry option has the same effect as the EFI_NOQUOTED_DELIMITER macro variable.

Example %let EFI_NOQUOTED_DELIMITER=YES;

specifies whether single quotation marks within a variable value are read as delimiters inside open quotation marks or as part of the variable value. If your variables contain delimiters, or to read in a single quotation mark as part of the variable value, set EFI_NOQUOTED_DELIMITER to Yes.

EFI_QUOTED_NUMERICS=YES | NO

Default NO

Note The QuotedNumerics SAS Registry option has the same effect as the EFI_QUOTED_NUMERICS macro variable.

Example %let EFI_QUOTED_NUMERICS=YES;

specifies whether to read numbers that are enclosed in quotation marks as character or numeric values. When PROC IMPORT reads an external file, values that are enclosed in quotation marks are automatically created as character values. Setting EFI_QUOTED_NUMERICS to Yes enables you to read quoted numbers as numeric. For example, "10MAY14" is a character value, but with EFI_QUOTED_NUMERICS set to Yes, it becomes 10MAY14, which is a date value.

Syntax: IMPORT Procedure

Restrictions: The IMPORT procedure is available for the following operating environments:

- Linux

A pathname for a file can have a maximum length of 201 characters.

Interaction: All data with the percent sign (%) is considered character data to avoid misinterpretation. Percentage data is considered character data because of the danger of misinterpretation.

- Supports:** PROC IMPORT supports the CSV, TAB, DLM, and JMP file types in CAS.
- Note:** You can use PROC IMPORT to import an external file to a SAS data set or table.
- Tips:** PROC IMPORT supports the VARCHAR data type for CAS tables. For more information, see [“Support for the VARCHAR Data Type” on page 1088](#).
Use the XLSX engine to read UTF-8 data. For more information, see [“LIBNAME Statement: XLSX Engine” in SAS/ACCESS Interface to PC Files: Reference for the SAS Viya Platform](#).
Do not use the DBMS=xls option to import spreadsheets that contain UTF-8 data. Use Microsoft Excel to convert your .xls spreadsheets to .xlsx spreadsheets before you import them with DBMS=xlsx.
- See:** [“ANYDTDTMw. Informat” in SAS Formats and Informats: Reference](#)

PROC IMPORT

```
DATAFILE="filename" | TABLE="tablename"
OUT=<libref.>SAS data set <(SAS data set options)>
<DBMS=identifier> <REPLACE>;
statements for importing from delimited files
DATAROW=n;
DELIMITER=char" | 'nnx;
GETNAMES=YES | NO;
GETSHEETNAMES=YES | NO;
GUESSINGROWS=n | MAX;
VARNAMEROW=n;
statements for importing from JMP files
DBENCODING=12-char SAS encoding-value;
FMTLIB=<libref.>format-catalog;
META=libref.member-data-set;
```

Statement	Task	Example
PROC IMPORT	Import an external data file to a SAS data set	Ex. 1, Ex. 2, Ex. 3, Ex. 4
DATAROW	Start reading data from a specific row in the delimited text file	Ex. 3
DBENCODING	Indicate the encoding character set to use for the JMP file	
DELIMITER	Specify the delimiter that separates columns of data in the input file	Ex. 1, Ex. 3, Ex. 4
FMTLIB	Save value labels to the specified SAS format catalog	
GETNAMES	Generate SAS variable names from the data values in the first row in the input file	Ex. 1, Ex. 2

Statement	Task	Example
GETSHEETNAMES	Returns the worksheet names within an Excel file as a SAS data set.	
GUESSINGROWS	Specify the number of rows of the input file to scan to determine the appropriate data type and length for the variables	
META	Save JMP metadata information to the specified SAS data set	
VARNAMEROW	Specifies a row from the file that contains the variable names for delimited files in PROC IMPORT.	

PROC IMPORT Statement

Imports an external data file to a SAS data set or table .

- Restrictions:** A pathname for a file can have a maximum length of 201 characters.
If the CAS LIBNAME engine is used to import a CAS table, you must have a connection to the CAS server
- Notes:** SAS Viya platform supports only the UTF-8 encoding and runs only on Linux.
Invalid UTF-8 characters are replaced with question marks so that Excel can open the file.
- Tips:** PROC IMPORT supports all access types that are available in the FILENAME statement.
PROC IMPORT supports the VARCHAR data type for CAS tables. For more information, see [“Support for the VARCHAR Data Type” on page 1088](#).
Use the XLSX engine to read UTF-8 data. For more information, see [“LIBNAME Statement: XLSX Engine” in SAS/ACCESS Interface to PC Files: Reference for the SAS Viya Platform](#).
Do not use the DBMS=xls option to import spreadsheets that contain UTF-8 data.
Use Microsoft Excel to convert your .xls spreadsheets to .xlsx spreadsheets before you import them with DBMS=xlsx.
- See:** [Chapter 31, “IMPORT Procedure,” on page 1085](#)

Syntax

PROC IMPORT

```
DATAFILE="filename " | TABLE="tablename "
OUT=<libref .>SAS data set <(SAS data set options)>
<DBMS=identifier> <REPLACE>;
```

Summary of Optional Arguments

DBMS=*data-source-identifier*

specifies the type of data to import.

<file-format-specific-statements>

REPLACE

overwrites an existing SAS data set.

SAS data set options

specifies SAS data set options.

SERVERAUTHDOMAIN=*server authentication-domain*

Required Arguments

DATAFILE=*"filename"* | *"fileref"*

specifies the complete path and filename or fileref for the input PC file, spreadsheet, or delimited external file. A fileref is a SAS name that is associated with the physical location of the output file. To assign a fileref, use the FILENAME statement. For more information about the FILENAME statement, see [SAS Global Statements: Reference](#). For more information about PC file formats, see [SAS/ACCESS Interface to PC Files: Reference for the SAS Viya Platform](#).

If you specify a fileref or if the complete path and filename does not include special characters such as the backslash in a path, lowercase characters, or spaces, then you can omit the quotation marks.

Alias **FILE=**

Restrictions The IMPORT procedure does not support device types or access methods for the FILENAME statement except for DISK. For example, the IMPORT procedure does not support the TEMP device type, which creates a temporary external file.

The IMPORT procedure can import data only if SAS supports the data type. SAS supports numeric and character types of data but not (for example) binary objects.

The CAS engine supports NUMERIC, CHAR, and VARCHAR data types. When a CHAR or string value in a PC file is imported to a CAS table, the CAS engine converts the CHAR to VARCHAR in the CAS table. If the data that you want to import is a type that SAS does not support, the IMPORT procedure might not be able to import it correctly. In many cases, the procedure attempts to convert the data to the best of its ability. However, conversion is not possible for some types.

Interactions By default, the IMPORT procedure reads delimited files as varying record-length files. If your external file has a fixed-length format, use a SAS DATA step with an INFILE statement that includes the RECFM=F and LRECL= options. For more information, see the [INFILE statement](#).

When you use a *fileref* to specify a delimited file to import, the logical record length (LRECL) defaults to 256, unless you specify the LRECL= option in the FILENAME statement. The maximum LRECL that the IMPORT procedure supports is 32767.

For delimited files, the first 20 rows are scanned to determine the variable attributes. You can increase the number of rows that are scanned by using the GUESSINGROWS= statement. All values are read in as character strings. If a Date and Time format or a numeric informat can be applied to the data value, the type is declared as numeric. Otherwise, the type remains character.

For some input data sources, such as a Microsoft Excel workbook, the first eight rows of data are scanned. The most prevalent data type (numeric or character) is used for a column. This is the default. If most of the data in the first eight rows is missing, SAS defaults to the CHAR data type and any subsequent numeric data for that column is set to missing.

The CAS engine defaults CHAR to VARCHAR for CAS tables in memory on the CAS server. Other file formats might have other default values.

- | | |
|----------|---|
| Note | For information about how SAS converts data types, see the specific information for the data source file format that you are importing. |
| See | The FILENAME statement in <i>SAS DATA Step Statements: Reference</i> |
| Examples | <p>“Example 1: Importing a Delimited File” on page 1105</p> <p>“Example 2: Importing a Specific Delimited File Using a Fileref” on page 1109</p> <p>“Example 3: Importing a Tab-Delimited File” on page 1112</p> <p>“Example 4: Importing a Comma-Delimited File with a CSV Extension” on page 1116</p> |

OUT=<libref.>SAS data set

identifies the output SAS data set with either a one or two-level SAS name (library and member name). If the specified SAS data set does not exist, the IMPORT procedure creates it. If you specify a one-level name, by default the IMPORT procedure uses either the USER library (if assigned) or the WORK library (if USER is not assigned). The physical data set name is created using upper-case letters. In SAS programs running on case-sensitive operating systems, this data set name might need to be referenced with upper-case letters.

A SAS data set name can contain a single quotation mark when the VALIDMEMNAME=EXTEND system option is also specified. Using VALIDMEMNAME= expands the rules for the names of certain SAS members, such as a SAS data set name. For more information, see [“Rules for Naming SAS Data Sets and Variables” in SAS Programmer’s Guide: Essentials](#).

Examples [“Example 1: Importing a Delimited File” on page 1105](#)

[“Example 2: Importing a Specific Delimited File Using a Fileref” on page 1109](#)

[“Example 3: Importing a Tab-Delimited File” on page 1112](#)

[“Example 4: Importing a Comma-Delimited File with a CSV Extension” on page 1116](#)

TABLE=“*tablename*”

specifies the name of the input DBMS table. If the name does not include special characters (such as question marks), lowercase characters, or spaces, you can omit the quotation marks. Note that the DBMS table name might be case sensitive.

Requirements You must have a license for SAS/ACCESS Interface to PC Files to import to a DBMS table.

When you import a DBMS table, you must specify the DBMS= option.

When importing Microsoft Access database tables, SAS/ACCESS converts the table name to a SAS member name. Note that SAS does not support member names longer than 32 bytes. When you import a DBMS table, you must specify the DBMS= identifier option.

Optional Arguments

DBMS=*data-source-identifier*

specifies the type of data to import. To import a DBMS table, specify DBMS= using a supported database identifier listed in Summary of DBMS Specifications in SAS/ACCESS Interface to PC Files: Reference for SAS Viya Platform. For example, DBMS=CSV specifies to import a delimited file with comma-separated values.

To import a JMP file, specify JMP as the identifier. The JMP file format must be Version 7 or later, and JMP variable names can be up to 255 characters long. SAS supports importing JMP files that have more than 32,767 variables.

To import a tab-delimited file, specify TAB as the identifier. To import any other delimited file that does not end in .CSV, specify DLM as the identifier. For a comma-separated file with a .CSV extension, DBMS= is optional. The IMPORT procedure recognizes .CSV as an extension for a comma-separated file.

The DBMS argument is required if you are importing a file that does not have an extension, and the data is delimited by tabs. It is also required if you are importing a .TXT file that has data that is delimited with a comma.

This table shows the available DBMS= identifiers. Except as noted in the table, all DBMS= specifications refer to local access.

CSV

The output data source is a comma-separated delimited file. The file extension is .CSV.

DLM

The output data source is a delimited file. You can use DELIMITER='char' to specify the type of data to export.

DTA

The output data source is a Stata file. The file extension is .dta.

JMP

The output data source is a JMP file, Version 7 or later format. The file extension is .JMP.

PARADOX

The output data source is a Paradox DB file. The file extension is .db.

SAV

The output data source is a SPSS file. The file extension is .sav.

TAB

The output data source is a tab delimited file. The file extension is .txt.

XLS

Microsoft Excel 97, 2000, 2002, or 2003 spreadsheet using file formats. The file extension is .xls.

XLSX

Excel 2007, 2010, and later spreadsheet, using .xlsx file format. The file extension is .xlsx.

Transcoding is not supported for DBMS=XLS. Save your .xls file to an .xlsx file and use DBMS=XLSX to support transcoding.

You can specify DBMS=XLSX as well as DBMS=XLS in order to read and write to Excel workbooks under Linux. You do not need to use the SAS PC Files Server.

SAS recommends using .xlsx for its support and enhancements.

Note: All DBMS= specifications refer to local access, except for DBMS=PCFS.

Restriction The availability of a data source depends on the operating environment and in some cases, the platform and whether your site has a SAS/ACCESS Interface for PC Files license.

See [DBMS Identifiers Supported in SAS on page 648](#) for more information about identifiers for this option.

[“DELIMITER Statement” on page 1100](#)

Examples [“Example 1: Importing a Delimited File” on page 1105](#)

[“Example 2: Importing a Specific Delimited File Using a Fileref” on page 1109](#)

[“Example 3: Importing a Tab-Delimited File” on page 1112](#)

[“Example 4: Importing a Comma-Delimited File with a CSV Extension” on page 1116](#)

Microsoft Excel

When you specify DBMS=XLS or DBMS=XLSX for an Excel file, you can read and write to Excel workbooks under Linux directly without having to access the SAS PC Files Server. This example demonstrates use of DBMS=XLSX specifying a range of cells.

```
proc import datafile="fieldtypes.xlsx"
  out=small dbms=xlsx;
  range=colsb_d;
run;
```

Note: When you specify DBMS=XLS with PROC IMPORT, if two or more characters are not valid in the regular naming convention, the code switches the variable names to **Var#**.

If you specify DBMS=XLSX, the IMPORT procedure can read only EXCEL2007 or later versions of the files that are saved in Microsoft Excel workbooks.

This table lists the versions of Microsoft Excel files that each DBMS= identifier supports.

Table 31.1 Microsoft Excel Workbook Specifications

DBMS= Identifier	Excel 2007 and later	Excel 97, 2000, 2002, 2003	Excel 5.0, 95	Excel 4.0
XLS	No	Yes	Yes	No
XLSX	Yes	No	No	No

This table lists the Excel-specific DBMS= identifier, the software that it uses, any required software, and the operating platform that can use the DBMS= specification.

Table 31.2 DBMS Specifications for Excel on Linux

DBMS= Identifier	Uses	Requires
XLS	File formats technology	
XLSX	File formats technology	

This table is a quick reference for which DBMS= data source identifier to use for Microsoft Excel files based on the location of your data.

Table 31.3 DBMS Data Source Identifiers for Microsoft Excel and Access Files

DBMS= Identifier	Linux
XLS	yes
XLSX	yes

Note Transcoding and multi-byte characters are not supported for DBMS=XLS. The output yields unpredictable results. When you use Linux, use DBMS=XLSX for transcoding.

REPLACE

overwrites an existing SAS data set. If you omit REPLACE, the IMPORT procedure does not overwrite an existing data set.

CAUTION

Using the IMPORT procedure with the REPLACE option to write to an existing SAS generation data set causes the most recent (base) generation data set or group of generation data sets to be deleted.

If you write to an existing generation data set using the IMPORT procedure with the REPLACE option and you do one of the following:

- specify the GENMAX= data set option to increase or decrease the number of generations, then all existing generations are deleted and replaced with a single new base generation data set
- omit the GENMAX= data set option, then all existing generations are deleted and replaced with a single new data set by the same name, but it is not a generation data set

Instead, use a SAS DATA step with the REPLACE= data set option to replace a permanent SAS data set and to maintain the generation group for that SAS data set. For more information, see [“Definitions for Generation Data Sets” in SAS V9 LIBNAME Engine: Reference](#).

Examples [“Example 1: Importing a Delimited File” on page 1105](#)

[“Example 2: Importing a Specific Delimited File Using a Fileref” on page 1109](#)

[“Example 3: Importing a Tab-Delimited File” on page 1112](#)

[“Example 4: Importing a Comma-Delimited File with a CSV Extension” on page 1116](#)

SAS data set options

specifies SAS data set options. For example, to assign a password to the resulting SAS data set, you can use the ALTER=, PW=, READ=, or WRITE= data

set option. To import only data that meets a specified condition, you can use the WHERE= data set option.

Restriction You cannot specify data set options when importing delimited, comma-separated, or tab-delimited external files.

See *SAS Data Set Options: Reference*

<file-format-specific-statements>

See [“File Format-Specific Reference for the IMPORT and EXPORT Procedures” in SAS/ACCESS Interface to PC Files: Reference for the SAS Viya Platform](#) for the supported syntax for your data source.

SERVERAUTHDOMAIN=server authentication-domain

Allows connection to a server by specifying the name of an authentication domain metadata object.

Note: This option is available beginning in 2025.02.

Note: SERVERAUTHDOMAIN provides more security and should be used instead of SERVERPASS and SERVERUSER.

DATAROW Statement

Starts reading data from the specified row number in the delimited text file.

Default: When GETNAMES=NO: 1, when GETNAMES=YES: 2

Restriction: When GETNAMES=NO, DATAROW must be equal to or greater than 1. When GETNAMES=YES, DATAROW must be equal to or greater than 2.

Interaction: The DATAROW statement is valid for delimited files and XLSX files.

See: [“GETNAMES Statement” on page 1101](#)

Example: [“Example 3: Importing a Tab-Delimited File” on page 1112](#)

Syntax

DATAROW=*n*;

Required Argument

n

specifies the row number in the input file for the IMPORT procedure to start reading data.

DBENCODING Statement

Indicates the encoding character set to use for the JMP file.

Interaction: The DBENCODING statement is valid only when DBMS=JMP.

Syntax

DBENCODING=*12-char SAS encoding-value*;

Required Argument

12-char SAS encoding-value

indicates the encoding to use with JMP files. Encoding maps each character in a character set to a unique numeric representation, which results in a table of code points. A single character can have different numeric representations in different encodings. This value can be up to 12 characters long.

DELIMITER Statement

Specifies the delimiter that separates columns of data in the input file.

Defaults: Comma for .CSV files
Blank space for all other file types

Interaction: If you specify DBMS=DLM, you must also specify the DELIMITER= statement.

See: [“DBMS=data-source-identifier”](#) for a list of DBMS identifiers supported in Viya.

Example: [“Example 1: Importing a Delimited File”](#) on page 1105

Syntax

DELIMITER=*char* | *'nnx*;

Required Argument

char* | *'nn*'*x

specifies the delimiter that separates columns of data in the input file. You can specify the delimiter as a single character or as a hexadecimal value. For example, if columns of data are separated by an ampersand, specify DELIMITER='&'.

Note: If you omit DELIMITER=, the IMPORT procedure assumes that the delimiter is a space.

The DELIMITER statement is required when you import a file that meets any of these criteria:

- a file that does not have a file extension
- a file that has a .TXT extension and contains data that is delimited by anything other than tabs
- a file that has a .TXT, .CSV, or .JMP extension and contains data that is delimited by blank spaces

This example shows how to use the DBMS argument and the DELIMITER statement to specify a comma delimiter for a file that has a .TXT extension.

```
proc import datafile="C:\temp\test.txt"
  out=test
  dbms=dlm
  replace;
  dlm=', ';
run;
```

FMTLIB Statement

Saves value labels to the specified SAS format catalog.

Interaction: The FMTLIB statement is valid only when DBMS=JMP.

Syntax

FMTLIB=*<libref.>format-catalog*;

Required Argument

<libref.>format-catalog

specifies the format catalog where the value labels are saved.

GETNAMES Statement

Specifies whether the IMPORT procedure generates SAS variable names from the data values in the first row in the input file.

Default: YES

Restrictions: Valid only with the IMPORT procedure.

If VALIDVARNAME=ANY is used, GETNAMES= might not prefix an underscore to the data value.

Interaction: The GETNAMES statement is valid only for delimited files.

Examples: [“Example 1: Importing a Delimited File” on page 1105](#)
[“Example 2: Importing a Specific Delimited File Using a Fileref” on page 1109](#)
[“Example 4: Importing a Comma-Delimited File with a CSV Extension” on page 1116](#)

Syntax

GETNAMES=YES | NO | COMPAT ;

Required Arguments

YES | NO

YES specifies that the IMPORT procedure generate SAS variable names from the data values in the first row of the imported delimited file.

NO specifies that the IMPORT procedure generate SAS variable names as VAR1, VAR2, and so on.

Note: If a data value in the first row in the input file is read and it contains special characters that are not valid in a SAS name, such as a blank, then SAS converts the character to an underscore. For example, the variable name `Occupancy Code` would become the SAS variable name `Occupancy_Code`. Because SAS variable names cannot begin with a number, GETNAMES= prefixes an underscore to a variable name rather than replace the value's first character. For example, `2014.CHANGES` becomes `_2014_CHANGES`.

COMPAT

specifies that the former method of generating SAS names based on available names from the data file is implemented.

IMPORTANT COMPAT is only implemented for XLSX.

Note You can also specify the former method of generating SAS names with the [“IMPORT_COMPAT_VARNAME Environment Variable” in SAS Global Statements: Reference](#) and the [“LIBNAME Statement: XLSX Engine” in SAS/ACCESS Interface to PC Files: Reference for the SAS Viya Platform](#).

Example `getnames=COMPAT;`

GETSHEETNAMES Statement

Returns the worksheet names within an Excel file as a SAS data set.

Default: NO

Restriction: Valid only with the DBMS=XLS or DMBS=XLSX arguments with the IMPORT procedure.

Syntax

GETSHEETNAMES=YES | NO;

Required Argument

YES | NO

- YES specifies that the IMPORT procedure return worksheet names from an Excel file as a SAS data set.
- NO specifies that the IMPORT procedure does not return worksheet names from an Excel file as a SAS data set.

GUESSINGROWS Statement

Specifies the number of rows of the file to scan to determine the appropriate data type and length for the variables.

Default: 20

Restriction: This value should be greater than the value specified for DATAROW.

Interaction: The GUESSINGROWS statement is valid only for delimited files.

Syntax

GUESSINGROWS=*n* | MAX;

Required Arguments

- n*** indicates the number of rows the IMPORT procedure scans in the input file to determine the appropriate data type and length of variables. The range is 1 to

2147483647 (or MAX). The scan data process scans from row 1 to the number that is specified by the GUESSINGROWS option.

Note: You can change the default row value in the SAS Registry. From the SAS command line, enter `regedit`. When the **Registry Editor** opens, select **Products** ⇒ **BASE** ⇒ **EFI** ⇒ **GuessingRows**.

MAX

can be specified instead of 2147483647. Specifying the maximum value could adversely affect performance.

META Statement

Saves JMP metadata information to the specified SAS data set. (Deprecated)

Interaction: The META statement is valid only when DBMS=JMP.

Syntax

META=*libref.member-data-set*;

Required Argument

libref.member-data-set

specifies the SAS data set that contains the metadata information is to be written.

The META statement is no longer supported for importing a JMP file and is ignored. Instead, extended attributes are automatically used. When importing a JMP file with extended attributes, the attributes are automatically attached to the new SAS data set.

The META statement can remain in programs, yet it generates a NOTE in the log saying that META has been replaced by extended attributes and is ignored.

VARNAMEROW Statement

Specifies a row from the delimited text file that contains the variable names.

Default: When GETNAMES=YES: 1; When GETNAMES=NO, no VARNAMEROW is used.

Restrictions: Valid only with the IMPORT procedure.
When VARNAMEROW statement is used, the DATAROW value must be greater than VARNAMEROW value.

Interactions: The VARNAMEROW statement is valid only for delimited files.

Note: If GETNAMES=NO, VARNAMEROW statement is ignored.

See: VARNAMEROW is valid beginning in 2024.05.

“GETNAMES Statement”

“DATAROW Statement”

Syntax

VARNAMEROW=*n*;

Required Argument

n
 specifies the row number in the input delimited file that contains the variable names.

Details

The VARNAMEROW statement specifies the row number of the delimited file where the variable names are located. This row must precede the DATAROW specification. If DATAROW is not specified, the VARNAMEROW+1 is used. GETNAMES=NO overrides any VARNAMEROW specification.

Examples: IMPORT Procedure

Example 1: Importing a Delimited File

Features:

- PROC IMPORT statement options
 - DATAFILE=
 - DBMS=
 - GETNAMES=
 - OUT=
 - REPLACE
- DELIMITER= statement
- OPTIONS statement
- PRINT procedure

Details

This example imports the following delimited external file and creates a temporary SAS data set named WORK.MYDATA:

```
Region&State&Month&Expenses&Revenue
Southern&GA&JAN2001&2000&8000
Southern&GA&FEB2001&1200&6000
Southern&FL&FEB2001&8500&11000
Northern&NY&FEB2001&3000&4000
Northern&NY&MAR2001&6000&5000
Southern&FL&MAR2001&9800&13500
Northern&MA&MAR2001&1500&1000
;
```

Program

```
options nodate ps=60 ls=80;

proc import datafile="C:\My Documents\myfiles\delimiter.txt"
           dbms=dlm
           out=mydata
           replace;

           delimiter='&';
           getnames=yes;

run;

proc print data=mydata;
run;
```

Program Description

Set your system options. The NODATE option suppresses the display of the date and time in the output. The LINESIZE= option specifies the output line length, and the PAGESIZE= option specifies the number of lines on an output page.

```
options nodate ps=60 ls=80;
```

Specify the input file. Specify that the input file is a delimited file. Replace the data set if it exists. Identify the output SAS data set.

```
proc import datafile="C:\My Documents\myfiles\delimiter.txt"
           dbms=dlm
           out=mydata
           replace;
```


Specify delimiter as an & (ampersand).

```
delimiter='&';
```

Generate variable names from first row of data.

```
getnames=yes;
```

```
run;
```

Print out the output data set.

```
proc print data=mydata;  
run;
```

Log

The SAS log displays information about the successful import. For this example, the IMPORT procedure generates a SAS DATA step, as shown in the partial log that follows.

Example Code 31.1 External File Imported to Create a SAS Data Set

```

1  options nodate ps=60 ls=80; proc import datafile="C:\My
1  ! Documents\myfiles\delimiter.txt" dbms=dml out=mydata replace; delimiter='&'
1  ! delimiter='&'; getnames=yes;run;

2  /*****
3  *   PRODUCT:   SAS
4  *   VERSION:   9.3
5  *   CREATOR:   External File Interface
6  *   DATE:      31JAN11
7  *   DESC:      Generated SAS Datastep Code
8  *   TEMPLATE SOURCE: (None Specified.)
9  *****/
10  data WORK.MYDATA ;
11  %let _EFIERR_ = 0; /* set the ERROR detection macro variable */
12  infile 'C:\My Documents\myfiles\delimiter.txt' delimiter = '&' MISSOVER
12 ! DSD lrecl=32767 firstobs=2 ;
13      informat Region $8. ;
14      informat State $2. ;
15      informat Month MONYY7. ;
16      informat Expenses best32. ;
17      informat Revenue best32. ;
18      format Region $8. ;
19      format State $2. ;
20      format Month MONYY7. ;
21      format Expenses best12. ;
22      format Revenue best12. ;
23  input
24      Region $
25      State $
26      Month
27      Expenses
28      Revenue
29      ;
30  if _ERROR_ then call symputx('_EFIERR_',1); /* set ERROR detection
30 ! macro variable */
31  run;

```

NOTE: The infile 'C:\My Documents\myfiles\delimiter.txt' is:
 Filename=C:\My Documents\myfiles\delimiter.txt,
 RECFM=V,LRECL=32767,File Size (bytes)=254,
 Last Modified=31Jan2011:11:44:29,
 Create Time=31Jan2011:11:44:29

NOTE: 7 records were read from the infile 'C:\My Documents\myfiles\delimiter.txt'.
 The minimum record length was 29.
 The maximum record length was 30.

NOTE: The data set WORK.MYDATA has 7 observations and 5 variables.

NOTE: DATA statement used (Total process time):

real time	0.06 seconds
cpu time	0.03 seconds

7 rows created in WORK.MYDATA from C:\My Documents\myfiles\delimiter.txt.

NOTE: WORK.MYDATA data set was successfully created.

Output

Output 31.1 Data Set Work.MyData

The SAS System					
Obs	Region	State	Month	Expenses	Revenue
1	Southern	GA	JAN2001	2000	8000
2	Southern	GA	FEB2001	1200	6000
3	Southern	FL	FEB2001	8500	11000
4	Northern	NY	FEB2001	3000	4000
5	Northern	NY	MAR2001	6000	5000
6	Southern	FL	MAR2001	9800	13500
7	Northern	MA	MAR2001	1500	1000

Example 2: Importing a Specific Delimited File Using a Fileref

Features:

- PROC IMPORT statement options
 - DATAFILE=
 - DBMS=
 - GETNAMES=
 - OUT=
 - REPLACE
- FILENAME statement
- PRINT procedure

Details

This example imports the following space-delimited file and creates a temporary SAS data set named Work.States.

```

Region State Capital Bird
South Georgia Atlanta 'Brown Thrasher'
South 'North Carolina' Raleigh Cardinal
North Connecticut Hartford Robin
West Washington Olympia 'American Goldfinch'
Midwest Illinois Springfield Cardinal

```

Program

```
filename stdata 'c:\temp\state_data.txt' lrecl=100;

proc import datafile=stdata
    dbms=dlm
    out=states
    replace;

    delimiter=' ';
    getnames=yes;
run;

proc print data=states;
run;
```

Program Description

Specify a filename.

```
filename stdata 'c:\temp\state_data.txt' lrecl=100;
```

Specify the input file. Specify the input file is a delimited file. Replace the data set if it exists. Identify the output SAS data set.

```
proc import datafile=stdata
    dbms=dlm
    out=states
    replace;
```

Specify a blank value for the DELIMITER statement. Generate variable names from the first row of data with the GETNAMES statement.

```
    delimiter=' ';
    getnames=yes;
run;
```

Print out the data set.

```
proc print data=states;
run;
```

Log

The SAS log displays information about the successful import. For this example, the IMPORT procedure generates a SAS DATA step, as shown in the partial log that follows.

Example Code 31.2 Importing a Specific Delimited File Using a Fileref

```

324 filename stdata 'c:\myfiles\state_data.txt' lrecl=100;
325
326 proc import datafile=stdata
327             dbms=dlm
328             out=states
329             replace;
330             delimiter=' ';
331             getnames=yes;
332
333 run;

334 /*****
335 *   PRODUCT:   SAS
336 *   VERSION:   9.4
337 *   CREATOR:   External File Interface
338 *   DATE:      18APR14
339 *   DESC:      Generated SAS Daststep Code
340 *   TEMPLATE SOURCE: (None Specified.)
341 *****/
342 data WORK.STATES ;
343   %let _EFIERR_ = 0; /* set the ERROR detection macro variable */
344   infile STDATA delimiter = ' ' MISOVER DSD lrecl=32767 firstobs=2 ;
345   informat Region $7. ;
346   informat State $16. ;
347   informat Capital $11. ;
348   informat Bird $20. ;
349   format Region $7. ;
350   format State $16. ;
351   format Capital $11. ;
352   format Bird $20. ;
353   input
354           Region $
355           State $
356           Capital $
357           Bird $
358   ;
359   if _ERROR_ then call symputx('_EFIERR_',1); /* set ERROR detection
359! macro variable */
360   run;

```

NOTE: The infile STDATA is:
 Filename=c:\myfiles\state_data.txt,
 RECFM=V,LRECL=32767,File Size (bytes)=225,
 Last Modified=15Apr2014:11:27:14,
 Create Time=15Apr2014:11:25:38

NOTE: 5 records were read from the infile STDATA.
 The minimum record length was 32.
 The maximum record length was 44.

NOTE: The data set WORK.STATES has 5 observations and 4 variables.

NOTE: DATA statement used (Total process time):
 real time 0.03 seconds
 cpu time 0.03 seconds

5 rows created in WORK.STATES from STDATA.

NOTE: WORK.STATES data set was successfully created.

Output

Output 31.2 *Work.States Data Set*

The SAS System				
Obs	Region	State	Capital	Bird
1	South	Georgia	Atlanta	Brown Thrasher
2	South	North Carolina	Raleigh	Cardinal
3	North	Connecticut	Hartford	Robin
4	West	Washington	Olympia	American Goldfinch
5	Midwest	Illinois	Springfield	Cardinal

Example 3: Importing a Tab-Delimited File

Features:

- PROC IMPORT statement options
 - DATAFILE=
 - DATAROW=
 - DBMS=
 - OUT=
 - REPLACE
- DELIMITER= statement
- PRINT procedure

Details

This example imports the following tab-delimited file and creates a temporary SAS data set named `Work.Class`.

Input Data 31.1 *Input*

Name	Gender	Age
Joyce	F	11
Thomas	M	11
Jane	F	12
Louise	F	12
James	M	12
John	M	12
Robert	M	12
Alice	F	13

Barbara	F	13
Jeffery	M	13
Carol	F	14
Judy	F	14
Alfred	M	14
Henry	M	14
Jenet	F	15
Mary	F	15
Ronald	M	15
William	M	15
Philip	M	16

Program

```
proc import datafile='C:\userid\pathname\Class.txt'
    out=class
    dbms=dlm
    replace;

    datarow=5;
    delimiter='09'x;
run;

proc print data=class;
run;
```

Program Description

Specify the input file. The GETNAMES= option defaults to 'yes'. Specify the input file as a delimited file. Replace the data set if it exists. Specify the output data set.

```
proc import datafile='C:\userid\pathname\Class.txt'
    out=class
    dbms=dlm
    replace;
```

The first row read will be row 5 due to the DATAROW= option specification.

```
    datarow=5;
```

Specify the delimiter. On an ASCII platform, the hexadecimal representation of a tab is '09'x. On an EBCDIC platform, the hexadecimal representation of a tab is a '05'x.

```
    delimiter='09'x;
run;
```

Print out the output data set.

```
proc print data=class;
run;
```

Log

The SAS log displays information about the successful import. For this example, the IMPORT procedure generates a SAS DATA step, as shown in the partial log that follows.

Example Code 31.3 Importing a Tab-Delimited File

```

1  proc import datafile='C:\userid\pathname\Class.txt'
2      out=class
3      dbms=dlm
4      replace;
5      datarow=5;
6      delimiter='09'x;
7  run;
8  /*****
9  *   PRODUCT:   SAS
10 *   VERSION:   9.4
11 *   CREATOR:   External File Interface
12 *   DATE:      04DEC18
13 *   DESC:      Generated SAS Datastep Code
14 *   TEMPLATE SOURCE: (None Specified.)
15 *****/
16  data WORK.CLASS ;
17      %let _EFIERR_ = 0; /* set the ERROR detection macro variable */
18      infile 'C:\userid\pathname\Class.txt' delimiter='09'x MISSOVER DSD
19      lrecl=32767
20      ! firstobs=5 ;
21      informat Name $7. ;
22      informat Gender $1. ;
23      informat Age $3. ;
24      format Name $7. ;
25      format Gender $1. ;
26      format Age $3. ;
27      input
28          Name $
29          Gender $
30          Age $
31      ;
32      if _ERROR_ then call symputx('_EFIERR_',1); /* set ERROR detection macro
variable */
33  run;

```

NOTE: The infile 'C:\userid\pathname\Class.txt' is:

Filename=C:\userid\pathname\Class.txt,
 RECFM=V,LRECL=32767,File Size (bytes)=253,
 Last Modified=26Nov2018:14:06:26,
 Create Time=26Nov2018:13:50:27

NOTE: 16 records were read from the infile 'C:\userid\pathname\Class.txt'.

The minimum record length was 9.
 The maximum record length was 12.

NOTE: The data set WORK.CLASS has 16 observations and 3 variables.

NOTE: DATA statement used (Total process time):

real time 0.04 seconds
 cpu time 0.01 seconds

16 rows created in WORK.CLASS from C:\userid\pathname\Class.txt.

NOTE: WORK.CLASS data set was successfully created.

NOTE: The data set WORK.CLASS has 16 observations and 3 variables.

NOTE: PROCEDURE IMPORT used (Total process time):

real time 0.45 seconds
 cpu time 0.20 seconds

Output

Output 31.3 *Work.Class Data Set*

The SAS System			
Obs	Name	Gender	Age
1	Louise	F	12
2	James	M	12
3	John	M	12
4	Robert	M	12
5	Alice	F	13
6	Barbara	F	13
7	Jeffery	M	13
8	Carol	F	14
9	Judy	F	14
10	Alfred	M	14
11	Henry	M	14
12	Jenet	F	15
13	Mary	F	15
14	Ronald	M	15
15	William	M	15
16	Philip	M	16

Example 4: Importing a Comma-Delimited File with a CSV Extension

Features:

- PROC IMPORT statement options
 - DATAFILE=
 - DBMS=
 - GETNAMES=
 - OUT=
 - REPLACE
- PRINT procedure

Details

This example imports the following comma-delimited file and creates a temporary SAS data set named Work.Shoes.

Note: In SAS, the default delimiter for a CSV file is a comma.

```
"Africa", "Boot", "Addis Ababa", "12", "$29,761", "$191,821", "$769"
"Asia", "Boot", "Bangkok", "1", "$1,996", "$9,576", "$80"
"Canada", "Boot", "Calgary", "8", "$17,720", "$63,280", "$472"
"Central America/Caribbean", "Boot", "Kingston", "33", "$102,372", "$393,376", "$4,454"
"Eastern Europe", "Boot", "Budapest", "22", "$74,102", "$317,515", "$3,341"
"Middle East", "Boot", "Al-Khobar", "10", "$15,062", "$44,658", "$765"
"Pacific", "Boot", "Auckland", "12", "$20,141", "$97,919", "$962"
"South America", "Boot", "Bogota", "19", "$15,312", "$35,805", "$1,229"
"United States", "Boot", "Chicago", "16", "$82,483", "$305,061", "$3,735"
"Western Europe", "Boot", "Copenhagen", "2", "$1,663", "$4,657", "$129"
```

Program

```
proc import datafile="C:\temp\test.csv"
    out=shoes
    dbms=csv
    replace;

    getnames=no;
run;

proc print data=work.shoes;
run;
```

Program Description

Specify the input data file. Replace the data set if it exists. Specify the output data set.

```
proc import datafile="C:\temp\test.csv"
    out=shoes
    dbms=csv
    replace;
```

Setting the GETNAMES= option to 'no' causes the variable names in record 1 are not used.

```
    getnames=no;
run;
```

Print the data set.

```
proc print data=work.shoes;  
run;
```

Log

The SAS log displays information about the successful import. For this example, the IMPORT procedure generates a SAS DATA step, as shown in the partial log that follows.

Example Code 31.4 Importing a Comma-Delimited File

```

457 proc import datafile="C:\myfiles\test.csv"
458     dbms=csv
459     out=shoes
460     replace;
461     getnames=no;
462 run;

463 /*****
464 *   PRODUCT:   SAS
465 *   VERSION:   9.4
466 *   CREATOR:   External File Interface
467 *   DATE:      18APR14
468 *   DESC:      Generated SAS Datastep Code
469 *   TEMPLATE SOURCE: (None Specified.)
470 *****/
471 data WORK.SHOES ;
472     %let _EFIERR_ = 0; /* set the ERROR detection macro variable */
473     infile 'C:\myfiles\test.csv' delimiter = ',' MISOVER DSD lrecl=32767 ;
474         informat VAR1 $27. ;
475         informat VAR2 $6. ;
476         informat VAR3 $13. ;
477         informat VAR4 $4. ;
478         informat VAR5 $10. ;
479         informat VAR6 $10. ;
480         informat VAR7 $8. ;
481         format VAR1 $27. ;
482         format VAR2 $6. ;
483         format VAR3 $13. ;
484         format VAR4 $4. ;
485         format VAR5 $10. ;
486         format VAR6 $10. ;
487         format VAR7 $8. ;
488     input
489         VAR1 $
490         VAR2 $
491         VAR3 $
492         VAR4 $
493         VAR5 $
494         VAR6 $
495         VAR7 $
496     ;
497     if _ERROR_ then call symputx('_EFIERR_',1); /* set ERROR detection
497! macro variable */
498     run;

```

NOTE: The infile 'C:\myfiles\test.csv' is:
 Filename=C:\myfiles\test.csv,
 RECFM=V,LRECL=32767,File Size (bytes)=657,
 Last Modified=18Apr2014:10:34:47,
 Create Time=18Apr2014:10:33:23

NOTE: 10 records were read from the infile 'C:\myfiles\test.csv'.
 The minimum record length was 51.
 The maximum record length was 81.

NOTE: The data set WORK.SHOES has 10 observations and 7 variables.

Output

Output 31.4 *Work.Shoes Data Set*

The SAS System							
Obs	VAR1	VAR2	VAR3	VAR4	VAR5	VAR6	VAR7
1	Africa	Boot	Addis Ababa	12	\$29,761	\$191,821	\$769
2	Asia	Boot	Bangkok	1	\$1,996	\$9,576	\$80
3	Canada	Boot	Calgary	8	\$17,720	\$63,280	\$472
4	Central America/Caribbean	Boot	Kingston	33	\$102,372	\$393,376	\$4,454
5	Eastern Europe	Boot	Budapest	22	\$74,102	\$317,515	\$3,341
6	Middle East	Boot	Al-Khobar	10	\$15,062	\$44,658	\$765
7	Pacific	Boot	Auckland	12	\$20,141	\$97,919	\$962
8	South America	Boot	Bogota	19	\$15,312	\$35,805	\$1,229
9	United States	Boot	Chicago	16	\$82,483	\$305,061	\$3,735
10	Western Europe	Boot	Copenhagen	2	\$1,663	\$4,657	\$129

JAVAINFO Procedure

Overview: JAVAINFO Procedure	1121
What Does the JAVAINFO Procedure Do?	1121
Syntax: JAVAINFO Procedure	1121
PROC JAVAINFO Statement	1122

Overview: JAVAINFO Procedure

What Does the JAVAINFO Procedure Do?

The JAVAINFO procedure conveys diagnostic information to the user about the Java environment that SAS is using. The diagnostic information can be used to confirm that the SAS Java environment has been configured correctly, and can be helpful when reporting problems to SAS technical support. Also, PROC JAVAINFO is often used to verify that the SAS Java environment is working correctly because PROC JAVAINFO uses Java to report its diagnostics.

Syntax: JAVAINFO Procedure

PROC JAVAINFO <options>;

Statement	Task
<code>PROC JAVAINFO</code>	Display diagnostic information about the SAS Java environment

PROC JAVAINFO Statement

Displays diagnostic information about the SAS Java environment.

Interaction: By default, PROC JAVAINFO is enabled when the SAS server is in a locked-down state. SAS administrators can disable PROC JAVAINFO by specifying `VIYA_LOCKDOWN_USER_DISABLED_METHODS=JAVA`. For more information, see [“Enable or Disable Access Methods” in SAS Viya Platform: Programming Run-Time Servers](#).

Syntax

PROC JAVAINFO *<options>*;

Optional Arguments

ALL

lists current information about the SAS Java environment.

CLASSPATHS

lists information about the classpaths that Java is using.

HELP

provides usage assistance in using the JAVAINFO procedure.

JREOPTIONS

lists the Java properties that are set when the JREOPTIONS configuration option is specified.

- When used in PROC JAVAINFO, JREOPTIONS specifies the JREOPTIONS Java properties that are set when Java is started.
- When used in PROC OPTIONS, JREOPTIONS specifies the Java options that are in the configuration file when SAS is started.

Note: SAS.cfg is the configuration file specified during installation, but other configuration files can be specified.

OS

lists information about the operating system that SAS is running under.

version

lists the Java Runtime Environment (JRE) that SAS is using.

JSON Procedure

Overview: JSON Procedure	1123
What Does the JSON Procedure Do?	1124
Concepts: JSON Procedure	1124
JSON Output File Containers	1124
Missing Values	1127
Scanning Input Strings	1128
JSON Output File Encoding	1128
Syntax: JSON Procedure	1129
PROC JSON Statement	1129
EXPORT Statement	1134
WRITE VALUES Statement	1138
WRITE OPEN Statement	1140
WRITE CLOSE Statement	1141
Usage: JSON Procedure	1142
Exporting Data	1142
Writing Values to a JSON Output File	1142
Controlling JSON Output with Options	1143
PROC JSON Video	1145
Examples: JSON Procedure	1146
Example 1: Exporting a JSON File Using Default Options	1146
Example 2: Exporting to a Top-Level JSON Array Container	1147
Example 3: Suppressing SAS Variable Names in Observation Data	
When Using the EXPORT Statement	1149
Example 4: Writing JSON Output without Exporting a SAS Data Set	1150
Example 5: Writing Values and Exporting Data in the Same Program	1153
Example 6: Writing Values and Controlling Containers in Exported Data	1156
Example 7: Applying SAS Formats to Exported Output	1160

Overview: JSON Procedure

What Does the JSON Procedure Do?

The JSON procedure reads data from a SAS data set and writes it to an external file in JSON¹ representation. You can control the exported data with several options that remove content and affect the format. In addition to exporting data from a SAS data set, PROC JSON provides statements that enable you to write additional data to the external file and control JSON containers.

SAS provides the JSON engine to read a JSON file. For more information, see [“LIBNAME Statement: JSON Engine” in SAS Global Statements: Reference](#).

Concepts: JSON Procedure

JSON Output File Containers

Structure of the JSON Output File

The JSON output file consists of a minimum of one container, which is referred to as the top-level container. All of the data, metadata, and additional containers are within the top-level container. The type of top-level container is determined by the statement that immediately follows the PROC JSON statement and the options that are enabled in that statement.

1. Java Script Object Notation (JSON) is a text-based, open standard data format that is designed for human-readable data interchange. JSON is based on a subset of the JavaScript programming language and uses JavaScript syntax for describing data objects.

JSON Containers

JSON output consists of two types of data structure containers:

JSON object container ({ })

begins with a left brace ({) and ends with a right brace (}). An object container collects key-value pairs that are written as pairs of names and values. A value can be any of the supported JSON data types, an object, or an array. Each name is followed a colon and then the value. The key-value pairs are separated by a comma.

JSON array container ([])

begins with a left bracket ([) and ends with a right bracket (]). An array container collects a list of values that are written as a list of values without names. A value can be any of the supported JSON data types, an object, or an array. Values are separated by a comma.

Creating Object Containers

The following statements create object containers:

- The EXPORT statement with default options enabled opens an object container as the top-level container. This object container is implicitly opened and automatically closed.

```
proc json out="example.json";
  export sashelp.class (where=(age=11));
  ...
```

- The WRITE VALUES statement opens an object container as the top-level container when it is the first statement after the PROC JSON statement. This object container is implicitly opened and automatically closed.

```
proc json out="example.json";
  write values "container" "object";
  write values "created" "implicitly";
  ...
```

- The WRITE OPEN OBJECT statement explicitly opens an object container, which you must explicitly close with the WRITE CLOSE statement.

```
proc json out="example.json"
  write open object;
  write values "container" "object";
  write values "created" "explicitly";
  write close;
  ...
```

Creating Array Containers

The following statements create array containers:

- The EXPORT statement with the NOSASTAGS option enabled opens a JSON array container as the top-level container. This top-level array container is implicitly opened and automatically closed.

```
proc json out="example.json" pretty;
  export sashelp.class (where=(age=11)) / nosastags;
  ...
```

- The WRITE OPEN ARRAY statement explicitly opens an array container, which you must explicitly close with the WRITE CLOSE statement.

```
proc json out="example.out";
  write open array;
  write values "container" "array";
  write values "created" "explicitly";
  write close;
  ...
```

Nesting Containers

The top-level container can include any number of containers. Containers, likewise, can nest containers to an arbitrary depth. When nesting containers, be careful to observe the data structure requirements of the current container.

- Objects require a list of key-value pairs, where the value can itself be an object or array.
- Arrays have no such structural requirement of key-value pairs and are merely a list of values, objects, or arrays.
- The WRITE VALUES statement or statements for an object container must result in an even number of values, and the key portion of the key-value pair must be a string.

In this example, an array is nested within an implicitly opened object:

```
proc json out="example.json";
  write values "level" 1;           /* implicit open of object */
  write values "container" "object"; /* write data to object */
  write values "created" "implicitly"; /* write data to object */
  write values "nest";             /* required string to start object */
  write open array;                 /* explicit open of array */
  write values "level" 2;           /* write data to array */
  write values "container" "array"; /* write data to array */
  write values "created" "explicitly"; /* write data to array */
  write close;                      /* close explicit open */
run;
```

```
{ "level":1,"container":"object","created":"implicitly","nest":["level",2,
"container","array","created","explicitly"] }
```

In this example, the exported data is nested within three array containers.

```
proc json out="example.json";
  write open array;                                /* explicit open of array */
  write values "level" 1;                          /* write data to array   */
  write values "container" "array";                /* write data to array   */
  write values "created" "explicitly";             /* write data to array   */
  write open array;                                /* explicit open of array */
  write values "level" 2;                          /* write data to array   */
  write values "container" "array";                /* write data to array   */
  write values "created" "explicitly";             /* write data to array   */
  export sashelp.class (where=(age=11)) / nokeys; /* export data without keys*/
  write close;                                     /* close explicit open   */
  write close;                                     /* close explicit open   */
run;
```

```
[ "level",1,"container","array","created","explicitly",["level",2,"container",
"array","created","explicitly","SASJSONExport","1.0 NO",
"SASTableData+CLASS",[[["Joyce","F",11,51.3,50.5],[["Thomas","M",11,57.5,85]]]]
```

Missing Values

A missing value in SAS is a type of value for a variable that contains no data for a particular observation or variable. By default, SAS represents a missing numeric value as a single period and a missing character value as a blank space.

JSON also has the concept of missing values referred to as a null value, which is a special value that indicates the absence of information.

PROC JSON writes a JSON null value to the JSON output file when the following is true:

- a SAS data set numeric variable contains a missing value
- the WRITE VALUES statement writes the NULL keyword as a value

By default, SAS writes an empty string ("") to the JSON output file when a SAS data set character variable contains a missing value. However, if you specify the NOTRIMBLANKS option, the entire string of blanks is written to the JSON output file.

Scanning Input Strings

A JSON string is a sequence of zero or more characters that are wrapped in double quotation marks. A JSON string can be any Unicode character except double quotation marks ("), backslash (\), or a control character. A JSON string can contain a backslash as an escape character, which invokes an alternative interpretation on subsequent characters in a character sequence. For example, to allow a JSON string to contain double quotation marks, the " can be escaped by preceding the character with a \.

By default, PROC JSON scans input strings to ensure that they contain only characters that are acceptable as JSON strings. Unacceptable characters in an input string are replaced with the proper escape sequence.

You can specify the NOSCAN option to indicate that the input string is known to contain acceptable characters or has already been scanned. NOSCANS is supported in the PROC JSON statement, the EXPORT statement, and the WRITE VALUES statement.

JSON Output File Encoding

By default, the resulting JSON output file uses the Unicode encoding UTF-8. To override the encoding of a JSON output file, you can use the ENCODING= option in the FILENAME statement. However, you can do so only in the following situations:

- If the current SAS session encoding is UTF-8, you can specify one of the following Unicode encoding forms. Only UTF-8 and the following encoding forms are JSON standards compliant. Other encoding forms might not be recognized by standards-compliant JSON parsers.

Table 33.1 Encodings That Are Valid in UTF-8 SAS Session

Name	Description
utf-16be	Unicode (UTF-16BE)
utf-32be	Unicode (UTF-32BE)
utf-32le	Unicode (UTF-32LE)

- If the current SAS session encoding is not UTF-8, you can override the encoding of a JSON output file only if the current SAS session encoding is compatible with US-ASCII and all strings written to the JSON output file contain only low-range Latin1 characters (that is, code points 0-127).

For example, the following code exports a JSON output file that uses the Unicode UTF-16BE character set encoding. The current SAS session encoding is Wlatin1. The ENCODING= option in the FILENAME statement tells PROC JSON to transcode the data from Wlatin1 to the specified Unicode UTF-16BE form when writing to the external file.

```
filename jsonout "path-to-JsonOutput.json" encoding="utf-16be";

proc json out=jsonout;
  export sashelp.class;
quit;
```

Syntax: JSON Procedure

```
PROC JSON OUT=fileref | "external-file" <options>;
EXPORT <libref.> SAS-data-set <(SAS-data-set-options)> </options>;
WRITE VALUES value(s) </options>;
WRITE OPEN type;
WRITE CLOSE;
```

Statement	Task	Example
PROC JSON	Exports a SAS data set to a JSON file.	Ex. 1, Ex. 2, Ex. 3, Ex. 4, Ex. 5, Ex. 6, Ex. 7
EXPORT	Identify the SAS data set to export and control the resulting output	Ex. 1, Ex. 2, Ex. 3, Ex. 5, Ex. 6, Ex. 7
WRITE VALUES	Write one or more values to the output file	Ex. 4, Ex. 5, Ex. 6
WRITE OPEN	Open and nest a JSON container in the output file	Ex. 4, Ex. 5, Ex. 6
WRITE CLOSE	Close a JSON container that is open in the output file	Ex. 4, Ex. 5, Ex. 6

PROC JSON Statement

Specifies the JSON output file and controls the resulting output.

Examples:

- [“Example 1: Exporting a JSON File Using Default Options” on page 1146](#)
- [“Example 2: Exporting to a Top-Level JSON Array Container” on page 1147](#)
- [“Example 3: Suppressing SAS Variable Names in Observation Data When Using the EXPORT Statement” on page 1149](#)
- [“Example 4: Writing JSON Output without Exporting a SAS Data Set” on page 1150](#)
- [“Example 5: Writing Values and Exporting Data in the Same Program” on page 1153](#)
- [“Example 6: Writing Values and Controlling Containers in Exported Data” on page 1156](#)
- [“Example 7: Applying SAS Formats to Exported Output” on page 1160](#)

Syntax

PROC JSON *OUT=fileref* | *"external-file"* *<options>* ;

Summary of Optional Arguments

FMTCHARACTER

NOFMTCHARACTER

determines whether to apply a character SAS format to the resulting output if a character SAS format is associated with a SAS data set variable.

FMTDATETIME

NOFMTDATETIME

determines whether to apply a date, time, or datetime SAS format to the resulting output if a date, time, or datetime SAS format is associated with a SAS data set variable.

FMTNUMERIC

NOFMTNUMERIC

determines whether to apply a numeric SAS format to the resulting output if a numeric SAS format is associated with a SAS data set variable.

KEYS

NOKEYS

determines whether exported observations are written as JSON objects or as JSON arrays.

PRETTY

NOPRETTY

determines how to format the JSON output.

SASTAGS

NOSASTAGS

determines whether to include or suppress SAS metadata when using the EXPORT statement.

SCAN

NOSCAN

determines whether PROC JSON scans and encodes input strings to ensure that only characters that are acceptable are exported to the JSON output file.

TRIMBLANKS**NOTRIMBLANKS**

determines whether to remove or retain trailing blanks from the end of character data in the JSON output.

Required Argument

OUT=fileref | "external-file"

identifies the JSON output file.

fileref

specifies the SAS fileref that is assigned to the JSON output file. To assign a fileref, use the FILENAME statement.

"external-file"

is the physical location of the JSON output file. Include the complete pathname and the filename. Enclose the physical name in single or double quotation marks. The maximum length is 200 characters.

Optional Arguments

FMTCHARACTER | NOFMTCHARACTER

determines whether to apply a character SAS format to the resulting output if a character SAS format is associated with a SAS data set variable.

Aliases FMTCHAR

NOFMTCHAR

Default NOFMTCHARACTER

Interaction You can specify FMTCHARACTER | NOFMTCHARACTER in the PROC JSON statement, the EXPORT statement, or both. If the option is specified in both statements, the EXPORT statement specification takes precedence.

Note User-defined formats are also supported.

FMTDATETIME | NOFMTDATETIME

determines whether to apply a date, time, or datetime SAS format to the resulting output if a date, time, or datetime SAS format is associated with a SAS data set variable. Applying a SAS format makes the date and time values in the resulting JSON output more readable.

Aliases FMTDT

NOFMTDT

Default FMTDATETIME

Interaction You can specify `FMTDATETIME` | `NOFMTDATETIME` in the PROC JSON statement, the EXPORT statement, or both. If the option is specified in both statements, the EXPORT statement specification takes precedence.

Tip User-defined formats are also supported.

FMTNUMERIC | NOFMTNUMERIC

determines whether to apply a numeric SAS format to the resulting output if a numeric SAS format is associated with a SAS data set variable.

Aliases FMTNUM

NOFMTNUM

Default NOFMTNUMERIC

Restriction Only the SAS formats `BESTw.`, `Ew.`, and `w.d` write a JSON number to the output file. All other numeric SAS formats result in a JSON string.

Requirement FMTNUMERIC applies numeric SAS formats. For date, time, and datetime SAS formats, use the `FMTDATETIME` option.

Interactions With `NOFMTNUMERIC` or when a numeric variable does not have an associated SAS format, numeric values are written to the output file with a maximum of 12 digits.

You can specify `FMTNUMERIC` | `NOFMTNUMERIC` in the PROC JSON statement, the EXPORT statement, or both. If the option is specified in both statements, the EXPORT statement specification takes precedence.

Note The SAS formats (`BESTw.`, `Ew.`, and `w.d`) write a JSON number to the output file will not apply to the integer values, regardless of the `FMTNUMERIC` option.

Tip User-defined formats are also supported.

KEYS | NOKEYS

determines whether exported observations are written as JSON objects or as JSON arrays. A JSON object stores SAS variable values from observations as key-value pairs. The SAS variable name is the key. A JSON array stores variable values only.

Default KEYS

Interaction You can specify `KEYS` | `NOKEYS` in the PROC JSON statement, the EXPORT statement, or both. If the option is specified in both statements, the EXPORT statement specification takes precedence.

Tip Use `NOKEYS` to suppress SAS variable names in observation data when using the EXPORT statement.

Example [“Example 3: Suppressing SAS Variable Names in Observation Data When Using the EXPORT Statement” on page 1149](#)

PRETTY | NOPRETTY

determines how to format the JSON output. PRETTY creates a more human-readable format that uses indention to illustrate the JSON container structure. NOPRETTY writes the output in a single line.

Default NOPRETTY

Restriction You can specify PRETTY | NOPRETTY in the PROC JSON statement only.

SASTAGS | NOSASTAGS

determines whether to include or suppress SAS metadata when using the EXPORT statement. The metadata consists of the SAS export version, exported SAS data set name, and any non-default option specification, such as PRETTY.

When EXPORT is the first statement in the procedure request, the top-level container for the exported data is a JSON object container. The SAS metadata precedes the exported data. When NOSASTAGS is specified, the top-level container is a JSON array container and the SAS metadata is suppressed.

See the following topics for information about what happens when the EXPORT statement is preceded by a WRITE OPEN statement in the PROC JSON request:

- WRITE OPEN ARRAY — [“Example 5: Writing Values and Exporting Data in the Same Program” on page 1153](#).

Default SASTAGS

Interactions NOSASTAGS affects the outermost container for exported data. The top-level JSON container is affected only when the EXPORT statement is the first statement in the procedure request.

You can specify SASTAGS | NOSASTAGS in the PROC JSON statement, the EXPORT statement, or both. If the option is specified in both statements, the EXPORT statement specification takes precedence.

Note To modify the format of the exported observations, use the KEYS | NOKEYS option.

Examples [“Example 1: Exporting a JSON File Using Default Options” on page 1146](#)

[“Example 2: Exporting to a Top-Level JSON Array Container” on page 1147](#)

[“Example 6: Writing Values and Controlling Containers in Exported Data” on page 1156](#)

SCAN | NOSCAN

determines whether PROC JSON scans and encodes input strings to ensure that only characters that are acceptable are exported to the JSON output file.

Default	SCAN
Interactions	NOSCAN indicates that the input string is known to contain acceptable JSON text or has already been scanned. When NOSCAN is in effect, the input string or character value is taken as is, and the output JSON string is enclosed in quotation marks. You can specify SCAN NOSCAN in the PROC JSON statement, the EXPORT statement, the WRITE VALUES statement, or all three. If the option is specified in multiple statements, the EXPORT statement specification takes precedence.
See	“Scanning Input Strings” on page 1128

TRIMBLANKS | NOTRIMBLANKS

determines whether to remove or retain trailing blanks from the end of character data in the JSON output. Only space characters are removed.

Default	TRIMBLANKS
Interaction	You can specify TRIMBLANKS NOTRIMBLANKS in the PROC JSON statement, the EXPORT statement, the WRITE VALUES statement, or all three. If the option is specified in multiple statements, the EXPORT statement specification takes precedence.

EXPORT Statement

Identifies the SAS data set to be exported and controls the resulting output.

Alias:	EX
Notes:	You can export multiple SAS data sets to the JSON output file by submitting multiple EXPORT statements. The resulting JSON output uses the Unicode encoding form UTF-8. You cannot override the encoding in the output file with the ENCODING= data set option in the EXPORT statement. For information about overriding the encoding, see “JSON Output File Encoding” on page 1128.
See:	“Exporting Data” on page 1142
Examples:	“Example 1: Exporting a JSON File Using Default Options” on page 1146 “Example 2: Exporting to a Top-Level JSON Array Container” on page 1147 “Example 3: Suppressing SAS Variable Names in Observation Data When Using the EXPORT Statement” on page 1149 “Example 5: Writing Values and Exporting Data in the Same Program” on page 1153 “Example 6: Writing Values and Controlling Containers in Exported Data” on page 1156 “Example 7: Applying SAS Formats to Exported Output” on page 1160

Syntax

```
EXPORT <libref.> SAS-data-set < (SAS-data-set-options) > < /options >;
```

Summary of Optional Arguments

(SAS-data-set-options)

specifies SAS data set options that apply to the input SAS data set.

Required Argument

<libref.>SAS-data-set

identifies the SAS data set to be exported with either a one- or two-level SAS name (library and member name). If you specify a one-level name, by default, the JSON procedure uses either the User library (if assigned) or the Work library.

Optional Argument

(SAS-data-set-options)

specifies SAS data set options that apply to the input SAS data set. For example, if the data set that you are exporting has an assigned password, you can use the ALTER=, PW=, READ=, or WRITE= data set options. To export a subset of data that meets a specified condition, you can use the WHERE= option. For information about SAS data set options, see [SAS Data Set Options: Reference](#).

Options to Use after a Slash Delimiter

FMTCHARACTER | NOFMTCHARACTER

determines whether to apply a character SAS format to the resulting output if a character SAS format is associated with a SAS data set variable.

Aliases FMTCHAR

NOFMTCHAR

Default NOFMTCHARACTER

Interaction You can specify FMTCHARACTER | NOFMTCHARACTER in the PROC JSON statement, the EXPORT statement, or both. If the option is specified in both statements, the EXPORT statement specification takes precedence.

FMTDATETIME | NOFMTDATETIME

determines whether to apply a date, time, or datetime SAS format to the resulting output if a date, time, or datetime SAS format is associated with a SAS data set variable. Applying a SAS format makes the date and time values in the resulting JSON output more readable.

Aliases	FMTDT NOFMTDT
Default	FMTDATETIME
Interaction	You can specify FMTDATETIME NOFMTDATETIME in the PROC JSON statement, the EXPORT statement, or both. If the option is specified in both statements, the EXPORT statement specification takes precedence.

FMTNUMERIC | NOFMTNUMERIC

determines whether to apply a numeric SAS format to the resulting output if a numeric SAS format is associated with a SAS data set variable.

Aliases	FMTNUM NOFMTNUM
Default	NOFMTNUMERIC
Restriction	Only the SAS formats BESTw., Ew., and w.d write a JSON number to the output file. All other numeric SAS formats result in a JSON string.
Requirement	FMTNUMERIC applies numeric SAS formats. For date, time, and datetime SAS formats, use the FMTDATETIME option.
Interactions	With NOFMTNUMERIC or when a numeric variable does not have an associated SAS format, numeric values are written to the output file with a maximum of 12 digits. You can specify FMTNUMERIC NOFMTNUMERIC in the PROC JSON statement, the EXPORT statement, or both. If the option is specified in both statements, the EXPORT statement specification takes precedence.

KEYS | NOKEYS

determines whether exported observations are written as JSON objects or as JSON arrays. A JSON object stores SAS variable values from observations as key-value pairs. The variable names are the keys. A JSON array stores variable values only.

Default	KEYS
Interaction	You can specify KEYS NOKEYS in the PROC JSON statement, the EXPORT statement, or both. If the option is specified in both statements, the EXPORT statement specification takes precedence.
Tip	Use NOKEYS to suppress SAS variable names in observation data when using the EXPORT statement.
Example	“Example 3: Suppressing SAS Variable Names in Observation Data When Using the EXPORT Statement” on page 1149

SASTAGS | NOSASTAGS

determines whether to include or suppress SAS metadata when using the EXPORT statement. The metadata consists of the SAS export version, exported SAS data set name, and any non-default option specification, such as PRETTY.

When EXPORT is the first statement in the procedure request, the top-level container for the exported data is a JSON object container. The SAS metadata precedes the exported data. When NOSASTAGS is specified, the top-level container is a JSON array container and the SAS metadata is suppressed.

See the following topics for information about what happens when the EXPORT statement is preceded by a WRITE OPEN statement in the PROC JSON request:

- WRITE OPEN ARRAY — [“Example 5: Writing Values and Exporting Data in the Same Program” on page 1153.](#)

Default SASTAGS

Interactions NOSASTAGS affects the outermost container for exported data. The top-level JSON container is affected only when the EXPORT statement is the first statement in the procedure request.

You can specify SASTAGS | NOSASTAGS in the PROC JSON statement, the EXPORT statement, or both. If the option is specified in both statements, the EXPORT statement specification takes precedence.

Note To modify the format of the exported observations, use the KEYS | NOKEYS option.

Examples [“Example 1: Exporting a JSON File Using Default Options” on page 1146](#)

[“Example 2: Exporting to a Top-Level JSON Array Container” on page 1147](#)

[“Example 6: Writing Values and Controlling Containers in Exported Data” on page 1156](#)

SCAN | NOSCAN

determines whether PROC JSON scans and encodes input strings to ensure that only characters that are acceptable are exported to the JSON output.

Default SCAN

Interactions NOSCAN indicates that the input string is known to contain acceptable JSON text or has already been scanned. When NOSCAN is in effect, the input string or character value is taken as is, and the output JSON string is enclosed in quotation marks.

You can specify SCAN | NOSCAN in the PROC JSON statement, the EXPORT statement, the WRITE VALUES statement, or all three statements. If the option is specified in multiple statements, the EXPORT statement specification takes precedence.

See [“Scanning Input Strings” on page 1128](#)

TABLENAME="name"

specifies a name for the exported SAS data set. The name is exported as SAS metadata in the JSON output file. Enclose the name in single or double quotation marks.

Default The default is the SAS data set member name.

Requirement The TABLENAME= option requires the SASTAGS option to include the SAS metadata in the JSON output.

TRIMBLANKS | NOTRIMBLANKS

determines whether to remove or retain trailing blanks from the end of character data in the JSON output. Only space characters are removed.

Default TRIMBLANKS

Interaction You can specify TRIMBLANKS | NOTRIMBLANKS in the PROC JSON statement, the EXPORT statement, the WRITE VALUES statement, or all three statements. If the option is specified in multiple statements, the EXPORT statement specification takes precedence.

WRITE VALUES Statement

Writes one or more values to the JSON output file.

Alias: W V

Interactions: If the WRITE VALUES statement is the first statement after the PROC JSON statement, PROC JSON opens a JSON object as the top-level container. PROC JSON automatically closes the implicitly opened top-level container. Specifying multiple values in one WRITE VALUES statement is equivalent to submitting multiple WRITE VALUES statements with only one value each. Only the order of the values is significant.

See: [“Writing Values to a JSON Output File” on page 1142](#)

Examples: [“Example 5: Writing Values and Exporting Data in the Same Program” on page 1153](#)
[“Example 6: Writing Values and Controlling Containers in Exported Data” on page 1156](#)

Syntax

WRITE VALUES *value(s)* < /*options* >;

Summary of Optional Arguments

[SCAN](#)

NOSCAN

determines whether PROC JSON scans and encodes input strings to ensure that only characters that are acceptable are exported to the JSON output.

TRIMBLANKS**NOTRIMBLANKS**

determines whether to remove or retain trailing blanks from the end of character data in the JSON output.

Required Argument

value(s)

specifies one or more values to write to the JSON output file. Separate values with a blank space. A value can be one of the following:

- a *string* . If the string is enclosed in quotation marks, there are no restrictions regarding content or length. However, if the string is not enclosed in quotation marks, the following rules apply:
 - The length of the string cannot exceed 256 bytes.
 - The first character must begin with a letter of the Latin alphabet (A–Z, a–z) or the underscore. Subsequent characters can be letters of the Latin alphabet, numerals, or underscores.
 - The string cannot contain blanks or special characters except for the underscore. A string can contain mixed-case letters.

Note: The string can be enclosed in single or double quotation marks.

- a *number* represented in integer, floating point, or exponential format.
- the Boolean value TRUE | T or FALSE | F.
- NULL | N.

For example, the statement `write values "success" true;` causes the following results to be written to the JSON output file:

- **"success": true** if the current container is a JSON object
- **"success", true** if the current container is a JSON array

Requirement When writing values to a JSON object container, the key portion of the key-value pair must be a string. For example, the statement `write values "abcd" "1";` is appropriate. However, the statement `write values 1 "abcd";` produces an error.

Options to Use after a Slash Delimiter

SCAN | NOSCAN

determines whether PROC JSON scans and encodes input strings to ensure that only characters that are acceptable are exported to the JSON output.

Default SCAN

Interactions NOSCANS indicates that the input string is known to contain acceptable JSON text or has already been scanned. When NOSCANS is in effect, the input string or character value is taken as is, and the output JSON string is enclosed in quotation marks.

You can specify SCAN | NOSCANS in the PROC JSON statement, the EXPORT statement, the WRITE VALUES statement, or all three statements. If the option is specified in multiple statements, the EXPORT statement specification takes precedence.

See [“Scanning Input Strings” on page 1128](#)

TRIMBLANKS | NOTRIMBLANKS

determines whether to remove or retain trailing blanks from the end of character data in the JSON output. Only space characters are removed.

Default TRIMBLANKS

Interaction You can specify TRIMBLANKS | NOTRIMBLANKS in the PROC JSON statement, the EXPORT statement, the WRITE VALUES statement, or all three statements. If the option is specified in multiple statements, the EXPORT statement specification takes precedence.

WRITE OPEN Statement

Opens and nests a JSON container in the output file.

Alias: W O

Interactions: If the WRITE OPEN statement is the first statement after the PROC JSON statement, the WRITE OPEN statement establishes the top-level container. Submit the WRITE CLOSE statement for containers that you explicitly open with the WRITE OPEN statement.

See: [“Writing Values to a JSON Output File” on page 1142](#)

Examples: [“Example 5: Writing Values and Exporting Data in the Same Program” on page 1153](#)
[“Example 6: Writing Values and Controlling Containers in Exported Data” on page 1156](#)

Syntax

WRITE OPEN *type*;

Required Argument

type

specifies the type of JSON container:

ARRAY

specifies that the JSON container is an array, which collects a list of values.

Alias A

Example write open array;

OBJECT

specifies that the JSON container is an object, which collects key-value pairs.

Alias O

Example write open object;

WRITE CLOSE Statement

Closes a JSON container that is open in the JSON output file.

Alias: W C

Restriction: The WRITE CLOSE statement cannot be the first statement after the PROC JSON statement.

Interaction: The WRITE CLOSE statement closes the most recently opened container of either type that was explicitly opened with the WRITE OPEN statement. You should submit the WRITE CLOSE statement for containers only if you explicitly opened the container with the WRITE OPEN statement.

See: [“Writing Values to a JSON Output File” on page 1142](#)

Examples: [“Example 5: Writing Values and Exporting Data in the Same Program” on page 1153](#)
[“Example 6: Writing Values and Controlling Containers in Exported Data” on page 1156](#)

Syntax

WRITE CLOSE ;

Usage: JSON Procedure

Exporting Data

The JSON procedure enables you to export one or more SAS data sets to a JSON output file. In the procedure statements, you specify the name of the JSON output file and the name of the SAS data set to be exported. For example, the following PROC JSON code writes the data in a SAS data set named Sashelp.Class to a JSON output file named Output.json.

```
proc json out="path-to-Output.json";  
    export sashelp.class;  
run;
```

The exported data is written to a JSON object container ({ }) by default. Each observation of the data set is also put into a JSON object container. Options are available to enable you to create both the outer container and the observation containers as arrays. The SASTAGS (or NOSASTAGS) argument controls the outer container. The KEYS (or NOKEYS) argument controls the observation containers. For more information, see [“Example 1: Exporting a JSON File Using Default Options” on page 1146](#), [“Example 2: Exporting to a Top-Level JSON Array Container” on page 1147](#), and [“Example 3: Suppressing SAS Variable Names in Observation Data When Using the EXPORT Statement” on page 1149](#).

Writing Values to a JSON Output File

PROC JSON provides the WRITE VALUES statement to explicitly write data to a JSON output file. The WRITE VALUES statement can be used to write one or more values to a JSON object container or a JSON array container. A value can be a string, a number, the Boolean value TRUE or FALSE, or the NULL keyword.

When the WRITE VALUES statement is the first statement specified under the PROC JSON statement, the statement implicitly creates a top-level JSON object container to collect the specified values. In this way, you can use the WRITE VALUES statement in PROC JSON to create your own JSON output file without exporting a SAS data set. SAS variable values are collected in the object container as key-value pairs. For example:

```
proc json out="example.json";  
    write values "container" "object";  
    write values "created" "implicitly";
```

```
run;
```

```
{"container":"object","created":"implicitly"}
```

The WRITE VALUES statement cannot directly add values to a JSON container that contains data from an exported SAS data set. The EXPORT statement implicitly opens and automatically closes a JSON object. You cannot write values to a closed JSON container with the WRITE VALUES statement. To add values to a JSON object that includes exported data, you must precede the WRITE VALUES statement with a WRITE OPEN statement.

The WRITE OPEN statement explicitly opens a JSON container. It also enables you to specify the JSON container type: object or array.

When the WRITE VALUES statement follows a WRITE OPEN OBJECT statement, then the first value in the pair is considered to be a key name. The second value is the key's value. For example:

```
proc json out="example.json"
  write open object;
  write values "container" "object";
  write values "created" "explicitly";
  write close;
run;

{"container":"object","created":"explicitly"}
```

When the WRITE VALUES statement follows a WRITE OPEN ARRAY statement, then the WRITE VALUES statement creates a comma-separated list of values in the JSON array container.

```
proc json out="example.out";
  write open array;
  write values "container" "array";
  write values "created" "explicitly";
  write close;
run;

["container","array","created","explicitly"]
```

Use the WRITE OPEN statement when you want to add information to the JSON file that contains exported SAS data and when you want to nest JSON containers.

Controlling JSON Output with Options

PROC JSON supports several options that control the resulting JSON output. You can specify the options in the PROC JSON, EXPORT, and WRITE VALUES statements. Not all options are supported for all statements.

The following table lists the options and whether they are supported in a statement.

Note: Default option keywords appear in bold.

Table 33.2 Statement Option Availability in the PROC JSON, EXPORT, and WRITE VALUES Statements

Option	Function	PROC JSON Statement	EXPORT Statement	WRITE VALUES Statement
FMTCHARACTER NOFMTCHARACTER	Determines whether to apply an associated character SAS format.	Yes	Yes	No
FMTDATETIME NOFMTDATETIME	Determines whether to apply an associated date, time, or datetime SAS format.	Yes	Yes	No
FMTNUMERIC NOFMTNUMERIC	Determines whether to apply an associated numeric SAS format (excluding date, time, and datetime SAS formats).	Yes	Yes	No
KEYS NOKEYS	Determines whether exported observations are written as JSON objects or as JSON arrays.	Yes	Yes	No
PRETTY NOPRETTY	Determines how to format the JSON output.	Yes	No	No
SASTAGS NOSASTAGS	Determines whether to include or suppress SAS metadata when the EXPORT statement is used. When EXPORT is the first statement in the procedure request, also	Yes	Yes	No

Option	Function	PROC JSON Statement	EXPORT Statement	WRITE VALUES Statement
	determines whether the top-level JSON container is an object container or an array container.			
<i>SAS-data-set-options</i>	Specify actions that apply to the SAS data set.	No	Yes	No
SCAN NOSCAN	Determines whether PROC JSON scans and encodes input strings.	Yes	Yes	Yes
TABLENAME ="name"	Specifies a name in the SAS metadata for the exported SAS data set.	No	Yes	No
TRIMBLANKS NOTRIMBLANKS	Determines whether to remove or retain trailing blanks.	Yes	Yes	Yes

Note: Options specified in the PROC JSON statement apply for the duration of the procedure. Options specified in the EXPORT and WRITE VALUES statement apply only to that statement. If options are specified in more than one statement, the options specified in the EXPORT statement or WRITE VALUES statement take precedence.

PROC JSON Video

For a video that demos how to use PROC JSON, see [How to Write JSON Output from SAS](#). The video shows you how to export a SAS data set to a JSON output file, how to export a subset of data and control the JSON output, and how to write free-form JSON output, control and nest JSON containers, and export multiple SAS data sets.

Examples: JSON Procedure

Example 1: Exporting a JSON File Using Default Options

Features: PROC JSON statement options
PRETTY
EXPORT statement

Details

This PROC JSON example exports a subset of the Sashelp.Class data set to a JSON output file. The WHERE data set option is specified to limit the number of observations that are returned. The PROC JSON PRETTY option is specified to return the output in a format that uses indentation to illustrate the default JSON container structure. Otherwise, default export options are enabled.

Specify the JSON output file and the EXPORT statement. The PROC JSON statement specifies the physical location of the JSON output file with the complete pathname and JSON filename. When the .json extension is omitted, the output is written to a text file.

```
proc json out="path-to-DefaultOutput.json" pretty;  
  export sashelp.class (where=(age=11));  
run;
```


Output: Exporting a JSON File Using Default Options

Output 33.1 Content of PROC JSON Output File DefaultOutput.json

```
{
  1
  "SASJSONExport": "1.0 PRETTY",
  2
  "SASTableData+CLASS": [
    3
    {
      4
      "Name": "Joyce",
      "Sex": "F",
      "Age": 11,
      "Height": 51.3,
      "Weight": 50.5
    },
    {
      "Name": "Thomas",
      "Sex": "M",
      "Age": 11,
      "Height": 57.5,
      "Weight": 85
    }
  ]
}
```

- 1 The top-level container in the output file is a JSON object container ({ }).
- 2 SAS metadata is included in the top of the top-level container. The string "SASJSONExport" specifies the creator of the JSON file. The string "1.0 PRETTY" specifies the version number of the SAS JSON format, followed by any non-default options that are used to create the JSON output file. The string "SASTableData+CLASS" specifies that a SAS data set has been exported and specifies the name of the SAS data set.
- 3 The set of exported observations is in a JSON array container ([]).
- 4 Following the default behavior (KEYS), each observation from the data set is exported to a JSON object container. Variable values in each object container are key-value pairs.

Example 2: Exporting to a Top-Level JSON Array Container

Features:

PROC JSON statement options
 PRETTY
 EXPORT statement options
 NOSASTAGS

Details

This PROC JSON example exports a subset of the Sashelp.Class data set to a JSON output file and specifies to write the exported data to a JSON array container.

Program

```
proc json out="path-to-NosastagsOutput.json" pretty;  
    export sashelp.class (where=(age=11)) / nosastags;  
run;
```

Program Description

Specify the JSON output file. The PROC JSON statement specifies the physical location of the JSON output file with the complete pathname and JSON filename. The PRETTY option creates a more readable format.

```
proc json out="path-to-NosastagsOutput.json" pretty;
```

Identify the SAS data set to be exported and control the top-level JSON container. NOSASTAGS in the EXPORT statement suppresses SAS metadata and specifies that the top-level container is an array container. The NOSASTAGS option can be specified on the PROC JSON statement or on the EXPORT statement.

```
    export sashelp.class (where=(age=11)) / nosastags;  
run;
```

Output: Exporting to a Top-Level JSON Array Container

The top-level container in the output file is an array container ([]). There is no SAS metadata. Each observation in the SAS data set is a JSON object ({ }) container. Variable values within the object containers are key-value pairs.

Output 33.2 Content of PROC JSON Output File NosastagsOutput.json

```
[
  {
    "Name": "Joyce",
    "Sex": "F",
    "Age": 11,
    "Height": 51.3,
    "Weight": 50.5
  },
  {
    "Name": "Thomas",
    "Sex": "M",
    "Age": 11,
    "Height": 57.5,
    "Weight": 85
  }
]
```

Example 3: Suppressing SAS Variable Names in Observation Data When Using the EXPORT Statement

Features:

- PROC JSON statement options
 - PRETTY
- EXPORT statement options
 - NOKEYS

Details

This example exports a subset of the Sashelp.Class data set to a JSON output file and specifies to write the exported data set observations as an array. Array containers store variable values in a value list, instead of as key-value pairs.

Specify the JSON output file. The PROC JSON statement specifies the physical location of the JSON output file with the complete pathname and JSON filename. The PRETTY option creates a more readable format.

```
proc json out="path-to-NokeysOutput.json" pretty;
```

Identify the SAS data set to be exported and control the observation container. NOKEYS in the EXPORT statement writes data set observations as values in an array container instead of to object containers. The NOKEYS option can be specified on the PROC JSON statement or on the EXPORT statement.

```
export sashelp.class (where=(age=11)) / nokeys;
run;
```

Output: Suppressing SAS Variable Names in Observation Data When Using the EXPORT Statement

Output 33.3 Content of PROC JSON Output File NokeysOutput.json

```
{ 1
  "SASJSONExport": "1.0 NOKEYS PRETTY", 2
  "SASTableData+CLASS": [ 3
    [
      "Joyce",
      "F",
      11,
      51.3,
      50.5
    ],
    [
      "Thomas",
      "M",
      11,
      57.5,
      85
    ]
  ]
}
```

- 1 The top-level container in the output file is a JSON object container ({ }).
- 2 SAS metadata is included in the top-level container. The metadata string "SASTableData+CLASS" is the key in a key-value pair. An array containing the exported data is the value in the key-value pair.
- 3 The NOKEYS option overrides the default KEYS behavior; therefore, each selected observation from the SAS data set is exported to a nested array container ([]) in the array. Variable values are listed without variable names.

Example 4: Writing JSON Output without Exporting a SAS Data Set

Features:

- PROC JSON statement options
PRETTY
- WRITE OPEN statement
- WRITE VALUES statement
- WRITE CLOSE statement

Details

This PROC JSON example illustrates how to write JSON output from SAS without exporting data from a SAS data set. This gives you complete control over the content of the JSON output, which enables you to produce arbitrary JSON output.

Program

```
proc json out="path-to-WriteOpenOutput.json" pretty;

  write open object;
  write values "Nested object sample";

  write open object;
  write values "Comment" "In a nested object";
  write close;

  write values "Nested array sample";
  write open array;
    write open array;
      write values "In a nested array";
      write values 1 true null;
    write close;
  write close;

  write values "Finished" "End of samples";

  write close;

run;
```

Program Description

Specify the JSON output file. The PRETTY option controls the resulting output by creating a more readable format.

```
proc json out="path-to-WriteOpenOutput.json" pretty;
```

Open a JSON object container and write a value. The WRITE OPEN OBJECT statement explicitly opens an object container ({}) as the top-level container. The WRITE VALUES statement writes a string to the object container.

```
  write open object;
  write values "Nested object sample";
```

Nest an object container with values, and then close the nested object container. The WRITE OPEN OBJECT statement explicitly opens a second object container in the top-level container. The WRITE VALUES statement writes two strings in the second-level object container. The WRITE CLOSE statement closes the most recently opened container.

```
    write open object;
```

```
write values "Comment" "In a nested object";
write close;
```

Write a value to the JSON output file, open an array container, nest an array container with values, and then close the two array containers. The WRITE VALUES statement writes a string in the top-level container. The WRITE OPEN ARRAY statements create an array container in the top-level container, and then nests a second array container in the previously opened array container. The WRITE VALUES statements write a string, a number, the Boolean value TRUE, and the NULL keyword to the second-level array container. Two WRITE CLOSE statements close the array containers.

```
write values "Nested array sample";
write open array;
  write open array;
    write values "In a nested array";
    write values 1 true null;
  write close;
write close;
```

Write a final value. The WRITE VALUES statement writes two strings to the top-level container.

```
write values "Finished" "End of samples";
```

Close the top-level container. The WRITE CLOSE statement closes the remaining open container, which is the top-level container.

```
write close;
run;
```

Output: Writing JSON Output without Exporting a SAS Data Set

Output 33.4 Content of PROC JSON Output File WriteOpenOutput.json

```
{
  "Nested object sample": { 1
    "Comment": "In a nested object"
  },
  "Nested array sample": [ 2
    [
      "In a nested array",
      1,
      true,
      null
    ]
  ], 3
  "Finished": "End of samples"
}
```

- 1 Because the top-level container was specified as an object container, SAS expects the WRITE VALUES statement to specify key-value pairs. The first

WRITE VALUES statement specifies only one value. Therefore, the string “Nested object sample” is output as the key in a key-value pair. The first nested container is the value of this key-value pair. Because the nested container is also an object container, the nested WRITE VALUES statement is output as a key-value pair. The key in the string is “Comment” and the value is the string “In a nested object”.

- 2 Because the preceding object containers were explicitly closed, the rules from the top-level container apply to the next WRITE VALUES statement. SAS expects another key-value pair. The string “Nested array sample” is the key in this key-value pair; the first nested array container is the value. The second nested array container is an actual array. The nested WRITE VALUES statements define a list of four values in the second-level array container.
- 3 Because the preceding array containers were explicitly closed, the final WRITE VALUES statement again is written to the top-level container and is expected to contain a key-value pair. The string “Finished” is the key in this key-value pair; the string “End of samples” is the value.

Example 5: Writing Values and Exporting Data in the Same Program

Features:

- PROC JSON statement options
- PRETTY
- WRITE OPEN ARRAY statement
- WRITE VALUES statement
- EXPORT statement
- WRITE CLOSE statement

Details

This example writes values and performs three exports of the same data to show the behavior of the default export settings, the NOSASTAGS option, and the NOKEYS option. The exported data is inserted into a top-level array container, to show the effect of NOSASTAGS on the exported data.

Specify the JSON output file in the PROC JSON statement. The PRETTY option creates the output in a more readable format.

```
proc json out="path-to-WriteAndExport.json" pretty;
```

Open a top-level JSON array container and specify values. The WRITE OPEN ARRAY statement explicitly opens a JSON array container ([]) as the top-level container. WRITE VALUES statements specify a string, a numeric value, and additional strings as values for the array.

```

write open array;
  write values "level" 1;
  write values "container" "array";
  write values "created" "explicitly";

```

Specify an EXPORT statement. The EXPORT statement specifies to export a subset of the observations from the SAS data set Sashelp.Class. The WHERE data set option limits the observations that are included in the output file. There are no export options specified. The EXPORT statement implicitly opens and closes the container that holds the exported data. A WRITE VALUES statement before the EXPORT statement specifies a descriptive label.

```

write values "Default Export";
export sashelp.class (where=(age=11));

```

Specify an EXPORT statement that specifies the NOSASTAGS option. The EXPORT statement specifies to export the same data as the preceding EXPORT statement. NOSASTAGS suppresses the SAS metadata. A WRITE VALUES statement before the EXPORT statement specifies a descriptive label.

```

write values "Export with NOSASTAGS Options";
export sashelp.class (where=(age=11)) / nosastags;

```

Specify an EXPORT statement that specifies the NOSASTAGS and NOKEYS options. The EXPORT statement specifies to export the same data as the preceding EXPORT statements. NOSASTAGS suppresses the SAS metadata. The NOKEYS option specifies to export each data set observation to an array container, instead of an object container. A WRITE VALUES statement before the EXPORT statement specifies a descriptive label.

```

write values "Export with NOSASTAGS and NOKEYS Option";
export sashelp.class (where=(age=11)) / nosastags nokeys;

```

Close the outer JSON array container.

```

write close;
run;

```


Output: Writing Values and Exporting Data in the Same Program

Output 33.5 Content of PROC JSON Output File WriteAndExport.json

```
[ 1
  "level",
  1,
  "container",
  "array",
  "created",
  "explicitly",
  "Default Export",
  "SASJSONExport", 2
  "1.0 PRETTY",
  "SASTableData+CLASS",
  [ 3
    {
      "Name": "Joyce",
      "Sex": "F",
      "Age": 11,
      "Height": 51.3,
      "Weight": 50.5
    },
    {
      "Name": "Thomas",
      "Sex": "M",
      "Age": 11,
      "Height": 57.5,
      "Weight": 85
    }
  ],
  "Export with NOSASTAGS Option", 4
  {
    "Name": "Joyce",
    "Sex": "F",
    "Age": 11,
    "Height": 51.3,
    "Weight": 50.5
  },
  {
    "Name": "Thomas",
    "Sex": "M",
    "Age": 11,
    "Height": 57.5,
    "Weight": 85
  },
  "Export with NOSASTAGS and NOKEYS Option", 5
  [
    "Joyce",
    "F",
    11,
    51.3,
    50.5
  ],
  [
    "Thomas",
    "M",
    11,
    57.5,
    85
  ]
]
```

- 1 Values that are defined by the initial WRITE VALUES statements are variable values in the explicitly opened top-level array container.
- 2 SAS metadata from the first EXPORT statement is also added as array values in the top-level array container.
- 3 The default export behavior writes exported data as an array value in the open array container. The exported observations are comma-separated JSON object containers within the new array.
- 4 The string "Export with NOSASTAGS Option" is printed as a label before exported data from the second EXPORT statement. NOSASTAGS in the second EXPORT statement suppresses the SAS metadata and the outer array container from the exported observations. The exported observations are comma-separated object containers in the existing array.
- 5 The string "Export with NOSASTAGS and NOKEYS Option" is printed as a label before the exported data from the third EXPORT statement. When the NOKEYS and NOSASTAGS options are specified together, the exported observations are added to the existing array as comma-separated array values.

Example 6: Writing Values and Controlling Containers in Exported Data

Features:

- PROC JSON statement options
 - NOSASTAGS
 - PRETTY
- WRITE OPEN statement
- WRITE VALUES statement
- EXPORT statement
- WRITE CLOSE statement

Details

This PROC JSON example illustrates how to write additional values to a JSON output file and control and nest JSON containers. The example exports a subset of the Sashelp.Cars data set to the JSON output file.

Program

```
%let vehicleType=Truck;  
%let minCost=26000;
```

```

proc json out="path-to-WriteOpenArrayOutput.json" nosastags pretty;

  write open array;
  write values "Vehicles";
  write open array;
  write values "&vehicleType";
  write open array;
  write values "Greater than $&minCost";
  /***** Asian *****/
  %let originator=Asia;
  write open object;
  write values "&originator";
  write open array;
  export sashelp.cars(where=((origin = "&originator") and
                           (type = "&vehicleType") and
                           (MSRP > &minCost)
                           )
                    keep=make model type origin MSRP);
  write close; /* data values */
  write close; /* Asia */
  /***** European *****/
  %let originator=Europe;
  write open object;
  write values "&originator";
  write open array;
  export sashelp.cars(where=((origin = "&originator") and
                           (type = "&vehicleType") and
                           (MSRP > &minCost)
                           )
                    keep=make model type origin MSRP);
  write close; /* data values */
  write close; /* Europe */
  /***** American *****/
  %let originator=USA;
  write open object;
  write values "&originator";
  write open array;
  export sashelp.cars(where=((origin = "&originator") and
                           (type = "&vehicleType") and
                           (MSRP > &minCost)
                           )
                    keep=make model type origin MSRP);
  write close; /* data values */
  write close; /* USA */
  write close; /* expensive */
  write close; /* vehicleType */
  write close; /* cars */
run;

```

Program Description

Assign macro variables. The %LET statements create macro variables and assign values to be used throughout the code.

```

%let vehicleType=Truck;
%let minCost=26000;

```

Specify the JSON output file and control the resulting output. The PROC JSON statement specifies the physical location of the JSON output file with the complete pathname and JSON filename. The NOSASTAGS option suppresses creation of SAS metadata and the PRETTY option creates a more readable format.

```
proc json out="path-to-WriteOpenArrayOutput.json" nosastags pretty;
```

Include additional PROC JSON statements and submit the request. These statements open a series of nested containers for each type of car and write values as labels for the containers. The EXPORT statement specifies the SAS data set to export, selects specific observations, and requests only certain SAS variables.

```

write open array;
write values "Vehicles";
write open array;
write values "&vehicleType";
write open array;
write values "Greater than $&minCost";
/***** Asian *****/
%let originator=Asia;
write open object;
write values "&originator";
write open array;
export sashelp.cars(where=((origin = "&originator") and
                        (type = "&vehicleType") and
                        (MSRP > &minCost)
                        )
            keep=make model type origin MSRP);
write close; /* data values */
write close; /* Asia */
/***** European *****/
%let originator=Europe;
write open object;
write values "&originator";
write open array;
export sashelp.cars(where=((origin = "&originator") and
                        (type = "&vehicleType") and
                        (MSRP > &minCost)
                        )
            keep=make model type origin MSRP);
write close; /* data values */
write close; /* Europe */
/***** American *****/
%let originator=USA;
write open object;
write values "&originator";
write open array;
export sashelp.cars(where=((origin = "&originator") and
                        (type = "&vehicleType") and
                        (MSRP > &minCost)
                        )
            keep=make model type origin MSRP);
write close; /* data values */
write close; /* USA */
write close; /* expensive */
write close; /* vehicleType */
write close; /* cars */
run;
```

Output: Controlling JSON Containers and Writing Values

Output 33.6 Content of PROC JSON Output File WriteOpenArrayOutput.json

```
[
  "Vehicles",
  [
    "Truck",
    [
      "Greater than $26000",
      {
        "Asia": [
          {
            "Make": "Nissan",
            "Model": " Titan King Cab XE",
            "Type": "Truck",
            "Origin": "Asia",
            "MSRP": 26650
          }
        ]
      },
      {
        "Europe": [
        ]
      },
      {
        "USA": [
          {
            "Make": "Cadillac",
            "Model": " Escalade EXT",
            "Type": "Truck",
            "Origin": "USA",
            "MSRP": 52975
          },
          {
            "Make": "Chevrolet",
            "Model": " Avalanche 1500",
            "Type": "Truck",
            "Origin": "USA",
            "MSRP": 36100
          },
          {
            "Make": "Chevrolet",
            "Model": " Silverado SS",
            "Type": "Truck",
            "Origin": "USA",
            "MSRP": 40340
          },
          {
            "Make": "Chevrolet",
            "Model": " SSR",
            "Type": "Truck",
            "Origin": "USA",
            "MSRP": 41995
          }
        ]
      }
    ]
  ]
]
```

```

    {
      "Make": "Ford",
      "Model": " F-150 Supercab Lariat",
      "Type": "Truck",
      "Origin": "USA",
      "MSRP": 33540
    },
    {
      "Make": "GMC",
      "Model": " Sierra HD 2500",
      "Type": "Truck",
      "Origin": "USA",
      "MSRP": 29322
    }
  ]
}
]

```

Example 7: Applying SAS Formats to Exported Output

Features:

- PROC JSON statement options
 - PRETTY
- EXPORT statement options
 - FMTDATETIME
 - FMTNUMERIC
- DATA step

Details

This PROC JSON example exports a SAS data set named Work.Formats that contains variables with associated SAS formats. The resulting JSON output file applies the SAS formats, which makes the output values more readable.

Program

```

data
formats;

  input name $ idnumber $ salary hiredate
mmddy10.;

```

```

        format salary dollar7. hiredate
date9.;

datalines;

Brad 0755 21163
9/24/2012

Lindzey 0767 34321
9/04/2012

;

proc json out="path-to-FormatsOutput.json" pretty;
    export work.formats / fmtnumeric;
run;

```

Program Description

Create a SAS data set with variables that have associated SAS formats. The DATA step creates a SAS data set named Work.Formats with four variables. The FORMAT statement associates the DOLLAR7. SAS numeric format with the variable Salary and the SAS date format DATE9. with the variable HireDate.

```

data
formats;

    input name $ idnumber $ salary hiredate
mmdyy10.;

    format salary dollar7. hiredate
date9.;

datalines;

Brad 0755 21163
9/24/2012

Lindzey 0767 34321
9/04/2012

;

```

Specify the JSON output file. The PROC JSON statement specifies the physical location of the JSON output file with the complete pathname and filename and includes the PRETTY option to create a more readable format.

```
proc json out="path-to-FormatsOutput.json" pretty;
```

Identify the SAS data set to be exported and specify the FMTNUMERIC option. The EXPORT statement specifies the SAS data set name. The FMTDATETIME option is available by default to apply the date SAS format that is associated with

the HireDate variable. The FMTNUMERIC option is specified to apply the numeric SAS format DOLLAR7. that is associated with the Salary variable.

```
export work.formats / fmtnumeric;
run;
```

Output: Applying SAS Formats to Exported Output

Output 33.7 PROC JSON Output File *FormatsOutput.json* with FMTDATETIME and FMTNUMERIC

```
{
  "SASJSONExport": "1.0 PRETTY FMTNUMERIC",
  "SASTableData+FORMATS": [
    {
      "name": "Brad",
      "idnumber": "0755",
      "salary": "$21,163",
      "hiredate": "24SEP2012"
    },
    {
      "name": "Lindzey",
      "idnumber": "0767",
      "salary": "$34,321",
      "hiredate": "04SEP2012"
    }
  ]
}
```

If the EXPORT statement specified the option NOFMTDATETIME and did not specify the option FMTNUMERIC, the resulting JSON output file would include Salary and HireDate values that are less readable.

Output 33.8 PROC JSON Output File *FormatsOutput.json* with NOFMTDATETIME and NOFMTNUMERIC

```
{
  "SASJSONExport": "1.0 PRETTY NOFMTDATETIME",
  "SASTableData+FORMATS": [
    {
      "name": "Brad",
      "idnumber": "0755",
      "salary": 21163,
      "hiredate": 19260
    },
    {
      "name": "Lindzey",
      "idnumber": "0767",
      "salary": 34321,
      "hiredate": 19240
    }
  ]
}
```


LUA Procedure

Overview: LUA Procedure	1163
What Does the LUA Procedure Do?	1164
Concepts: LUA Procedure	1165
Verifying the Security of Lua Scripts	1165
Running External Lua Files	1166
Calling CAS Actions with PROC LUA	1166
Calling Standard SAS Functions within Lua Statements	1168
Submitting SAS Code within Lua Statements	1168
Scope for PROC LUA	1169
How PROC LUA Interprets Math.Huge	1170
How Lua Interprets Boolean Values	1170
Calling Lua Functions	1171
Data Set Functions for the LUA Procedure	1171
Processing Data Set Observations	1174
System Functions for the LUA Procedure	1175
About Lua Libraries	1176
Table Functions for PROC LUA	1176
Functions That Manipulate Strings	1177
Object Syntax within Lua Statements	1178
Calling PROC FCMP Functions within Lua Statements	1179
Using the FULLSTIMER Option with PROC LUA	1179
Syntax: LUA Procedure	1180
PROC LUA Statement	1181
SUBMIT Statement	1183
ENDSUBMIT Statement	1184
Usage: LUA Procedure	1184
Submitting Lua Statements within a SAS Program	1184
Opening a SAS Data Set within Lua Code	1185
Examples: LUA Procedure	1186
Example 1: Create a Sample Data Set	1186
Example 2: Running Lua Scripts with SECURELUA Enabled	1187
Example 3: Specifying Input from an External Lua Script	1191
Example 4: Loading a SAS Data Set and Viewing the Resulting Lua Table	1192
Example 5: Writing a SAS Data Set from a Lua Table	1194
Example 6: Using Table Functions	1198
Example 7: Using String Functions	1200
Example 8: Adding an Observation to a Lua Table	1203

Example 9: Using SAS Macro Variable Values within Lua Statements	1207
Example 10: Submitting SAS Code with Lua Variable Substitutions	1208
Example 11: Using an Iterator Function for a Small Table	1211
Example 12: Using Iterator Functions for a Large Table	1213
Example 13: Defining and Adding Variables to a Data Set	1216
Example 14: Connecting to the CAS Server	1217
Example 15: Running PROC FCMP Functions	1222

Overview: LUA Procedure

What Does the LUA Procedure Do?

The Lua programming language is an embeddable scripting language that runs on any platform that has a standard C compiler. Lua uses simple syntax, runs fast computations, and automatically manages memory allocation.

The LUA procedure enables you to run statements from the Lua programming language within SAS code. You can submit Lua statements from an external Lua script, or enter Lua statements directly in SAS code.

PROC LUA enables you to perform these tasks:

- run Lua code within a SAS session
- call most SAS functions within Lua statements
- call PROC FCMP functions within Lua statements
- submit SAS code from Lua
- call CAS actions
- read VARCHAR data

Note: The SUBMIT/ENDSUBMIT statement within PROC LUA is not allowed inside a macro definition. Instead, you can use an %INCLUDE statement or the INFILE= argument to run LUA code from the macro definition.

Concepts: LUA Procedure

Verifying the Security of Lua Scripts

Because of security vulnerabilities in the Lua run time, only trusted Lua scripts can be executed. Trust is established by creating digital certificates. When the SECURELUA system option is set to TRUE, PROC LUA does not execute a Lua script or load a Lua module unless it is trusted. The default value for the SECURELUA system option is FALSE.

Lua scripts can be executed in the following ways:

- A simple Lua script that is not compiled can be invoked using the INFILE option on the Lua statement, or the LOAD or LOADFILE functions in a Lua script.
- A Lua module that is not implemented in C code can be invoked by the REQUIRE function in a Lua script.

In these circumstances, the directory in which the Lua script or module resides must be specified in the LOCKDOWN allowlist and the SECURELUALOC system option.

Note: If the Lua script is compiled as a binary or if the Lua module is written in C, the Lua code cannot be executed when the SECURELUA system option is turned on.

If the Lua script is executed from the PROC LUA SUBMIT statement, a digital signature must be specified on the SECURELUATOKEN argument for the PROC LUA statement.

The administrator is responsible for verifying the safety of user-written Lua scripts or modules.

- Directory paths specified via SECURELUALOC must be writeable only by a trusted security administrator at the customer site.
- User-written scripts and modules should undergo a security review before being stored within a SECURELUALOC path.
- The administrator is responsible for supplying site-specific configuration options to add the directory paths to the LOCKDOWN allowlist and SECURELUALOC option.

The location of the certificate used to validate the Lua content must be specified by the LUACERTLOC environment variable. For information about creating digital

certificates, see “[Generate Digital Certificates Using OpenSSL](#)” in *Encryption in SAS*.

See “[Example 2: Running Lua Scripts with SECURELUA Enabled](#)” on page 1187 for a description of how to create and run Lua scripts when SECURELUA is enabled.

Running External Lua Files

You can run external Lua scripts (*.lua or *.luc files) by using the %INCLUDE statement in a SAS session. For example, to run the Lua script abc.luc, enter the following line in your SAS program:

```
%include "./tmp/abc.luc";
```

Calling CAS Actions with PROC LUA

Requirements to Connect to the CAS Server

You can call CAS actions from PROC LUA. You need to know the following information to connect to the CAS server:

- the name of the server (host) that is running the CAS server
- the CAS server port
- your user ID and password, which must have permission to access the CAS server. Alternatively, you must have created an authinfo file.

The following resources must also be in place. These files are typically installed automatically as part of your installation:

- middleclass.lua
- swat.lua
- tkluaswat.so

For more information, see “[Requirements](#)” in *SAS Viya Platform Programming: Getting Started with Lua for CAS*.

To connect to the CAS server and run Lua code, you must load the appropriate settings and utilities in your SAS program. You must also load the SAS Scripting Wrapper for Analytics Transfer (SWAT) library. For more information, see “[Example 14: Connecting to the CAS Server](#)” on page 1217.

Functions That Interact with the CAS Server

Here are the functions that interact with the CAS server. For an example that shows how to connect to the CAS server, see [“Example 14: Connecting to the CAS Server” on page 1217](#).

Note: It is a SAS convention to document function names in all capitals. However, when you call these functions within the LUA procedure, you must use all lower-case letters.

*Lua-var:*HELP{}

requests the list of available CAS actions for the CAS session that is associated with the Lua variable. For example, you might specify `s:help{}` if you have assigned your CAS session to the Lua variable `s`.

CAS.OPEN(*"host-name"*, *port*<, *"user-ID"*><, *"password"*>)

opens a session on the CAS server. You provide the host name and port for the CAS server, and optionally, you supply your user ID and password values. Your user ID must have permission to access the CAS server. If you do not include your user ID and password, then you must have defined an authinfo file. For more information, see [“Requirements” in SAS Viya Platform Programming: Getting Started with Lua for CAS](#).

Assign the result of CAS.OPEN to a Lua variable that represents the CAS session, as shown in the following alternative versions:

```
s = cas.open("host.mycompany.com", 5570, "myuserid", "password")
```

```
s = cas.open("host.mycompany.com", 5570)
```

After you establish the connection, you can refer to the attributes of `s`, such as `s.port` or `s.hostname`.

*Lua-var:*SHUTDOWN{}

ends the CAS session that is associated with the Lua variable and closes the connection to the CAS server. For example, to close the CAS session represented by the Lua variable `s`, use this command:

```
s:shutdown{}
```

Support for VARCHAR Data with PROC LUA

VARCHAR data is supported on the CAS server only. This means that DATA steps that run entirely on the CAS server and any actions that support VARCHAR data can use VARCHAR values. For other tasks, VARCHAR data is first converted to fixed-length character data.

For more information, see [“Determining Where the DATA Step Is Running” in SAS Cloud Analytic Services: DATA Step Programming](#) and the documentation for an action, procedure, or function to ensure that VARCHAR data is supported.

Calling Standard SAS Functions within Lua Statements

You can run most standard SAS functions within Lua statements by preceding the function call with a “SAS.” prefix. For example, to call the SAS function MAX, precede it with “SAS.” as shown here:

```
local max = sas.max(a,b,c)
```

Note: Functions that run only in the DATA step, such as ADDRLONG, do not run in Lua code.

If you do not supply a value for a required argument in a SAS function, then the SAS function converts that argument to a missing value.

Submitting SAS Code within Lua Statements

Functions That Submit SAS Code

Because Lua is a scripting language within SAS, you can submit SAS code and substitute SAS values within Lua statements. You submit SAS code by using the following functions:

Note: It is a SAS convention to document function names in all capitals. However, when you call these functions within the LUA procedure, you must use all lower-case letters.

SAS.SUBMIT(*[[SAS code]]*, {*substitution(s)*})

submits and executes the specified SAS code when the function is issued within Lua statements. Enclose SAS code within *[[* and *]]* brackets.

When substitutions are provided, they are assigned first. Any remaining substitutions are supplied by local variables.

If no substitution values are submitted, then local variables from the calling function environment are used. Enclose variables for substitution within @ symbols. For example, to assign the value of a variable called userID to a variable user, issue the following assignment:

```
user = @userID@
```

Here is an example that shows the use of a substitution with the call to SAS.SUBMIT.

```
sas.submit ([ [
    data @name@;
    x = @x@;
run;
]], {name="foo"})
```

When you submit SAS code, do not include these keywords within SAS comments:

```
run;    %macro
quit;   %mend
```

The presence of these keywords within comments in a SAS code block might trigger warnings or errors in the SAS log and might cause unexpected results. To avoid this situation, modify the keywords by placing blank spaces or escape characters within them. For example, `run ;` or `\%\macro` within a comment would not trigger unexpected behavior.

The return value from SAS.SUBMIT is assigned to the value of the SYSERR automatic macro variable.

SAS.SUBMIT_(SAS code, substitution(s))

submits SAS code and substitution values but does not execute the code. You must issue the SAS.SUBMIT function (no underscore) to run all submitted SAS commands and end preceding calls to SAS.SUBMIT_.

If no substitution values are submitted, then local variables from the calling function environment are used. Enclose variables for substitution within @ symbols.

For more information, see [“Opening a SAS Data Set within Lua Code” on page 1185](#).

About Lua Variable Substitution within SAS Statements

Variable substitutions are made via a hash table that contains key-value pairs. The key-value pairs are defined by the Lua variable assignments that are made within the submitted Lua statements. To substitute a value, enclose a key within @ symbols, such as @key@. The value of @key@ is replaced with its corresponding value. All substitutions are case sensitive, so @key@ and @Key@ are different key values.

Scope for PROC LUA

Any filerefs, librefs, and local variables that are defined within a call to PROC LUA are defined only for the duration of that procedure call. Similarly, macro variables and macro definitions that are created within a PROC LUA call can be used only

within that procedure call. You cannot refer to these objects outside of the LUA procedure.

How PROC LUA Interprets Math.Huge

In Lua, the constant `math.huge` is a computer representation of infinity. SAS represents this value with the constant value `1.7976931348623E308`.

How Lua Interprets Boolean Values

Lua has two data types that are used in the Boolean context. There is a Nil type, which can take the single value of `nil`. There is a Boolean type, which can take values `true` or `false`. In Lua, the values `nil` or `false` are considered to be false. All other values, including 0, are considered to be true.

In SAS, there are many numeric functions that return a value of 1 if a condition is true and 0 if that condition is false. Examples of these functions include `SYMEXIST`, `SYMLOCAL`, `SYMGLOBAL`, and `MISSING`. Other SAS functions, such as the `ANY*` character functions, search a string for given characters and return the position of the first occurrence in the string. These functions return 0 if the characters are not found.

If you call any of these SAS functions in your Lua code, make sure that your Lua code interprets the results as intended.

For example, suppose that the macro variable `FOOBAR` is not defined. Therefore, it does not exist in a local symbol table. The following code incorrectly states that the variable does exist and is local.

```
/* INCORRECT use of SAS Boolean functions */
proc lua;
  submit;
    if sas.symexist("foobar") then
      if sas.symlocal("foobar") then
        print("In Proc LUA, foobar exists and is LOCAL.")
      else
        print("In Proc LUA, foobar exists but is not LOCAL.")
      end
    else
      print("In Proc LUA, foobar does not exist.")
    end
  endsubmit;
run;
```

The preceding code incorrectly states that `FOOBAR` exists and is local, because the 0 value that is returned by the SAS functions `SYMEXIST` and `SYMLOCAL` are interpreted as true in Lua.

Instead, when you call SAS Boolean functions, explicitly test for the desired return value in your Lua code. The following code correctly tests to see whether the macro variable FOOBAR exists and is local.

```
/* CORRECT use of SAS Boolean functions */
proc lua;
  submit;
    if sas.symexist("foobar") == 1 then
      if sas.symlocal("foobar") == 1 then
        print("In Proc LUA, foobar exists and is LOCAL.")
      else
        print("In Proc LUA, foobar exists but is not LOCAL.")
      end
    else
      print("In Proc LUA, foobar does not exist.")
    end
  endsubmit;
run;
```

The preceding code correctly states that the macro variable FOOBAR does not exist and is not local, because it was never defined. The return value of 0 from the SAS function SYMEXIST is now explicitly compared to the value 1.

Calling Lua Functions

When you run PROC LUA, you can call most Lua functions in your Lua statements. For more information, see [your Lua documentation](#). Check the SAS log to ensure that all the functions you call run without error.

Data Set Functions for the LUA Procedure

The following functions have been defined to run in the LUA procedure. These functions interact with SAS or with data sets.

For small data sets, you can read an entire SAS data set into a Lua table. You can then modify or query the Lua table, and you can write changes to a new SAS data set. However, for large data sets, it is more efficient to process the data one observation at a time, as you would with a DATA step. For this reason, you should submit a DATA step within your Lua code for large data sets.

Here are the data set functions that run within the LUA procedure:

Note: It is a SAS convention to document function names in all capitals. However, when you call these functions within the LUA procedure, you must use all lower-case letters.

SAS.ADD_VARS(*data-set-ID*, *variable-definitions*)

adds the specified Lua variables to a new data set. That is, use SAS.ADD_VARS only when you open a data set using SAS.OPEN with mode 'o'.

The format for defining variables is as follows:

```
{ {name="varname", type="N|C",
  format="format.", length="value",
  label="varlabel", informat="informat."},
  {additional variable definition}, ...
}
```

Only the variable name is required. The default variable type is numeric (N) if the Lua variable with the same name is also numeric. The default length of character variables is 200.

You can specify full SAS formats, such as `format="best12.3"`. If you supply a format that includes a period (.), then the system expects that the format is complete and additional format attributes are ignored. If you do not supply a full SAS format, you can use a combination of these variable attributes to provide the format of a variable:

- `FORMAT` specifies no length nor decimal specification (for example, `format="best"`)
- `FORMAT_WIDTH` specifies the number of characters or digits (for example, `format_width=12`)
- `FORMAT_DEC` specifies the number of decimal places (for example, `format_dec=3`)

For more information, see [“Example 13: Defining and Adding Variables to a Data Set” on page 1216](#).

SAS.ATTR(*data-set-ID*, *attribute-name*)

returns the value of *attribute-name* for a data set. For example, `sas.attr(data-set-ID, 'label')` returns the label for the specified data set.

SAS.CLOSE(*data-set-ID*)

closes an open data set. This function exits if the data set ID is not valid.

SAS.EXISTS(*SAS-data-set-name*)

returns the Boolean value **true** if the data set exists or **false** if the data set does not exist.

Note: This function is different from the standard SAS function EXIST (with no 's' as the end), which returns a 0 or 1 value. In Lua code, 0 and 1 are interpreted as true. As an alternative, test whether the condition `(sas.exist("work.test") > 0)` is true. The SAS.EXISTS function is available only within Lua code blocks.

SAS.NOBS(*data-set-ID*)

returns the number of observations in a data set.

SAS.NVARS(*data-set-ID*)

returns the number of variables in a data set.

SAS.OPEN(SAS-data-set-name<, mode>)

opens a SAS data set and returns a data set ID if the data set opens successfully. If the data set does not open, then the function returns `nil`. Therefore, use the SAS.EXISTS function before calling the SAS.OPEN function.

You can use data set options, such as KEEP=, DROP=, or WHERE= with the SAS.OPEN function. For example, you can open the data set Sashelp.Class and keep only the variable Age with the following code:

```
local dsid = sas.open('sashelp.class(keep=age)')
```

Valid data set modes are *I* (for reading), *O* (for creating), and *U* (for updating). If you do not supply a value for data set mode, then the default value of *I* is used.

SAS.READ_DS(SAS-data-set-name)

returns a Lua table that contains the data from the specified SAS data set. If the data set does not exist, the function returns `nil`. Therefore, use the SAS.EXISTS function before calling the SAS.READ_DS function.

As a best practice, use the SAS.READ_DS function for small data sets only. For large data sets, iterate over observations with the functions that process individual observations. For more information, see [“Functions That Process Observations” on page 1174](#).

Alias: SAS.LOAD_DS(SAS-data-set-name)

SAS.SET_ATTR(data-set-ID, attribute-name, value)

assigns a value to an attribute of a data set.

SAS.WHERE(data-set-ID, where-clause)

applies a WHERE clause to a data set. If there is a previously existing WHERE clause for the data set, then the specified WHERE clause is added to the previous one. However, if the Boolean value for the optional *replace-where-clause* argument is true, then the specified WHERE clause replaces a previously existing WHERE clause.

The return code from this function is an integer value. Because Lua interprets all integer values as true, explicitly test the return value to see whether it is equal to 0 (false in SAS).

Operations such as `sas.where(dsid, "also <new condition>")`, `sas.where(dsid, "undo")`, and `sas.where(dsid, "clear")` are supported. For more information, see *SAS Component Language: Reference*.

You can submit Lua statements similar to the following to apply a WHERE clause to a data set:

```
sas.submit("data work.air; set sashelp.air; run;")
dsid = sas.open("work.air")
rc, msg = sas.where(dsid, "air=222 or air=999")
print(rc, msg)
rc=sas.close(dsid)
```

SAS.WRITE_DS(Lua-table, SAS-data-set-name)

creates a SAS data set from a Lua table. You can specify a two-level name for the SAS data set, such as Work.Random. The Lua table must conform to the same structure that is returned by the SAS.READ_DS function. There must be at least one observation in the source table.

Processing Data Set Observations

Functions That Process Observations

You can use the following Lua functions to process individual observations within a data set. You must first open the data set in Update mode ('u').

Note: It is a SAS convention to document function names in all capitals. However, when you call these functions within the LUA procedure, you must use all lower-case letters.

SAS.APPEND(*data-set-ID*)

appends a newly created observation to a data set.

SAS.DELOBS(*data-set-ID*)

deletes the current observation in a data set.

SAS.GET_VALUE(*data-set-ID*, *variable-number* | *variable-name*)

returns the value of the specified variable in the current observation. Identify the variable by its position (number) in the data set or by its name.

SAS.NEXT(*data-set-ID*)

moves to the next observation in a data set for processing. If you have not yet begun processing a data set, the SAS.NEXT function moves to the first observation in that data set. The SAS.NEXT function enables you to work directly with a SAS data set without needing to first read the data into a Lua table. This function is useful for large data sets.

SAS.PUT_VALUE(*data-set-ID*, *variable-name*, *value*)

loads a value into the specified variable in a data set.

SAS.ROWS(*data-set-ID*)

iterates over the observations in a data set, loads each row into a Lua table for processing, and adds a Lua nil at the end of processing. This function is useful for data sets with relatively few variables.

SAS.UPDATE(*data-set-ID*)

updates an observation with values that were added by calling the SAS.PUT_VALUE function.

SAS.VARS(*data-set-ID*)

iterates over the variables in a data set.

Sequence to Add an Observation

To add a new observation, call the observation-processing functions in the following order:

- 1 SAS.APPEND.
- 2 SAS.PUT_VALUE. Repeat this function call until values are set for all desired variables in an observation.
- 3 SAS.UPDATE.

Out of Scope Warning

If a data set ID (handle) goes out of scope, then the associated SAS data set is automatically closed if it has not been closed already. A warning similar to the following appears in the SAS log:

```
WARNING: Closing SASHELP.CLASS - handle has gone out of scope.
```

A data set ID goes out of scope when the program is no longer able to access it. For example, when a data set ID is defined as a local variable in a user-defined function, the data set ID becomes out of scope at the end of that function definition. As a best practice, close a data set before the data set ID goes out of scope.

System Functions for the LUA Procedure

The following functions control the behavior of SAS code that is submitted from within the LUA procedure. For more information, see [“Submitting SAS Code within Lua Statements” on page 1168](#).

Note: It is a SAS convention to document function names in all capitals. However, when you call these functions within the LUA procedure, you must use all lower-case letters.

SAS.GET_MAX_SYSERR()

returns the current maximum SYSERR value that can be returned from calls to SAS.SUBMIT without triggering an error in the SAS log.

SAS.IS_QUIET()

returns `true` or `false`, depending on the value that was set using the SAS.SET_QUIET function. This Boolean value indicates whether SAS language statements that are submitted to the SAS.SUBMIT function are written to the SAS log.

`SAS.SET_MAX_SYSERR(value)`

specifies the maximum allowable SYSERR value that can be returned by `SAS.SUBMIT` calls. If SAS returns a SYSERR value greater than the specified value, an error is printed in the SAS log. The default is zero, and possible values include positive integers.

`SAS.SET_QUIET(value)`

specifies whether SAS language statements that are submitted to the `SAS.SUBMIT` function are written to the SAS log.

Possible argument values are `true` or `false`. The default value is `false`.

About Lua Libraries

Lua Libraries and SAS Extensions to Lua

The Lua programming language contains several libraries that can be used to manipulate tables and strings and to perform common math functions. Most of these functions are available to the LUA procedure. In addition, there are functions that were created for PROC LUA as an extension of the Lua language. These functions enable you to work with tables and strings and to perform calculations on your data. These extension functions are available within PROC LUA, but they are not part of the Lua programming language and cannot be used outside of PROC LUA.

Restrictions to the Lua IO Library

When SAS is running in lockdown mode (in a locked-down state), access to functions in the Lua IO library is restricted. It is possible for a SAS administrator to disable this restriction. For more information, see [“Security Features” in SAS Viya Platform: Programming Run-Time Servers](#).

Table Functions for PROC LUA

Here some of the table functions that are available with PROC LUA. You call these functions by preceding them with ‘TABLE’, such as `table.size(t)`.

Note: It is a SAS convention to document function names in all capitals. However, when you call these functions within the LUA procedure, you must use all lower-case letters.

Here are some of the commonly used table functions from the Lua table library that are available to PROC LUA.

```
TABLE.CONCAT(Lua-table<, "delimiter"><, start-position<, end-position>>)
TABLE.INSERT(Lua-table, <position>, > value)
TABLE.REMOVE(Lua-table<, position>)
TABLE.SORT(Lua-table<, comparison-function>)
```

Here are table functions that have been created specifically for PROC LUA. These functions can be called only within PROC LUA.

TABLE.CONTAINS(Lua-table-name,v)
returns the Boolean value **true** if the specified table contains the value *v*.
Enclose character values in single or double quotation marks.

For example, suppose you have created table T with a list of names. You can check to see whether a text string is contained in that table and print an appropriate message to the log.

```
local t = {"John", "Paul", "George", "Ringo"}

if (table.contains(t,"Ringo")) then
  print ("The table contains 'Ringo'.")
else
  print ("The table does not contain 'Ringo'.")
end
```

TABLE.SIZE(Lua-table-name)
returns the number of elements in the specified table.

TABLE.TOSTRING(Lua-table-name)
returns a formatted string representation of the specified table.

For more information, see [“Example 4: Loading a SAS Data Set and Viewing the Resulting Lua Table” on page 1192](#) and [“Example 6: Using Table Functions” on page 1198](#).

Functions That Manipulate Strings

Here are the string functions that are available with PROC LUA. You call these functions by preceding them with ‘STRING’, such as `string.trim(text-variable)`.

Note: It is a SAS convention to document function names in all capitals. However, when you call these functions within the LUA procedure, you must use all lower-case letters.

These string functions have been created specifically for PROC LUA. These functions can be called only within PROC LUA.

STRING.ENDS_WITH(string1, string2)
returns a Boolean that indicates whether the end of String1 matches the value of String2. The comparison is case sensitive. Enclose a literal text value in quotation marks.

`STRING.RESOLVE(string, {token1="text-value1"<, token2="text-value2"<, ...>>});`
 returns a string with tokens substituted into it. You provide a string variable with a value that contains substitution tokens in the format `@token-name@`. For each token, you provide a text value to substitute into the returned string value.

For example, suppose you have declared the variable Code:

```
local code="data @results@; set @in@; where @where@; run;"
```

You can provide substitute values for the tokens results, in, and where:

```
string.resolve(code, {results="work.foo", in="bar", where="x > 2"})
```

This call to `STRING.RESOLVE` returns this string:

```
data work.foo; set bar; where x > 2; run;
```

`STRING.SPLIT(string, delimiter, remove-empty-strings)`
 separates a string value into a table of substrings by using the specified delimiter value. Enclose the delimiter value in single or double quotation marks or inside double square brackets (`[[ ]]`). Use the last argument, a Boolean value, to specify whether to remove empty strings from the resulting table. By default, the last argument is `TRUE`.

`STRING.STARTS_WITH(string1, string2)`
 returns a Boolean that indicates whether the beginning of `String1` matches the value of `String2`. The comparison is case sensitive. Enclose a literal text value in single or double quotation marks or inside double square brackets (`[[ ]]`).

`STRING.TRIM(string)`
 returns a string with whitespace characters removed from the beginning and end of the original string value. Enclose literal text in single or double quotation marks or inside double square brackets (`[[ ]]`).

For more information, see [“Example 7: Using String Functions” on page 1200](#).

Object Syntax within Lua Statements

For some objects, such as data set IDs, Lua code supports object syntax, where you specify a function preceded by the object that the function is to act upon. For example, the following Lua statements are equivalent:

```
local luavar = sas.get_value(dsid, 'some_var')
```

```
local luavar = dsid.get_value('some_var')
```

Make sure that you use a colon (`:`) when placing the object name before the function that it acts upon.

Similarly, the following code block from [“Example 12: Using Iterator Functions for a Large Table” on page 1213](#) could be written in two ways.

```
-- Iterate over the rows of the data set
local i=0
while sas.next(dsid) do
  i=i+1
```



```

print("OBS=" .. i)
for vname,var in pairs(vars) do
    print(vname, '=', sas.get_value(dsid, vname) )
end
end

```

The SAS.NEXT and SAS.GET_VALUE functions can both be represented with object syntax.

```

-- Iterate over the rows of the data set
local i=0
while dsid:next() do
    i=i+1
    print("OBS=" .. i)
    for vname,var in pairs(vars) do
        print(vname, '=', dsid:get_value(vname) )
    end
end
end

```

Calling PROC FCMP Functions within Lua Statements

You can submit functions that are created using the FCMP procedure within Lua code. When you call PROC FCMP functions, you must specify the package in which the function is stored in the SAS OPTIONS statement.

If a PROC FCMP function modifies one of its arguments, that argument is specified in the OUTARGS statement. To retrieve changes to an argument in the OUTARGS statement within Lua code, that argument must be defined as an array.

For more information, see the following information:

- [Chapter 21, “FCMP Procedure,” on page 669](#)
- [“Example 15: Running PROC FCMP Functions” on page 1222](#)

Using the FULLSTIMER Option with PROC LUA

When you use the FULLSTIMER option with PROC LUA, the run times that FULLSTIMER reports in the log reflect the aggregate of run times for SAS code that is called within the LUA procedure. The run times include the run time of the LUA procedure itself.

For example, consider PROC LUA code that calls the SORT procedure three times. You might see the following output in the SAS log.

Output 34.1 PROC LUA Output Using FULLSTIMER

```

proc sort data=one; by i; run;
NOTE: There were 10000 observations read from the data set WORK.ONE.
NOTE: The data set WORK.ONE has 10000 observations and 2 variables.
NOTE: PROCEDURE SORT used (Total process time):
real time          0.03 seconds
user cpu time      0.03 seconds
system cpu time    0.00 seconds
memory            2252.60k
OS Memory          14492.00k
Timestamp          10/20/2015 11:43:40 AM
Step Count         10  Switch Count  0

proc sort data=two; by i; run;
NOTE: There were 10000000 observations read from the data set WORK.TWO.
NOTE: The data set WORK.TWO has 10000000 observations and 2 variables.
NOTE: PROCEDURE SORT used (Total process time):
real time          8.03 seconds
user cpu time      12.34 seconds
system cpu time    0.70 seconds
memory            71231.17k
OS Memory          83136.00k
Timestamp          10/20/2015 11:43:48 AM
Step Count         10  Switch Count  0

proc sort data=three; by i; run;
NOTE: There were 100 observations read from the data set WORK.THREE.
NOTE: The data set WORK.THREE has 100 observations and 2 variables.
NOTE: PROCEDURE SORT used (Total process time):
real time          0.01 seconds
user cpu time      0.00 seconds
system cpu time    0.01 seconds
memory            71231.17k
OS Memory          83136.00k
Timestamp          10/20/2015 11:43:48 AM
Step Count         10  Switch Count  0

NOTE: PROCEDURE LUA used (Total process time):
real time          8.15 seconds
user cpu time      12.37 seconds
system cpu time    0.78 seconds
memory            71231.17k
OS Memory          83136.00k
Timestamp          10/20/2015 11:43:48 AM
Step Count         10  Switch Count  0

```

In the output, the system CPU time, 0.78 seconds, reflects the aggregate of the calls to PROC SORT and additional CPU time that is used by PROC LUA.

Syntax: LUA Procedure

Restriction:

When SAS is in a locked-down state, access to the Lua IO and OS libraries is limited. Your server administrator can follow the steps in the following links to remove these restrictions.

- [“Example 3: Enabling the Lua OS Library While in Lockdown Mode” in SAS Viya Platform: Programming Run-Time Servers](#)

- “Example 4: Enabling the Lua IO Library While in Lockdown Mode” in *SAS Viya Platform: Programming Run-Time Servers*

```
PROC LUA <INFILE='filename'> <SECURELUATOKEN=
"digital_signature"><RESTART> <TERMINATE>;
  <SUBMIT <"assignment(s);">;>
    ... Lua statements ...
  <ENDSUBMIT>;>
run;
```

Statement	Task
PROC LUA	Execute Lua statements within SAS code or specify a file that contains Lua statements to execute
SUBMIT	Identify the beginning of a block of Lua statements
ENDSUBMIT	Identify the end of a block of Lua statements

PROC LUA Statement

Runs Lua statements within a SAS session.

Syntax

```
PROC LUA <INFILE='filename'><SECURELUATOKEN= "digital_signature">
<RESTART> <TERMINATE>;
```

Optional Arguments

INFILE= 'filename'

identifies a source file that contains Lua statements to run within a SAS session. SAS expects this file to end with a .lua file extension, but do not include the extension in the filename that you specify.

Requirements Enclose the filename in single or double quotation marks.

If you use the INFILE= option, then you must specify the path to the Lua script. Define the path to your Lua scripts by providing a value for the LUAPATH filename before the PROC LUA statement. For more information, see [“Example 3: Specifying Input from an External Lua Script”](#) on page 1191.

Example

Specify

```
INFILE= 'open_data '
```

to use the code in the open_data.lua Lua script file.

SECURELUATOKEN= "digital_signature"

provides a *digital_signature* encoded in BASE64URL encoding.

Restriction PROC LUA does not allow the Lua LOAD function to load compiled binary code. The LOAD function supports only ordinary Lua scripts that are not compiled.

Requirement

- The LUACERTLOC environment variable defines the certificate used to validate the Lua content. The location of the certificate file must also be included in the LOCKDOWN path list and specified via the SECURELUALOC system option.
- The key pair must be generated with the RSA public key algorithm and a key length of 4096 bits.
- A digest of the Lua script must be computed with the SHA 256 algorithm. The line endings in the Lua script must be represented using the LF control character (0x0A), not CRLF.
- The digest for the Lua script must be signed using the private key and the PKCS1 v1.5 padding scheme defined in [RFC 3447](#).
- The certificate file is found via the LUACERTLOC environment variable.
- The location of the certificate file must be added to the LOCKDOWN path list via the LOCKDOWN PATH statement and specified via the SECURELUALOC option.
- For more information about creating digital signatures, see [How to use OpenSSL: Hashes, digital signatures, and more..](#)

Example Specify

```
SECURELUATOKEN="digital_signature"
```

to validate the security of the Lua script. The SECURELUATOKEN argument may also be specified using a SAS macro variable. For example:

```
%let digsig = drQr4TrD9EBCFCNkkpcftcf-fxj3E74k5YeGct15BV8.....;

proc lua secureluatoken=&digsig;
submit;
    <lua statements...>
endsubmit;
run;
```

RESTART

resets the state of Lua code submissions for a SAS session. The LUA procedure is a reentrant procedure that maintains the state for Lua code across calls to the LUA procedure. This means that global Lua variable assignments or function definitions remain in memory until you issue the RESTART option, issue the TERMINATE option, or end the SAS session.

You can specify RESTART at the beginning of a new block of Lua code.

```
Example  proc lua restart;
         submit;
         <lua statements...>
         endsubmit;
         run;
```

TERMINATE

stops maintaining the Lua code state in memory and terminates the Lua state when the LUA procedure completes. Subsequent calls to the LUA procedure begin a new instance of the Lua code state.

SUBMIT Statement

Identifies the beginning of a block of Lua code. Enter Lua statements between the SUBMIT and ENDSUBMIT statements.

Requirement: Each SUBMIT statement must have a corresponding ENDSUBMIT statement.

Syntax

SUBMIT <'assignments';>;

Optional Argument

assignments;

identifies one or more macro variable assignments that are passed to the block of Lua statements. If only one assignment is listed, then the semicolon (;) within the quotation marks is not required. SAS does not expand macro variables within a block of Lua statements. Therefore, macro values must be passed within the list of assignments for the SUBMIT statement.

Example To assign the value of macro variable N to the Lua variable Name, enter the following SUBMIT statement:

```
SUBMIT "name=&n";
```

Details

The SUBMIT/ENDSUBMIT statement within PROC LUA is not allowed inside a macro definition. Instead, you can use an %INCLUDE statement or the INFILE= argument to run LUA code from the macro definition.

ENDSUBMIT Statement

Identifies the end of a block of Lua statements. Do not enter any other statement on the same line as the ENDSUBMIT statement.

Syntax

```
ENDSUBMIT;
```

Details

The SUBMIT/ENDSUBMIT statement within PROC LUA is not allowed inside a macro definition. Instead, you can use an %INCLUDE statement or the INFILE= argument to run LUA code from the macro definition.

Usage: LUA Procedure

Submitting Lua Statements within a SAS Program

You can submit Lua statements within the PROC LUA invocation, between SUBMIT and ENDSUBMIT statements. The following code executes a single Lua print statement.

```
proc lua;  
  submit;  
    print("Hello from Lua")  
  endsubmit;  
run;
```

Any filerefs, librefs, macro variables, and so on, that you define within a SUBMIT and ENDSUBMIT block are available only within that block of code.

For example, the macro variable Mymacrovar is defined within the SUBMIT and ENDSUBMIT block in the following call to PROC LUA. However, the variable is not defined in the %PUT statement at the end of the example.

```
proc lua restart;
submit;
  sas.submit([[%let mymacrovar=Hi there;]])
  txt = sas.symget("mymacrovar")
  print(txt)
endsubmit;
run;

%put &mymacrovar;
```

Output 34.2 Scope for Macro Variable in PROC LUA

```
%let mymacrovar=Hi there;
Hi there
NOTE: PROCEDURE LUA used (Total process time):
      real time          0.14 seconds
      cpu time           0.07 seconds

WARNING: Apparent symbolic reference MYMACROVAR not resolved.
17
18  %put &mymacrovar;
&mymacrovar
```

Opening a SAS Data Set within Lua Code

Calling the SAS.SUBMIT function enables you to submit a block of SAS code. Enclose the SAS code within [[and]] brackets. The submitted code in this sample first verifies that the SAS data set exists. You can make any changes to the data, via the submitted DATA step, that you would in Base SAS code.

```
/* Test whether a data set exists */
proc lua;
submit;
  if sas.exists("sashelp.air") then
    print("The data set SASHELP.AIR exists.")
  else
    print("The data set SASHELP.AIR does not exist.")
  end
endsubmit;
run;
```

If the data set exists, then you can open the data set and read it into a new data set, WORK.AIR. You can make any changes to the data via the submitted DATA step that you would in a typical DATA step.

Note: Longer blocks of SAS code are typically assigned to a Lua variable.

```
proc lua;
submit;
  sas.submit( [[ data work.air; set sashelp.air; run; ]] )
endsubmit;
run;
```

You can also substitute Lua variable values within the SAS code. The following code shows simple substitutions. For more information, see [“Example 10: Submitting SAS Code with Lua Variable Substitutions”](#) on page 1208.

```
proc lua;
submit;
  local dest = 'work.class'
  local source = 'sashelp.class'

  sas.submit( [[ data @dest@; set @source@; run; ]] )
endsubmit;
run;
```

Examples: LUA Procedure

Example 1: Create a Sample Data Set

This example creates a sample SAS data set, `Homes`, that contains several variables and 20 observations. This data set is the input for some of the examples that follow.

Here is the code to create a sample data set that can be accessed using PROC LUA.

```
data homes;
  input bad loan mortdue value reason $ job $ yoj derog delinq
        clage ninq clno debtinc;
datalines;
1 1100 25860 39025 HomeImp Other 10.5 0 0 94.37 1 9 .
0 4700 71855 88566 HomeImp Other 2.0 2 0 283.96 0 5 36.475
0 5500 72147 69918 HomeImp Sales 4.0 2 0 158.53 0 23 43.404
1 6400 25144 45200 HomeImp Other 25.0 0 2 128.00 4 17 .
0 7000 58114 93391 HomeImp ProfEx 6.0 0 0 200.08 1 24 31.737
1 7900 67222 75189 HomeImp Other 4.0 0 0 95.68 0 23 36.980
0 8300 54039 89301 HomeImp Other 18.0 0 0 173.19 0 28 26.267
0 8800 . 32221 DebtCon Other 0.0 0 0 276.84 0 14 24.199
0 9400 71600 99682 DebtCon Other 16.0 . . 159.04 4 16 25.607
0 10000 68807 76581 HomeImp Office 14.0 0 0 237.65 0 32 42.336
0 10200 16322 86505 DebtCon Other 8.0 0 0 259.16 0 14 21.253
0 10700 115118 124198 DebtCon Self 6.0 1 0 174.34 0 19 31.758
0 11100 148235 182053 HomeImp Office 4.0 0 0 198.81 0 55 33.539
0 11700 56441 86987 HomeImp Other 17.0 0 0 198.03 0 16 33.931
0 12100 58556 76724 HomeImp Other 3.0 0 1 234.66 1 16 37.639
```



```

0 12500 81865 101048 DebtCon Other 1.0 0 0 147.40 0 23 38.855
0 12900 18106 37881 DebtCon Mgr 8.0 0 0 134.38 0 10 20.516
0 13400 98701 129679 HomeImp ProfEx 10.0 0 0 179.13 2 32 29.549
1 13900 . . HomeImp Other 4.0 0 2 209.48 0 15 35.775
0 14400 45516 63924 DebtCon Other . 0 0 109.33 1 19 40.872
;

```

Example 2: Running Lua Scripts with SECURELUA Enabled

Details

The SECURELUA system option is enabled using one of the following methods.

- specifying -SECURELUA option in the command starting SAS
- in the SAS configuration file, `sasv9_local.cfg`

When this option is enabled, there are several steps that you need to take to successfully run Lua scripts. These are provided in this example.

Program

- 1 Run the following command at a Linux command shell prompt to create the 4096-bit RSA private key. This generates the file, `private_key.pem`, which contains the 4096-bit RSA private key.

```
openssl genpkey -algorithm RSA -out private_key.pem -pkeyopt rsa_keygen_bits:4096
```

- 2 You can extract the RSA public key from the RSA private key. The RSA public key is not directly used again in this example. Its creation is shown here for educational purposes only. In this example, the file `public_key.pem` contains the RSA public key when the command has completed. This step is optional.

```
openssl pkey -in private_key.pem -pubout -out public_key.pem
```

- 3 Create a simple text file, `my_lua_script.lua`. This file contains only the Lua code (or script) that appears inside the Submit and Endsubmit block of PROC LUA. The record delimiter in this text file must be LF (0x0a) and not CRLF (0x0d0a). For example, your SAS program might look like this.

```

proc lua;
  submit;
    print ("") --blank line
    print ("Hello from 'Proc Lua'.")
  endsubmit;
quit;

```

```

        print ("Length of word 'house' is " .. string.len("house") .. " bytes.")
        print ("Goodbye from 'Proc Lua'.")
        print ("") --blank line
    endsubmit;
run;

```

Place the Print lines in the new file, 'my_lua_script.lua'. Do not place 'submit;' or 'endsubmit;' in 'my_lua_script.lua'.

```

    print ("") --blank line
    print ("Hello from 'Proc Lua'.")
    print ("Length of word 'house' is " .. string.len("house") .. " bytes.")
    print ("Goodbye from 'Proc Lua'.")
    print ("") --blank line

```

Note:

- If the lines of Lua code in `my_lua_script.lua` are not identical to those in the actual SAS program, PROC LUA cannot run the Lua code in the Submit and Endsubmit block when it is operating in SECURELUA mode. This restriction is in place to ensure that the Lua code in the Submit and Endsubmit block has not changed between the time that it was first digitally signed and the time PROC LUA executes it.
 - Each of the five PRINT statements in the Submit and Endsubmit block above is preceded by four spaces. The spaces must be included in the lines of Lua code added to `my_lua_script.lua`.
 - The first and last PRINT statements contain a comment at the end of the statement. The entire line containing each of these PRINT statements, including the comment at the end, must be added to `my_lua_script.lua`.
-

- 4 Create a digital signature (encrypted hash) of the PROC LUA code in the Submit and Endsubmit block. In the previous step, this PROC LUA code was placed in text file `my_lua_script.lua`. Run the following command at a Linux command shell prompt.

```

openssl dgst -sha256 -sign private_key.pem -sigopt rsa_padding_mode:pkcs1
                                -out sign.sha256 my_lua_script.lua.

```

The file `sign.sha256` now contains the binary digital signature of the PROC LUA code in `my_lua_script.lua`.

- 5 The SECURELUATOKEN argument to PROC LUA is the Base64URL-encoded digital signature of the PROC LUA code in the Submit and Endsubmit block. File `sign.sha256` contains the binary digital signature of this PROC LUA code. This binary digital signature must be transcoded to Base64URL before it is suitable for use as the SECURELUATOKEN argument to PROC LUA.

OpenSSL has no native support to transcode binary data to Base64URL encoding. Instead, use the Linux-standard *basenc* utility to perform the conversion. Run the following command at a Linux command shell prompt:

```

basenc --base64url --wrap=0 sign.sha256 > sign.sha256.base64url

```

The file `sign.sha256.base64url` now contains the Base64URL-encoded digital signature of the PROC LUA code in `my_lua_script.lua`. It is the contents of file

sign.sha256.base64url that must be specified as the value of the SECURELUATOKEN argument to PROC LUA.

- 6 Run the following command to create the self-signed certificate that is used to validate the digital signature created above.

```
openssl req -x509 -sha256 -days 365 -key private_key.pem
-sigopt rsa_padding_mode:pkcs1 -out certificate.crt
```

Note: This command generates many questions that you can skip over by pressing Enter to each one. You can avoid the questions by appending -batch to the above openssl req -x509 command.

The file certificate.crt now contains the certificate that PROC LUA uses to validate the digital signature. You now have the OpenSSL artifacts (RSA keys, digital signature, and certificate) that are necessary to run PROC LUA in SECURELUA mode. Only a few setup steps remain.

- 7 This example assumes that all necessary PROC LUA components are located in a single directory, /users/sasuser. An autoexec.sas file containing the LOCKDOWN statement should be located in this directory. Create the LUACERTLOC environment variable that defines the location of the certificate file.

```
export LUACERTLOC=/users/sasuser/certificate.crt
```

- 8 Add the following line to /users/sasuser/autoexec.sas. If this file does not exist, create it.

```
lockdown path="/users/sasuser";
```

This lockdown line is an instance of the LOCKDOWN statement. The LOCKDOWN statement can be successfully executed only if the LOCKDOWN option is enabled.

- 9 Run a simple PROC LUA test case in SAS. This SAS code is named my_securelua_test.sas. It runs successfully in SECURELUA mode.

```
$ cd /users/sasuser
$ cat my_securelua_test.sas
```

```
/* Display values of "internal" SECURELUA and SECURELUALOC options. */
proc options option=(securelua securelualoc) internal; run;
```

```
/* Create fileref for passed-in filename containing Base64URL-encoded digital
 * signature.
 */
filename myinfil "&sysparm";
```

```
/* Use Data Step to open passed-in file and read Base64URL-encoded digital
 * signature.
 *
 * Grab Base64URL-encoded digital signature. Place in Data Step variable.
 */
data _null_;
```

```

infile myinfil;
length inrec $ 1024;
input inrec $;

/* Create new macro variable 'digsig' and set its value to that of Data
 * Step variable 'inrec'. Need to trim trailing blanks from Data Step
 * variable value before assigning this value to a new macro variable.
 */
call symput('digsig', trim(inrec));
run;

/* Run 'Proc Lua' in SECURELUA mode. */
proc lua secureluatoken="&digsig";
submit;
  print ("") --blank line
  print ("Hello from 'Proc Lua'.")
  print ("Length of word 'house' is " .. string.len("house") .. " bytes.")
  print ("Goodbye from 'Proc Lua'.")
  print ("") --blank line
endsubmit;
run;

endsas;

```

- 10 Run a PROC LUA test case in SECURELUA mode. Note that PROC LUA operates in SECURELUA mode only if the following conditions are true.

- LOCKDOWN and SECURELUA options are enabled
- A value is specified for the SECURELUALOC option

- 11 Run the following commands. This example assumes that the main SAS executable is `/install/SASHome/SASFoundation/9.4/sas`.

```

$ cd /users/sasuser
$ /install/SASHome/SASFoundation/9.4/sas -sysin my_securelua_test.sas
    -securelua -securelualoc '/users/sasuser' -lockdown -sysparm sign.sha256.base64url

```

If the PROC LUA test case, `my_securelua_test.sas` runs successfully to completion, the SAS log file contains the following:

```

NOTE: Lua initialized.
[Blank line.]
Hello from 'Proc Lua'.
Length of word 'house' is 5 bytes.
Goodbye from 'Proc Lua'.
[Blank line.]

```

If the log file contains any of these errors, it is highly likely that at least one component of PROC LUA's SECURELUA mode environment has not been set up correctly.

- Unable to decode SECURELUATOKEN string.
ERROR: Unable to validate SECURELUATOKEN.
- NOTE: RS256 public key signature did not validate.
NOTE: bad signature
- Unable to locate certificate file.
ERROR: Not authorized to submit Lua code in LOCKDOWN.

Example 3: Specifying Input from an External Lua Script

Features: FILENAME statement
 PROC LUA statement, INFILE= option

Details

This example specifies an external Lua script and one or more possible paths to that script by using the FILENAME statement and the INFILE= option in the PROC LUA statement.

Program

```
filename LUAPATH ('/usr/local/scripts/lua','/home/user/myname/
my_scripts');

proc lua infile='my_script';
run;
```

Program Description

Specify the directories to search for the input Lua script. The FILENAME statement defines the directories in which Lua scripts are stored and assigns them to the LUAPATH fileref. Enclose a directory path within single or double quotation marks.

```
filename LUAPATH ('/usr/local/scripts/lua','/home/user/myname/
my_scripts');
```

Execute the PROC LUA statement and specify the Lua script name. The INFILE= option provides the name of the Lua script that contains Lua statements. In this example, SAS executes the Lua statements in the my_script.lua file. Do not specify the '.lua' or '.luc' file extension. Check the package.path variable for your system to see whether LUA or LUC files are opened first.

```
proc lua infile='my_script';
run;
```

Example 4: Loading a SAS Data Set and Viewing the Resulting Lua Table

Features: SAS.EXISTS function
 SAS.READ_DS function
 Observation processing: WHILE and FOR loops

Details

This example uses the Homes data set that you created in Example 1.

In this example, you read the SAS data set Homes and print the data to the SAS log. Treat individual observations as an entry in an array. Each array entry contains associated attributes that are derived from the variables in the original SAS data set. For example, the value of `t[2].loan` is 4700.

Program

```
proc lua;
submit;
  if (sas.exists("work.homes")) then
    local t = sas.read_ds("work.homes")
    i=1
    while(t[i] ~= nil) do
      print("Obs #" .. i)
      for k,v in pairs(t[i]) do
        print(k,v)
      end
      print("\n")
      i = i+1
    end
  end
endsubmit;
run;
```

Program Description

Verify that the SAS data set Homes exists and read it into the Lua table T. Each observation in the data set can be accessed as if it were in an array, such as `t[i]`.

```

proc lua;
submit;
  if (sas.exists("work.homes")) then
    local t = sas.read_ds("work.homes")
    i=1

```

Process each observation in the data set. Initialize an iterator, *i*, and process each observation by using a WHILE loop. Check to see whether the next observation exists and then print each variable-value pair. Use a FOR loop to process each variable name and value, printing both to the SAS log. At the end of each observation, print a newline character and increment *i*.

```

  while(t[i] ~= nil) do
    print("Obs #" .. i)
    for k,v in pairs(t[i]) do
      print(k,v)
    end
    print("\n")
    i = i+1
  end

```

Submit the PROC LUA call to SAS.

```

  end
endsubmit;
run;

```

Output: Lua Table

Output 34.3 *Sample Lua Data*

```
Obs #1
reason      HomeImp
derog       0
job         Other
yoj         10.5
clno        9
loan        1100
bad         1
debtinc     .
mortdue     25860
value       39025
clage       94.37
ning        1
delinq      0

...
Obs #20
reason      DebtCon
derog       0
job         Other
yoj         .
clno        19
loan        14400
bad         0
debtinc     40.872
mortdue     45516
value       63924
clage       109.33
ning        1
delinq      0

NOTE: PROCEDURE LUA used (Total process time):
      real time          0.17 seconds
      cpu time           0.18 seconds
```

Example 5: Writing a SAS Data Set from a Lua Table

Features:

- PROC LUA statement
- SUBMIT and ENDSUBMIT statements
- SAS.WRITE_DS function
- TABLE.TOSTRING function

Details

This example creates a Lua table and then writes that table to a SAS data set. The values in the Lua table are generated by calling the SAS RANNOR and RANUNI random number generator functions. This code also uses Lua conventions for processing arrays.

Program

```
proc lua;
submit;
  local tbl = {}

  for i=1,10 do
    vars = {}
    vars.seed = 1234 * i;
    vars.randnor = sas.rannor( vars.seed )
    vars.randuni = sas.ranuni( vars.seed )
    vars.color = "purple"
    tbl[#tbl+1] = vars
  end

  print("Lua table:", table.tostring(tbl))

  sas.write_ds(tbl, "work.random")
endsubmit;
run;

proc print data=random;run;
```

Program Description

Execute the PROC LUA statement and begin a block of Lua code. Declare a local Lua array called TBL.

```
proc lua;
submit;
  local tbl = {}
```

Generate the contents of the Lua table. This example generates the content for the array TBL. The variables Seed, Randnor, Randuni, and Color are created and assigned a value over ten iterations of a FOR loop.

```
  for i=1,10 do
    vars = {}
```

```

vars.seed = 1234 * i;
vars.randnor = sas.rannor( vars.seed )
vars.randuni = sas.ranuni( vars.seed )
vars.color = "purple"
tbl[#tbl+1] = vars
end

```

Print the generated Lua table. The resulting Lua table is printed to the SAS log.

```
print("Lua table:", table.tostring(tbl))
```

Write the Lua table to a SAS data set. This example writes the Lua table to a SAS data set called Random in the Work library. By saving the data set to the Work library, it is accessible only within the same SAS session. You can save the data set permanently by saving it to a library, such as work.

```

sas.write_ds(tbl, "work.random")
endsubmit;
run;

```

Print the SAS data set. After you have written the SAS data set, you can access it outside of the LUA procedure. If you save the data set to a library, such as Sasuser, then the data set is accessible in later SAS sessions.

```
proc print data=random;run;
```

Output: SAS Table from a Lua Table

Output 34.4 Partial Lua Table from the SAS Log

```

Lua table:      table: 000000002892EAE0=
{
  [1]=table: 00000000289359A0=
  {
    ["color"]="purple"
    ["randuni"]=0.3831937143
    ["randnor"]=1.4215132075
    ["seed"]=1234
  }
  [2]=table: 0000000028937640=
  {
    ["color"]="purple"
    ["randuni"]=0.088249594
    ["randnor"]=-0.102644377
    ["seed"]=2468
  }
  [3]=table: 000000000C195400=
  {
    ["color"]="purple"
    ["randuni"]=0.4458283407
    ["randnor"]=2.0089305781
    ["seed"]=3702
  }
  [4]=table: 0000000028928FE0=
  {
    ["color"]="purple"
    ["randuni"]=0.4562234927
    ["randnor"]=1.9125666228
    ...
  }
}

```

Output 34.5 Generated SAS Table

The SAS System				
Obs	seed	randuni	randnor	color
1	1234	0.38319	1.42151	purple
2	2468	0.08825	-0.10264	purple
3	3702	0.44583	2.00893	purple
4	4936	0.45622	1.91257	purple
5	6170	0.96744	1.82697	purple
6	7404	0.37316	-0.63993	purple
7	8638	0.28390	-1.49036	purple
8	9872	0.43640	-0.05252	purple
9	11106	0.00276	-0.45912	purple
10	12340	0.23450	0.96227	purple

Example 6: Using Table Functions

Features:

- PROC LUA statement
- SUBMIT and ENDSUBMIT statements
- TABLE.CONCAT function
- TABLE.INSERT function
- TABLE.REMOVE function
- TABLE.SIZE function
- TABLE.SORT function
- TABLE.TOSTRING function

Details

This example creates a simple table, prints the number of elements, and prints the table to the SAS log.

Program

```
proc lua;
submit;

  local t={"a", "b", "c", "foo", "bar"}

  print(table.size(t))
  print("Output with function TABLE.TOSTRING")
  print(table.tostring(t))
  print("Output with function TABLE.CONCAT")
  print(table.concat(t, ", "))

  print("Inserting and Removing Values")
  table.insert(t,5,"new")
  print("Table with new value: ", table.concat(t, ", "))
  table.remove(t,4)
  print("Table after removing value at position 4: ",
table.concat(t, ", "))

  print("Sorting a Table")
  table.sort(t)
  print("Table with sorted values: ",table.concat(t, ", "))
  table.sort(t, function(a,b) return a>b end)
```

```

        print("Table sorted in descending order: ",table.concat(t," "))
    endsubmit;
run;

```

Program Description

Execute the PROC LUA statement. The PROC LUA statement enables you to call Lua code within your SAS session.

```
proc lua;
```

Identify the beginning of a block of Lua statements. The SUBMIT statement identifies the beginning of a block of Lua statements.

```
submit;
```

Create a simple table. Assign elements in a table T.

```
local t={"a", "b", "c", "foo", "bar"}
```

Print information about the table. Print the number of elements in the table and then print the table using the TABLE.TOSTRING function and the TABLE.CONCAT function.

```

print(table.size(t))
print("Output with function TABLE.TOSTRING")
print(table.tostring(t))
print("Output with function TABLE.CONCAT")
print(table.concat(t, " "))

```

Insert and remove items in the table. Print a label, and then insert the value "new" into the table in the fifth position using the TABLE.INSERT function. Print the table to see the added value. Remove the value in the fourth position in the table, "foo", using the TABLE.REMOVE function. Print the resulting table.

```

print("Inserting and Removing Values")
table.insert(t,5,"new")
print("Table with new value: ", table.concat(t," "))
table.remove(t,4)
print("Table after removing value at position 4: ",
table.concat(t," "))

```

Sort values in the table. Print a label, and then sort the table in ascending order using the TABLE.SORT function. Print the resulting table to see the sorted values. Next, sort the table again and specify a function that sorts the table in descending order. Print the resulting table.

```

print("Sorting a Table")
table.sort(t)
print("Table with sorted values: ",table.concat(t," "))
table.sort(t, function(a,b) return a>b end)
print("Table sorted in descending order: ",table.concat(t," "))

```

Identify the end of a block of Lua statements with the ENDSUBMIT statement.

```
endsubmit;
run;
```

Output: SAS Log That Shows Table Details

Output 34.6 Results of TABLE Functions

```
Output with function TABLE.TOSTRING
table: 00000000289200E0=
{
  [1]="a"
  [2]="b"
  [3]="c"
  [4]="foo"
  [5]="bar"
}
Output with function TABLE.CONCAT
a, b, c, foo, bar
Inserting and Removing Values
Table with new value:      a, b, c, foo, new, bar
Table after removing value at position 4:      a, b, c, new, bar
Sorting a Table
Table with sorted values:      a, b, bar, c, new
Table sorted in descending order:      new, c, bar, b, a
NOTE: PROCEDURE LUA used (Total process time):
      real time          0.03 seconds
      cpu time           0.01 seconds
```

Example 7: Using String Functions

Features:

- PROC LUA statement
- SUBMIT and ENDSUBMIT statements
- STRING.ENDS_WITH function
- STRING.RESOLVE function
- STRING.SPLIT function
- STRING.STARTS_WITH function
- STRING.TRIM function

Details

This example uses the STRING functions to manipulate text values.

Program

```

proc lua;
  submit;

  mystring = "@noun@ @verb@ the @object@"

  newstring1 = string.resolve(mystring,
    {noun="Pigs",verb="eat",object="pie"})
  newstring2 = string.resolve(mystring, {noun="Cars",verb="guzzle",
    object="gasoline"})

  print ("Original string: " .. mystring)
  print ("New string 1: " .. newstring1)
  print ("New string 2: " .. newstring2)

  if string.starts_with(newstring2, "Cars") then
    print("Could be about cars: " .. newstring2)
  else
    print("Not about cars: " .. newstring2)
  end

  if string.ends_with(newstring2, "pie") then
    print ("I love pie!" .. newstring2)
  else
    print("Too bad. No pie: " .. newstring2)
  end

  mystring = "      The fox ate the chicken.  "
  trimmedstring = string.trim(mystring)

  print ("Original string between ***s: ***" .. mystring .. "****")
  print ("Trimmed string between ***s: ***" .. trimmedstring ..
    "****")

  t = string.split(mystring," ")

  for element in pairs(t) do
    print("Table index " .. element .. ", Value is: " ..
    t[element] .. "\n")
  end

endsubmit;
run;

```

Program Description

Execute the PROC LUA statement and begin processing Lua statements. The PROC LUA statement enables you to call Lua code within your SAS session. The SUBMIT statement identifies the beginning of a block of Lua statements.

```
proc lua;
```

```
submit;
```

Create a string variable and substitute different values in it. Create a string, Mystring, that contains substitution tokens Noun, Verb, and Object. Use the STRING.RESOLVE function to generate new strings, Newstring1 and Newstring2. Print the original string and the new strings to the SAS log.

```
mystring = "@noun@ @verb@ the @object@"

newstring1 = string.resolve(mystring,
{noun="Pigs",verb="eat",object="pie"})
newstring2 = string.resolve(mystring, {noun="Cars",verb="guzzle",
object="gasoline"})

print ("Original string: " .. mystring)
print ("New string 1: " .. newstring1)
print ("New string 2: " .. newstring2)
```

Look for matching at the beginning and end of string variables. Use the STRING.STARTS_WITH function to check whether Newstring2 begins with “Cars” and then print the appropriate message. Next, use the STRING.ENDS_WITH function to check whether Newstring2 ends with “pie” and then print the appropriate message.

```
if string.starts_with(newstring2, "Cars") then
  print("Could be about cars: " .. newstring2)
else
  print("Not about cars: " .. newstring2)
end

if string.ends_with(newstring2, "pie") then
  print ("I love pie!" .. newstring2)
else
  print("Too bad. No pie: " .. newstring2)
end
```

Trim leading and trailing whitespace characters from a string value. Assign a new value to Mystring. Use the STRING.TRIM function to remove leading and trailing whitespace characters. Print the original and trimmed strings for comparison.

```
mystring = "    The fox ate the chicken.  "
trimmedstring = string.trim(mystring)

print ("Original string between ***s: ***" .. mystring .. "****")
print ("Trimmed string between ***s: ***" .. trimmedstring ..
"****")
```

Create a list of words in a sentence string. Use the STRING.SPLIT function to generate a list of individual words from the string Mystring. By default, blank values are not included in the resulting list. Print the list of words in Mystring.

```
t = string.split(mystring, " ")

for element in pairs(t) do
  print("Table index " .. element .. ", Value is: " ..
t[element] .. "\n")
```



```
end
```

End the block of Lua statements and run the LUA procedure. End the block of Lua statements with the ENDSUBMIT statement.

```
endsubmit;
run;
```

Output: SAS Log That Shows String Function Results

Output 34.7 Results of STRING Functions

```
NOTE: Lua initialized.
Original string: @noun@ @verb@ the @object@
New string 1: Pigs eat the pie
New string 2: Cars guzzle the gasoline
Could be about cars: Cars guzzle the gasoline
Too bad. No pie: Cars guzzle the gasoline
Original string between ***s: ***   The fox ate the chicken.   ***
Trimmed string between ***s: ***The fox ate the chicken.***
Table index 1, Value is: The

Table index 2, Value is: fox

Table index 3, Value is: ate

Table index 4, Value is: the

Table index 5, Value is: chicken.

NOTE: PROCEDURE LUA used (Total process time):
      real time           0.20 seconds
      cpu time            0.06 seconds
```

Example 8: Adding an Observation to a Lua Table

Features:

- SAS.APPEND function
- SAS.CLOSE function
- SAS.GET_VALUE function
- SAS.PUT_VALUE function
- SAS.UPDATE function
- Concatenation operator (..)

Details

This example demonstrates how to add an observation using the SAS functions in PROC LUA. First, create a simple data set, and add a record to it using the SAS.APPEND, SAS.PUT_VALUE, and SAS.UPDATE functions. Next, retrieve and print the values from the updated data set.

Program

```
data foo;
    input x y;
    id+1;
    datalines;
1 10
2 20
3 30
;

proc lua;
submit;
    dsid = sas.open("work.foo",'u')

    while(sas.next(dsid) ~= nil) do
        myid = sas.get_value(dsid,"id")
        myx = sas.get_value(dsid,"x")
        myy = sas.get_value(dsid,"y")
        print("ID: " .. myid .. " x: " .. myx .. " y: " .. myy)
    end

    while(sas.next(dsid) ~= nil) do
        myid = sas.get_value(dsid,"id")
        myx = sas.get_value(dsid,"x")
        myy = sas.get_value(dsid,"y")
        print("ID: " .. myid .. " x: " .. myx .. " y: " .. myy)
    end

    sas.append(dsid)
    sas.put_value(dsid,"id",4)
    sas.put_value(dsid,"x",4)
    sas.put_value(dsid,"y",40)
    sas.update(dsid)
    rc=sas.close(dsid)
endsubmit;
run;

proc lua restart;
submit;
    dsid=sas.open("work.foo")
```

```

while(sas.next(dsid) ~= nil) do
  myid = sas.get_value(dsid,"id")
  myx = sas.get_value(dsid,"x")
  myy = sas.get_value(dsid,"y")
  print("ID: " .. myid .. " x: " .. myx .. " y: " .. myy)
end

rc=sas.close(dsid)
endsubmit;
run;

```

Program Description

Create a data set. Use the DATA step to create a small data set called Foo.

```

data foo;
  input x y;
  id+1;
  datalines;
1 10
2 20
3 30
;

```

Invoke PROC LUA and open the Foo data set for updating. Use the SAS.OPEN function to assign the contents of the Foo data set to the Dsid Lua variable. By specifying that you want to open the data set in Update mode (by using the 'u' argument), you are able to write to the data set. If you do not specify Update mode, the data set opens in Read-Only mode by default.

```

proc lua;
submit;
  dsid = sas.open("work.foo",'u')

  while(sas.next(dsid) ~= nil) do
    myid = sas.get_value(dsid,"id")
    myx = sas.get_value(dsid,"x")
    myy = sas.get_value(dsid,"y")
    print("ID: " .. myid .. " x: " .. myx .. " y: " .. myy)
  end
end

```

Read the values in Foo and print them to the SAS log. Use the WHILE loop with the SAS.NEXT function to iterate through observations in the data set. For each observation, retrieve the values for Myid, Myx, and Myy from the ID, X, and Y variables, respectively. Print these values to the SAS log.

```

while(sas.next(dsid) ~= nil) do
  myid = sas.get_value(dsid,"id")
  myx = sas.get_value(dsid,"x")
  myy = sas.get_value(dsid,"y")
  print("ID: " .. myid .. " x: " .. myx .. " y: " .. myy)
end

```

Append a new observation and assign a value to each variable. Call the SAS.APPEND function to prepare the data set for a new observation. Assign values using the SAS.PUT_VALUE function and finally update the data set with the SAS.UPDATE function. Close the data set.

```
sas.append(dsid)
sas.put_value(dsid,"id",4)
sas.put_value(dsid,"x",4)
sas.put_value(dsid,"y",40)
sas.update(dsid)
rc=sas.close(dsid)
endsubmit;
run;
```

Restart PROC LUA and open the Work.Foo data set. Assign the contents of Work.Foo to the Dsid Lua variable.

```
proc lua restart;
submit;
dsid=sas.open("work.foo")
```

Retrieve the data from the Work.Foo data set. Use a WHILE loop to process observations in the Work.Foo data set. Retrieve the values using the SAS.GET_VALUE function, and print the values to the SAS log.

```
while(sas.next(dsid) ~= nil) do
  myid = sas.get_value(dsid,"id")
  myx = sas.get_value(dsid,"x")
  myy = sas.get_value(dsid,"y")
  print("ID: " .. myid .. " x: " .. myx .. " y: " .. myy)
end
```

Close the Work.Foo data set.

```
rc=sas.close(dsid)
endsubmit;
run;
```

Output 34.8 Listing of Original Work.Foo Data Set

```
ID: 1 x: 1 y: 10
ID: 2 x: 2 y: 20
ID: 3 x: 3 y: 30
NOTE: PROCEDURE LUA used (Total process time):
      real time          0.02 seconds
      cpu time           0.01 seconds
```

Output 34.9 Listing of Work.Foo Data Set Appended with New Observation

```
ID: 1 x: 1 y: 10
ID: 2 x: 2 y: 20
ID: 3 x: 3 y: 30
ID: 4 x: 4 y: 40
NOTE: PROCEDURE LUA used (Total process time):
      real time          0.16 seconds
      cpu time           0.06 seconds
```

Example 9: Using SAS Macro Variable Values within Lua Statements

Features:

- PROC LUA statement
- SUBMIT and ENDSUBMIT statements
- Macro variable assignments
- Concatenation operator (..)

Details

No macro substitution occurs between the semicolon (;) at the end of the SUBMIT statement and the beginning of the ENDSUBMIT statement, except for those within a SAS.SUBMIT code block. You must specify macro value assignments for Lua commands within the SUBMIT statement. This example shows how to assign macro variable values for use within Lua code.

Separate multiple assignments with a semicolon (;) and place all assignments within a single pair of quotation marks. This example prints the following text to the SAS log: Hello, George. Have a nice day.

Program

```
%let g='George';
%let h='Have a nice day.';

proc lua;
  submit "name=&g; msg=&h";
    print('Hello, ' .. name .. ' ' .. msg)
  endsubmit;
run;
```

Program Description

Define macro variable values. The macro variables G and H are assigned to character values. The use of single quotation marks in the macro variable assignment is required for Lua. Single quotation marks are not normally required for macro variable assignments in SAS.

```
%let g='George';  
%let h='Have a nice day.';
```

Execute the PROC LUA statement. The PROC LUA statement enables you to call Lua code within your SAS session.

```
proc lua;
```

Identify the beginning of a block of Lua statements and assign any macro variables. The SUBMIT statement identifies the beginning of a block of Lua statements. This example assigns the values of the macro variables g and h to the local Lua variables name and msg, respectively. No macro expansion occurs between the end of the SUBMIT statement and the beginning of the ENDSUBMIT statement. The Lua code refers to the local Lua variables.

```
submit "name=&g; msg=&h";
```

Execute Lua statements. Enter Lua statements between the SUBMIT and ENDSUBMIT statements. Lua statements are not required to end with a semicolon (;). This example prints text values to the SAS log. Concatenate strings using the '..' operator. Each print statement that you use begins on a new line in the log.

```
print('Hello, ' .. name .. ' ' .. msg)
```

Identify the end of a block of Lua statements with the ENDSUBMIT statement.

```
endsubmit;  
run;
```

Example 10: Submitting SAS Code with Lua Variable Substitutions

Features:	PROC LUA statement
	SUBMIT and ENDSUBMIT statements
	SAS.SUBMIT function
	Variable substitution

Details

This example submits a block of Lua statements. Within the Lua statements, assign a block of SAS code to a local variable. The example then invokes the SAS code with the SAS.SUBMIT function and substitutes variable values.

The example uses the Work.Answer data set as input to the local Lua variable Code.

Figure 34.1 *Work.Answer Data Set*

The SAS System		
Obs	CCUID	Value
1	60	12
2	61	27
3	62	5
4	62	8
5	67	9
6	67	4

Program

```
proc lua;
  submit;

    local rc
    local code = [[
      data sample; set answer;
      where CCUID = @ccuid@;
      y = @subValue@;
      run;
    ]]

    rc = sas.submit(code, {ccuid="67", subValue=72})
  endsubmit;
run;

proc print data=sample;
run;
```

Program Description

Execute the LUA procedure.

```
proc lua;
```

Identify the beginning of the block of Lua statements to execute. Use the SUBMIT statement to identify the Lua statements. Declare a local variable Rc. Semicolons (;) are not required at the end of Lua statements.

```
submit;
```

```
local rc
```

Assign a block of SAS code to a local Lua variable. A block of SAS code, indicated by the [[and]] brackets, is assigned to the local Lua variable Code. The SAS code opens the WORK.ANSWER data set and keeps only records where the value of CCUID matches the value that is specified by @ccuid@. When the SAS code is executed, a value is assigned to the key @ccuid@. A new variable Y is assigned to the value that is specified by @subValue@. The resulting data set is saved to the WORK.SAMPLE data set.

```
local code = [[
  data sample; set answer;
  where CCUID = @ccuid@;
  y = @subValue@;
  run;
]]
```

Execute the SAS code and assign the result to the Lua variable Rc. The SAS.SUBMIT function tells the system to run SAS code. The code block that was assigned to the Code variable is now executed. The call to the SAS.SUBMIT function includes two variable substitutions. When the SAS code executes, the character value "67" is substituted for @ccuid@, and the value 72 is substituted for @subValue@.

```
rc = sas.submit(code, {ccuid="67", subValue=72})
```

The ENDSUBMIT statement identifies the end of the block of Lua statements. The RUN statement completes the call to PROC LUA.

```
endsubmit;
run;
```

Print the resulting data set, Work.Sample.

```
proc print data=sample;
run;
```


Output: SAS Table Resulting from Substituted Lua Values

Output 34.10 *Generated Work.Sample Table*

The SAS System			
Obs	CCUID	Value	y
1	67	9	72
2	67	4	72

Example 11: Using an Iterator Function for a Small Table

Features: SAS.OPEN function
SAS.ROWS function
SAS.CLOSE function

Details

This example traverses a data set and prints the variable names and corresponding values for each observation. Use the SAS.ROWS function when there are relatively few variables.

Program

```
proc lua;
submit;

  local dsid = sas.open("sashelp.class")

  for row in sas.rows(dsid) do
    for n,v in pairs(row) do
      if type(n)=="string" then
        print(n,'=', v)
      end
    end
  end
end
```

```

        sas.close(dsid)
    endsubmit;
run;

```

Program Description

Execute the PROC LUA and SUBMIT statements to begin a block of Lua code.

```

proc lua;
submit;

```

Open the Sashelp.Class SAS data set and load the contents into the Lua variable Dsid. By default, the data set is opened for reading.

```

local dsid = sas.open("sashelp.class")

```

Process the rows of the data set. The outer FOR loop processes each row in the data set. The PAIRS function pulls the variable name and value pairs from each row in the Sashelp.Class data set. The pairs of variable names and values are then printed to the SAS log. Note that in Lua, a double equal sign (==) is used to assess whether two values are equal in the IF condition.

```

for row in sas.rows(dsid) do
    for n,v in pairs(row) do
        if type(n)=="string" then
            print(n,'=', v)
        end
    end
end
end

```

Close the data set, end the block of Lua code, and complete the call to the LUA procedure.

```

        sas.close(dsid)
    endsubmit;
run;

```

Output: ROWS Iterator for a Small Data Set

Output 34.11 *Iterating over a Small Data Set with the ROWS Function (Partial Output)*

```
sex      =      M
height   =      69
name     =      Alfred
weight   =      112.5
age      =      14
sex      =      F
height   =      56.5
name     =      Alice
weight   =      84
age      =      13
sex      =      F
height   =      65.3
name     =      Barbara
weight   =      98
age      =      13
sex      =      F
height   =      62.8
name     =      Carol
weight   =      102.5
age      =      14
...
sex      =      M
height   =      66.5
name     =      William
weight   =      112
age      =      15
NOTE: PROCEDURE LUA used (Total process time):
      real time           0.08 seconds
      cpu time            0.06 seconds
```

Example 12: Using Iterator Functions for a Large Table

Features:

- SAS.OPEN function
- SAS.VARS function
- SAS.NEXT function
- SAS.GET_VALUE function
- SAS.CLOSE function

Details

This example uses the SAS.VARS and SAS.NEXT functions to traverse a SAS data set and print its contents.

Program

```
proc lua;
submit;

    local dsid = sas.open("sashelp.company") -- open for input

    local vars = {}
    -- Iterate over the variables in the data set
    for var in sas.vars(dsid) do
        vars[var.name:lower()] = var
    end

    -- Iterate over the rows of the data set
    local i=0
    while (sas.next(dsid) ~= nil) do
        i=i+1
        print("OBS=" .. i)
        for vname,var in pairs(vars) do
            print(vname, '=', sas.get_value(dsid, vname) )
        end
    end

    sas.close(dsid)
endsubmit;
run;
```

Program Description

Execute the PROC LUA and SUBMIT statements to begin a block of Lua code.

```
proc lua;
submit;
```

Open the Sashelp.Company data set.

```
    local dsid = sas.open("sashelp.company") -- open for input
```

Declare a local Lua array, VARS, and populate it with the values of the variables in the data set. The brackets ({ }) identify an array in Lua. Use the SAS.VARS function to iterate over all the variables in the data set. The example assigns the value of each variable to the Vars array, where the array key is the variable name in lowercase.

```
    local vars = {}
    -- Iterate over the variables in the data set
    for var in sas.vars(dsid) do
        vars[var.name:lower()] = var
    end
```

Iterate over each data set observation and print the values for each variable. The example initializes an iterator variable, I, to 0. The SAS.NEXT iterator function then cycles through the observations in the data set. For each observation, the example prints the variable name and corresponding value for each variable.

```
    -- Iterate over the rows of the data set
    local i=0
```

```

while (sas.next(dsid) ~= nil) do
    i=i+1
    print("OBS=" .. i)
    for vname,var in pairs(vars) do
        print(vname, '=', sas.get_value(dsid, vname) )
    end
end
end

```

Close the data set and end the call to the LUA procedure.

```

    sas.close(dsid)
endsubmit;
run;

```

Output: Iterators with a Large Data Set

Output 34.12 *Using Iterator Functions with a Large Data Set*

```

OBS=1
level4    =    CONTRACTS
job1      =    MANAGER
level3    =    ADMIN
level2    =    TOKYO
n         =    1
depthhead =    1
level1    =    International Ai
level5    =    So Suumi
OBS=2
level4    =    CONTRACTS
job1      =    ASSISTANT
level3    =    ADMIN
level2    =    TOKYO
n         =    1
depthhead =    2
level1    =    International Ai
level5    =    Steffen Graff
...
OBS=48
level4    =    MIS
job1      =    TECH. CONS.
level3    =    TECHN. SERVICES
level2    =    NEW YORK
n         =    1
depthhead =    2
level1    =    International Ai
level5    =    Roy Hobbs
NOTE: PROCEDURE LUA used (Total process time):
      real time           0.23 seconds
      cpu time            0.21 seconds

```

Example 13: Defining and Adding Variables to a Data Set

Features:

- PROC LUA statement
- SAS.OPEN function with 'o' option
- SAS.ADD_VARS function
- SAS.CLOSE function

Details

This example defines new variables within Lua code and adds them to a data set. The example uses the SAS.ADD_VARS function to define the variables and add them to the output data set.

Program

```
proc lua;
submit;

    local dsid = sas.open("work.sample","o")

    sas.add_vars(dsid, { {name="var1", type="N", format="BEST12.2"},
                        {name="var2", type="C", length=500,
format="$char80."},
                        {name="var3", type="C", label="Character
Data"}
                    })

    sas.close(dsid)
endsubmit;
run;
```

Program Description

Execute the PROC LUA statement and begin the block of Lua statements. The PROC LUA statement enables you to call Lua code within your SAS session. The SUBMIT statement identifies the beginning of the block of Lua statements.

```
proc lua;
submit;
```

Open the data set and assign its contents to a local Lua variable. Invoke the SAS.OPEN function to open the WORK.SAMPLE data set. Use the 'o' mode to create the data set if it does not already exist.

```
local dsid = sas.open("work.sample","o")
```

Define variables to add to the data set. This example invokes the SAS.ADD_VARS function to add three variables—Var1, Var2, and Var3—to the data set. The type, numeric (N) or character (C), is specified for each variable. Various additional attributes are assigned to the variables. Only the name attribute is required. For more information, see [“Data Set Functions for the LUA Procedure” on page 1171](#).

```
sas.add_vars(dsid, { {name="var1", type="N", format="BEST12.2"},
                    {name="var2", type="C", length=500,
format="$char80."},
                    {name="var3", type="C", label="Character
Data"}
                })
```

Close the data set and end the call to the LUA procedure. This example executes the SAS.CLOSE function to close the data set that is associated with the Lua variable Dsid.

```
sas.close(dsid)
endsubmit;
run;
```

Output: Adding Variables to a Data Set

Output 34.13 Partial Output from PROC CONTENTS for Work.Sample

Alphabetic List of Variables and Attributes					
#	Variable	Type	Len	Format	Label
1	var1	Num	8	BEST12.2	
2	var2	Char	500	\$CHAR80.	
3	var3	Char	200		Character Data

Example 14: Connecting to the CAS Server

Features:

- CAS.OPEN function
- CLOSE function
- SUBMIT and ENDSUBMIT statements

Details

This example shows how to connect to a CAS server. The host name is **server.mycompany.com**, the port is 5570.

Program

```
proc lua;
submit;

-- Show content of swat_enabled variable
print("NOTE: swat_enabled=", swat_enabled)

-- Show path and cpath info at start
print("NOTE: before")
print("NOTE: package.path=\n" .. package.path)
print("NOTE: package.cpath=\n" .. package.cpath)

-- Add required directories to path and cpath
if swat_enabled == false or swat_enabled == nil then
  print("NOTE: changing path and cpath")
  package.path = package.path ..
    '/opt/sas/viya/home/SASFoundation/misc/casluacInt/luapl/?.lua'
  package.path = package.path ..
    '/opt/sas/viya/home/SASFoundation/misc/casluacInt/luapl/deps/?.lua'
  package.cpath = package.cpath ..
    '/opt/sas/viya/home/SASFoundation/misc/casluacInt/luapl/lib/?.so'

  swat_enabled = true

-- Show updated path and cpath values
print("NOTE: after")
print("NOTE: package.path=\n" .. package.path)
print("NOTE: package.cpath=\n" .. package.cpath)
end

-- Preload library
cas = require "swat"

-- Connect to the CAS server on your host
s=cas.open("server.mycompany.com",5570)

-- Print the session ID
print("NOTE: s=", s)

-- Run the builtins.serverStatus action
-- and display some information
result = s:builtins_serverStatus
print("NOTE: server=\n",result['server'])
```



```

print("NOTE: nodestatus=\n",result['nodestatus'])

-- Run the addFmtLib action to add a format library
s:addFmtLib{fmtlibname="myformats"}

-- Create a format called demo in the myformats library
s:addFormat{fmtLibName="myformats",
             fmtname="demo",
             ranges={"0-17='demo1'", "18-34='demo2'",
                    "35-49='demo3'", "50-64='demo4'"
                    }}

-- Format some values with the newly created format
result = s:listFmtValues{fmtname = "demo",
                        nvals = {15,23,39,61}}

-- Display the results
print(result)

-- Shut down the CAS session
s:close{}

endsubmit;
run;

```

Program Description

Start PROC LUA and load utility resources. Use the concatenation operator to append the middleclass.lua location to the package.path value. Similarly, add the location of the tkluaswat.so file to the package.cpath value. A variable called Swat_enabled is created to prevent updating the package.path and package.cpath values if you run this program more than once. For more information about SWAT, see [“Requirements” in SAS Viya Platform Programming: Getting Started with Lua for CAS](#).

```

proc lua;
submit;

-- Show content of swat_enabled variable
print("NOTE: swat_enabled=", swat_enabled)

-- Show path and cpath info at start
print("NOTE: before")
print("NOTE: package.path=\n" .. package.path)
print("NOTE: package.cpath=\n" .. package.cpath)

-- Add required directories to path and cpath
if swat_enabled == false or swat_enabled == nil then
  print("NOTE: changing path and cpath")
  package.path = package.path ..
    '/opt/sas/viya/home/SASFoundation/misc/casluaclnt/lua/?..lua'
  package.cpath = package.cpath ..
    '/opt/sas/viya/home/SASFoundation/misc/casluaclnt/lua/deps/?..lua'

```

```

package.cpath = package.cpath ..
    '/opt/sas/viya/home/SASFoundation/misc/casluaclnt/luabin/lib/?'

swat_enabled = true

-- Show updated path and cpath values
print("NOTE: after")
print("NOTE: package.path=\n" .. package.path)
print("NOTE: package.cpath=\n" .. package.cpath)
end

```

Load the SWAT library.

```

-- Preload library
cas = require "swat"

```

Connect to the CAS server. Use the CAS.OPEN function to start a CAS session on the CAS server. Assign the CAS session to the Lua variable S. Verify that the connection was successful and print session information. This version of the CAS.OPEN function assumes that you have created an authinfo file. For more information, see [“Connect and Start a Session” in SAS Viya Platform Programming: Getting Started with Lua for CAS](#).

```

-- Connect to the CAS server on your host
s=cas.open("server.mycompany.com",5570)

-- Print the session ID
print("NOTE: s=", s)

-- Run the builtins.serverStatus action
-- and display some information
result = s:builtins_serverStatus
print("NOTE: server=\n",result['server'])
print("NOTE: nodestatus=\n",result['nodestatus'])

```

Run a CAS action. Run the addFmtLib action to add a new format library. Add a new format definition with the addFormat action. Then apply the format to some values and check the output to see the formatted values.

```

-- Run the addFmtLib action to add a format library
s:addFmtLib{fmtlibname="myformats"}

-- Create a format called demo in the myformats library
s:addFormat{fmtLibName="myformats",
    fmtname="demo",
    ranges={"0-17='demo1'", "18-34='demo2'",
        "35-49='demo3'", "50-64='demo4'"
    }}

-- Format some values with the newly created format
result = s:listFmtValues{fmtname = "demo",
    nvals = {15,23,39,61}}

-- Display the results
print(result)

```

Example 15: Running PROC FCMP Functions

Features:	PROC FCMP statement PROC LUA statement FILENAME statement SUBMIT and ENDSUBMIT statements
Restriction:	When you call a function that was created with the FCMP procedure, you can modify only OUTARG arguments that are arrays.

Details

This example calls functions that have been defined using the FCMP procedure, similar to calling standard SAS functions. The function definitions do not need to be included in the same SAS program. PROC FCMP functions can be stored in a function library for use by any program if the CMPLIB system option has been defined within that program. This example defines the SUMX and ADD_SCALAR functions and saves them to the work.myFuncs data set in the ArrayFuncs package. A *package* is a collection of related routines that are specified by a user. The package name groups related functions in the data set that contains the PROC FCMP functions.

Program

```
proc fcmp outlib=work.myFuncs.ArrayFuncs;
function sumx(x[*]);
    sum = 0;
    do i = 1 to dim(x);
        sum = sum + x[i];
    end;
    return(sum );
endsub;

function add_scalar( scalar, x[*] );
    outargs x;
    do i = 1 to dim(x);
        x[i] = x[i] + scalar;
    end;
    return( dim(x) );
endsub;
run;

options cmplib=work.myFuncs;
```

```

proc lua;
submit;
  array = { 1, 2, 3, 4, 5 }
  sum = sas.sumx(array)
  print(sum)
endsubmit;
run;

proc lua;
submit;
  array = { 1, 2, 3, 4, 5 }
  dim = sas.add_scalar(5, array)
  a1 = array[1]
  a2 = array[2]
  print(a1)
  print(a2)
endsubmit;
run;

```

Program Description

Define functions using the FCMP procedure. The SUMX function sums the values of an array and returns that value. The ADD_SCALAR function adds a scalar value to each member of an array. Because the OUTARGS argument for ADD_SCALAR specifies an array, the call to this function using PROC LUA can change the value of the submitted array. The ADD_SCALAR function returns the number of elements in the array.

```

proc fcmp outlib=work.myFuncs.ArrayFuncs;
function sumx(x[*]);
  sum = 0;
  do i = 1 to dim(x);
    sum = sum + x[i];
  end;
  return(sum );
endsub;

function add_scalar( scalar, x[*] );
  outargs x;
  do i = 1 to dim(x);
    x[i] = x[i] + scalar;
  end;
  return( dim(x) );
endsub;
run;

```

Specify the location of the compiled functions. This example finds the previously defined functions in the work.myFuncs data set. The call to the FCMP procedure does not need to be included in the same program with the LUA procedure if the functions are stored. Use the OPTIONS statement to specify the function location before calling the function within the LUA procedure.

```
options cmplib=work.myFuncs;
```

Call the SUMX function within the LUA procedure. You call the SUMX function by preceding the call with “SAS.”, similar to calling any standard SAS function.

```
proc lua;
submit;
  array = { 1, 2, 3, 4, 5 }
  sum = sas.sumx(array)
  print(sum)
endsubmit;
run;
```

Call the ADD_SCALAR function within the LUA procedure. You call the ADD_SCALAR function by preceding the call with “SAS.”, similar to calling any standard SAS function. The calls to SUMX and ADD_SCALAR can come within the same LUA procedure.

```
proc lua;
submit;
  array = { 1, 2, 3, 4, 5 }
  dim = sas.add_scalar(5, array)
  a1 = array[1]
  a2 = array[2]
  print(a1)
  print(a2)
endsubmit;
run;
```

Output: Working with User-Defined Array Functions

Output 34.15 Calling Functions That Were Created By Using PROC FCMP

```

388 proc fcmp outlib=work.myFuncs.ArrayFuncs;
389 function sumx(x[*]);
390     sum = 0;
391     do i = 1 to dim(x);
392         sum = sum + x[i];
393     end;
394     return(sum );
395 endsub;
396
397 function add_scalar( scalar, x[*] );
398     outargs x;
399     do i = 1 to dim(x);
400         x[i] = x[i] + scalar;
401     end;
402     return( dim(x) );
403 endsub;
404 run;

NOTE: Function add_scalar saved to work.myFuncs.ArrayFuncs.
NOTE: Function sumx saved to work.myFuncs.ArrayFuncs.
NOTE: PROCEDURE FCMP used (Total process time):
      real time          1.06 seconds
      cpu time           0.15 seconds

405
406 options cmplib=work.myFuncs;
407
408 proc lua;
409 submit;
410     array = { 1, 2, 3, 4, 5 }
411     sum = sas.sumx(array)
412     print(sum)
413 endsubmit;
414 run;

NOTE: Resuming Lua state from previous PROC LUA invocation.
15
NOTE: PROCEDURE LUA used (Total process time):
      real time          0.30 seconds
      cpu time           0.07 seconds

415
416 proc lua;
417 submit;
418     array = { 1, 2, 3, 4, 5 }
419     dim = sas.add_scalar(5, array)
420     a1 = array[1]
421     a2 = array[2]
422     print(a1)
423     print(a2)
424 endsubmit;
425 run;

NOTE: Resuming Lua state from previous PROC LUA invocation.
6
7
NOTE: PROCEDURE LUA used (Total process time):
      real time          0.05 seconds
      cpu time           0.04 seconds

```


MEANS Procedure

Overview: MEANS Procedure	1227
What Does the MEANS Procedure Do?	1228
What Types of Output Does PROC MEANS Produce?	1229
PROC MEANS and the ODS OUTPUT Statement	1231
Concepts: MEANS Procedure	1231
Using Class Variables	1231
Computational Resources	1233
In-Database Processing for PROC MEANS	1235
Threaded Processing of Input DATA Sets	1236
CAS Processing for PROC MEANS	1237
Syntax: MEANS Procedure	1239
PROC MEANS Statement	1240
ATTRIB Statement	1255
BY Statement	1255
CLASS Statement	1260
FORMAT Statement	1266
FREQ Statement	1266
ID Statement	1267
LABEL Statement	1268
OUTPUT Statement	1270
TYPES Statement	1280
VAR Statement	1282
WAYS Statement	1283
WEIGHT Statement	1284
WHERE Statement	1291
Usage: MEANS Procedure	1293
Computation of Moment Statistics	1293
Confidence Limits	1294
Student's t Test	1294
Quantiles	1295
Results: MEANS Procedure	1296
Missing Values	1296
Column Width for the Output	1296
The N Obs Statistic	1297
Output Data Set	1297
Examples: MEANS Procedure	1299

Example 1: Computing Specific Descriptive Statistics	1299
Example 2: Computing Descriptive Statistics with Class Variables	1301
Example 3: Using the BY Statement with Class Variables	1304
Example 4: Using a CLASSDATA= Data Set with Class Variables	1306
Example 5: Using Multilabel Value Formats with Class Variables	1309
Example 6: Using Preloaded Formats with Class Variables	1314
Example 7: Computing a Confidence Limit for the Mean	1318
Example 8: Computing Output Statistics	1320
Example 9: Computing Different Output Statistics for Several Variables	1322
Example 10: Computing Output Statistics with Missing Class Variable Values ..	1325
Example 11: Identifying an Extreme Value with the Output Statistics	1327
Example 12: Identifying the Top Three Extreme Values with the Output Statistics	1330
Example 13: Using the STACKODSOUTPUT Option to Control Data	1335
References	1341

Overview: MEANS Procedure

What Does the MEANS Procedure Do?

The MEANS procedure provides data summarization tools to compute descriptive statistics for variables across all observations and within groups of observations. For example, PROC MEANS does the following:

- calculates descriptive statistics based on moments
- estimates quantiles, which includes the median
- calculates confidence limits for the mean
- identifies extreme values
- performs a t test

By default, PROC MEANS displays output. You can also use the OUTPUT statement to store the statistics in a SAS data set.

PROC MEANS and PROC SUMMARY are very similar; see [Chapter 60, “SUMMARY Procedure,” on page 2153](#) for an explanation of the differences.

What Types of Output Does PROC MEANS Produce?

PROC MEANS Default Output

Output 1.1 shows the default output that PROC MEANS displays. The data set that PROC MEANS analyzes contains the integers 1 through 10. The output reports the number of observations, the mean, the standard deviation, the minimum value, and the maximum value. The statements that produce the output follow:

```
proc means data=OnetoTen;
run;
```

Output 35.1 *The Default Descriptive Statistics*

The SAS System					1
The MEANS Procedure					
Analysis Variable : Integer					
N	Mean	Std Dev	Minimum	Maximum	
10	5.5000000	3.0276504	1.0000000	10.0000000	

PROC MEANS Customized Output

The following output shows the results of a more extensive analysis of two variables, MoneyRaised and HoursVolunteered. The analysis data set contains information about the amount of money raised and the number of hours volunteered by high-school students for a local charity. PROC MEANS uses six combinations of two categorical variables to compute the number of observations, the mean, and the range. The first variable, School, has two values and the other variable, Year, has three values. For an explanation of the program that produces the output, see [“Example 11: Identifying an Extreme Value with the Output Statistics” on page 1327](#).

Output 35.2 Specified Statistics for Class Levels and Identification of Maximum Values

Summary of Volunteer Work by School and Year							1
The MEANS Procedure							
School	Year	N	Variable	N	Mean	Range	
Kennedy	1992	15	MoneyRaised	15	29.0800000	39.7500000	
			HoursVolunteered	15	22.1333333	30.0000000	
	1993	20	MoneyRaised	20	28.5660000	23.5600000	
			HoursVolunteered	20	19.2000000	20.0000000	
	1994	18	MoneyRaised	18	31.5794444	65.4400000	
			HoursVolunteered	18	24.2777778	15.0000000	
Monroe	1992	16	MoneyRaised	16	28.5450000	48.2700000	
			HoursVolunteered	16	18.8125000	38.0000000	
	1993	12	MoneyRaised	12	28.0500000	52.4600000	
			HoursVolunteered	12	15.8333333	21.0000000	
	1994	28	MoneyRaised	28	29.4100000	73.5300000	
			HoursVolunteered	28	19.1428571	26.0000000	

Best Results: Most Money Raised and Most Hours Worked								2
Obs	School	Year	_TYPE_	_FREQ_	Most Cash	Most Time	Money Raised	Hours Volunteered
1		.	0	109	Willard	Tonya	78.65	40
2		1992	1	31	Tonya	Tonya	55.16	40
3		1993	1	32	Cameron	Amy	65.44	31
4		1994	1	46	Willard	L.T.	78.65	33
5	Kennedy	.	2	53	Luther	Jay	72.22	35
6	Monroe	.	2	56	Willard	Tonya	78.65	40
7	Kennedy	1992	3	15	Thelma	Jay	52.63	35
8	Kennedy	1993	3	20	Bill	Amy	42.23	31
9	Kennedy	1994	3	18	Luther	Che-Min	72.22	33
10	Monroe	1992	3	16	Tonya	Tonya	55.16	40
11	Monroe	1993	3	12	Cameron	Myrtle	65.44	26
12	Monroe	1994	3	28	Willard	L.T.	78.65	33

In addition to the report, the program also creates an output data set (located on page 2 of the output) that identifies the students who raised the most money and who volunteered the most time over all the observations and within the combinations of School and Year:

- The first observation in the data set shows the students with the maximum values overall for MoneyRaised and HoursVolunteered.
- Observations 2 through 4 show the students with the maximum values for each year, regardless of school.
- Observations 5 and 6 show the students with the maximum values for each school, regardless of year.

- Observations 7 through 12 show the students with the maximum values for each school-year combination.

PROC MEANS and the ODS OUTPUT Statement

The template for procedures MEANS and SUMMARY causes ODS to apply default formats and format attributes to its output, overriding any formats associated and applied to variables of the input data set, because the option `USE_FORMAT_DEFAULTS` is specified in the template. This option affects the width and alignment of ODS-formatted output, which can vary from the width and alignment of formats found in the input data set or those specified on a `FORMAT` or `ATTRIB` statement. These format defaults affect ODS output and not internal-numeric representation or calculations.

For example, numeric format outputs can be wider and have more characters than expected when the default-format width is greater than the specified width specified in the data set or the statement.

You can prevent default formats being applied to the output data set by creating your output data set using the `OUTPUT` statement instead of the `ODS OUTPUT` statement.

If you prefer that format defaults not be applied to ODS output, you can modify the `Base.Summary` ODS table template with the following code:

```
proc template;
  edit base.summary;
  use_format_defaults=off;
end;
run;
```

Concepts: MEANS Procedure

Using Class Variables

Using TYPES and WAYS Statements

The `TYPES` statement controls which of the available class variables PROC MEANS uses to subgroup the data. The unique combinations of these active class

variable values that occur together in any single observation of the input data set determine the data subgroups. Each subgroup that PROC MEANS generates for a given type is called a level of that type. Note that for all types, the inactive class variables can still affect the total observation count of the rejection of observations with missing values.

When you use a WAYS statement, PROC MEANS generates types that correspond to every possible unique combination of n class variables chosen from the complete set of class variables. For example

```
proc means;
  class a b c d e;
  ways 2 3;
run;
```

is equivalent to

```
proc means;
  class a b c d e;
  types a*b a*c a*d a*e b*c b*d b*e c*d c*e d*e
        a*b*c a*b*d a*b*e a*c*d a*c*e a*d*e
        b*c*d b*c*e c*d*e;
run;
```

If you omit the TYPES statement and the WAYS statement, then PROC MEANS uses all class variables to subgroup the data (the NWAY type) for displayed output and computes all types (2^k) for the output data set.

Ordering the Class Values

PROC MEANS determines the order of each class variable in any type by examining the order of that class variable in the corresponding one-way type. You see the effect of this behavior in the options ORDER=DATA or ORDER=FREQ. If you specify the ORDER=DATA option for input data in a DBMS table, PROC MEANS computation might produce different results for separate runs of the same analysis. In addition to determining the order of variable levels in crosstabulation table displays, the ORDER= option can also affect the values of many of the test statistics and measures that PROC MEANS computes. When PROC MEANS subdivides the input data set into subsets, the classification process does not apply the options ORDER=DATA or ORDER=FREQ independently for each subgroup. Instead, one frequency and data order is established for all output based on a non-subdivided view of the entire data set. For example, consider the following statements:

```
data pets;
  input Pet $ Gender $;
  datalines;
dog  m
dog  f
dog  f
dog  f
cat  m
cat  m
cat  f
```

```
;
proc means data=pets order=freq;
  class pet gender;
run;
```

The statements produce this output.

Output 35.3 Ordering Class Values

The SAS System			1
The MEANS Procedure			
Pet	Gender	N	

dog	f	3	
	m	1	
cat	f	1	
	m	2	

In the example, PROC MEANS does not list male cats before female cats. Instead, it determines the order of gender for all types over the entire data set. PROC MEANS found more observations for female pets (f=4, m=3). The default for ORDER is ORDER=INTERNAL.

Computational Resources

PROC MEANS uses the same memory allocation scheme across all operating environments. When class variables are involved, PROC MEANS must keep a copy of each unique value of each class variable in memory. You can estimate the memory requirements to group the class variable by calculating

$$Nc_1(Lc_1 + K) + Nc_2(Lc_2 + K) + \dots + Nc_n(Lc_n + K)$$

where

Nc_i
is the number of unique values for the class variable.

Lc_i
is the combined unformatted and formatted length of c_i .

K
is some constant on the order of 32 bytes (64 for 64-bit architectures).

When you use the GROUPINTERNAL option in the CLASS statement, Lc_i is simply the unformatted length of c_i .

Each unique combination of class variables, $c_{1i} c_{2j}$ for a given type forms a level in that type. See “TYPES Statement” on page 1280. You can estimate the maximum potential space requirements for all levels of a given type, when all combinations actually exist in the data (a complete type), by calculating

$$W * N_{c_1} * N_{c_2} * \dots * N_{c_n}$$

where

W

is a constant based on the number of variables analyzed and the number of statistics calculated (unless you request QMETHOD=OS to compute the quantiles).

$N_{c_1} \dots N_{c_n}$

are the number of unique levels for the active class variables of the given type.

Clearly, the memory requirements of the levels overwhelm the levels of the class variables. For this reason, PROC MEANS can open one or more utility files and write the levels of one or more types to disk. These types are either the primary types that PROC MEANS built during the input data scan or the derived types.

If PROC MEANS must write partially complete primary types to disk while it processes input data, then one or more merge passes can be required to combine type levels in memory with the levels on disk. In addition, if you use an order other than DATA for any class variable, then PROC MEANS groups the completed types on disk. For this reason, the peak disk space requirements can be more than twice the memory requirements for a given type.

When PROC MEANS uses a temporary work file, you receive the following note in the SAS log:

```
Processing on disk occurred during summarization.
Peak disk usage was approximately nnn
Mbytes.
Adjusting MEMSIZE or REALMEMSIZE may improve performance.
```

In most cases processing ends normally.

When you specify class variables in a CLASS statement, the amount of data-dependent memory that PROC MEANS uses before it writes to a utility file is controlled by the SAS system option REALMEMSIZE=. The value of REALMEMSIZE= indicates the amount of real as opposed to virtual memory that SAS can expect to allocate. PROC MEANS determines how much data-dependent memory to use before writing to utility files by calculating the lesser of these two values:

- the value of REALMEMSIZE=
- $0.8 * (M - U)$, where M is the value of MEMSIZE= and U is the amount of memory that is already in use

REALMEMSIZE also affects the behavior of other memory intensive PROCs such as PROC SORT.

As an alternative, you can use the PROC option SUMSIZE=. Like the PROC option SORTSIZE=, SUMSIZE= sets the memory threshold where disk-based operations begin. For best results, set SUMSIZE= to less than the amount of real memory that

is likely to be available for the task. For efficiency reasons, PROC MEANS can internally round up the value of SUMSIZE=. SUMSIZE= has no effect unless you specify class variables.

Operating Environment Information: The REALMEMSIZE= SAS system option is not available in all operating environments. For details, see the SAS Companion for your operating environment.

If PROC MEANS reports that there is insufficient memory, then increase SUMSIZE= (or REALMEMSIZE=). A SUMSIZE= (or REALMEMSIZE=) value that is greater than MEMSIZE= has no effect. Therefore, you might also need to increase MEMSIZE=. If PROC MEANS reports insufficient disk space, then increase the WORK space allocation. See the SAS documentation for your operating environment for more information about how to adjust your computation resource parameters.

Another way to enhance performance is by carefully applying the TYPES or WAYS statement, limiting the computations to only those combinations of class variables that you are interested in. In particular, significant resource savings can be achieved by not requesting the combination of all class variables.

In-Database Processing for PROC MEANS

When large data sets are stored in an external database, the transfer of the data sets to computers that run SAS can be impacted by performance, security, and resource management issues. SAS in-database processing can greatly reduce data transfer by having the database perform the initial data aggregation.

In-database processing for PROC MEANS supports the following database management systems:

- Amazon Redshift
- DB2
- Google BigQuery
- Greenplum
- Hadoop
- Impala
- Microsoft SQL Server
- MySQL (as of 2021.1.5)
- Netezza
- Oracle
- PostgreSQL
- SAP HANA
- Snowflake
- Spark
- Teradata

■ Vertica

Under the correct conditions, PROC MEANS generates an SQL query based on the statements that are used and the output statistics that are specified in the PROC step. If class variables are specified, the procedure creates an SQL GROUP BY clause that represents the n-way type. The result set that is created when the aggregation query executes in the database is read by SAS into the internal PROC MEANS data structure, and all subsequent types are derived from the original n-way type to form the final analysis results. When SAS format definitions have been deployed in the database, formatting of class variables occurs in the database. If the SAS format definitions have not been deployed in the database, the in-database aggregation occurs on the raw values, and the relevant formats are applied by SAS as the results' set is merged into the PROC MEANS internal structures. Multi-label formatting is always done by SAS using the initially aggregated result set that is returned by the database. The CLASS, TYPES, WAYS, VAR, BY, FORMAT, and WHERE statements are supported when PROC MEANS is processed inside the database. FREQ, ID, IDMIN, IDMAX, and IDGROUPS are not supported. The following statistics are supported for in-database processing: N, NMISS, MIN, MAX, RANGE, SUM, SUMWGT, MEAN, CSS, USS, VAR, STD, STDERR, PRET, UCLM, LCLM, CLM, and CV.

Weighting for in-database processing is supported only for N, NMISS, MIN, MAX, RANGE, SUM, SUMWGT, and MEAN.

The following statistics are currently not supported for in-database processing: SKEW, KURT, P1, P5, P10, P20, P25/Q1, P30, P40, P50/MEDIAN, P60, P70, P75/Q3, P80, P90, P95, P99, and MODE.

The SQLGENERATION system option or LIBNAME statement option controls whether and how in-database procedures are run inside the database. By default, the in-database procedures are run inside the database when possible. There are many data set options that prevent in-database processing, such as, OBS=, FIRSTOBS=, RENAME=, and DBCONDITION=. For a complete listing, see [“Data Set Options” in SAS In-Database Products: User's Guide](#).

In-database processing can greatly reduce the volume of data transferred to the procedure if there are no class variables (one row is returned) or if the selected class variables have a small number of unique values. However, because PROC MEANS loads the result set into its internal structures, the memory requirements for the SAS process are equivalent to what would have been required without in-database processing. The CPU requirements for the SAS process should be significantly reduced if the bulk of the data summarization occurs inside the database. The real time required for summarization should be significantly reduced because many database-process queries are in parallel.

For more information about in-database processing, see *SAS/ACCESS for Relational Databases: Reference*.

Threaded Processing of Input DATA Sets

The THREADS option enables or disables parallel processing of the input data set. Threaded processing achieves a degree of parallelism in the processing operations.

This parallelism is intended to reduce the real time to completion for a given operation and therefore limit the cost of additional CPU resources. For more information, see [Using Parallel Processing](#).

The value of the SAS system option CPUCOUNT= affects the performance of the threaded sort. CPUCOUNT= suggests how many system CPUs are available for use by the threaded procedures.

For more information see the “THREADS System Option” and “CPUCOUNT= System Option” in the *SAS System Options: Reference*.

Calculated statistics can vary slightly, depending on the order in which observations are processed. Such variations are due to numerical errors that are introduced by floating-point arithmetic, the results of which should be considered approximate and not exact. The order of observation processing can be affected by nondeterministic effects of multithreaded or parallel processing. The order of processing can also be affected by inconsistent or nondeterministic ordering of observations that are produced by a data source, such as a DBMS that delivers query results through an ACCESS engine. For more information, see “[Threading in Base SAS](#)” in *SAS Programmer's Guide: Essentials*

CAS Processing for PROC MEANS

When analyzing a data table that resides on a CAS server, a portion of the work that is performed by PROC MEANS is done within CAS. The work that is done in CAS is similar to work performed within a DBMS by in-database processing. If your input data set originates from CAS, some of the PROC MEANS processing can be performed by the CAS server. Running PROC MEANS with CAS actions has several advantages over processing within SAS. These advantages include reduced network traffic, and the potential for faster processing. Faster processing is possible because in-memory tables are manipulated locally on the server instead of being transferred across a relatively slow network connection. CAS is used because it might have more processing resources.

Under the correct conditions, PROC MEANS generates and executes a CAS action based on the statements that are used and the output statistics that are specified in the PROC step. If class variables are specified, the procedure creates a GROUPBY input table parameter that represents the n -way type. The result set that is created when the action executes on the CAS server is read by SAS into the internal PROC MEANS data structure. All subsequent types are derived from the original n -way type to form the final analysis results.

For intrinsic formats, formatting of class variables occurs in CAS. For user-defined formats, formatting of class variables occurs in CAS if the formats have been defined in or copied to the server. If formats are not available on the CAS server, initial aggregation occurs on the server using raw values, and the relevant formats are applied by SAS as the result set is merged into PROC MEANS internal structure. Multilabel formatting is always done by SAS using the initially aggregated result set that is returned by CAS.

The VARCHAR variable type is not supported in PROC MEANS. Any variables declared as VARCHAR will be converted to CHAR.

The CLASS, TYPES, WAYS, VAR, BY, FORMAT, and WHERE statements are supported when PROC MEANS is processed inside the CAS server. FREQ, ID, IDMIN, IDMAX, and IDGROUPS are not supported.

The following statistics are supported for processing: N, NMISS, MIN, MAX, RANGE, SUM, SUMWGT, MEAN, CSS, USS, VAR, STD, STDERR, PRET, UCLM, LCLM, CLM, and CV.

Weighting for CAS processing is supported only for N, NMISS, MIN, MAX, RANGE, SUM, SUMWGT, and MEAN.

The following statistics are currently not supported in CAS: SKEW, KURT, P1, P5, P10, P20, P25/Q1, P30, P40, P50/MEDIAN, P60, P70, P75/Q3, P80, P90, P95, P99, and MODE.

By default, when the DATA= input data set references an in-memory table or view in CAS, the MEANS procedure runs on the CAS server when possible. There are many data set options that prevent processing in CAS, such as OBS=, FIRSTOBS=, and RENAME=.

Processing in CAS can reduce the volume of data that is transferred to the procedure if there are no class variables (one row is returned) or if the selected class variables have a small number of unique values. However, because PROC MEANS loads the result set into its internal structure, the memory requirements for the SAS process are equivalent to what would have been required when not processing in CAS. The CPU requirements for the SAS process should be significantly reduced if the bulk of the data summarization occurs inside the CAS server. The real-time processing required for summarization should be significantly reduced because CAS can process the data in parallel. If the results of PROC MEANS are directed back to the CAS server using an OUTPUT statement, processing of intermediate aggregates must still be performed by the MEANS procedure in SAS. If you want all processing to be performed in CAS, you can invoke an appropriate action directly, using PROC CAS or one of the many other CAS clients and languages.

Note: Intermediate results are always processed by the client regardless of the final destination—CAS or the client.

Here is an example of how to run PROC MEANS with CAS. The CAS LIBNAME engine is used to connect a SAS 9.4 session to an existing CAS session through the CAS session name or the CAS session UUID. The resulting libref is then used by SAS to communicate with the specific CAS session.

```
/* Connect to a CAS server */
cas casauto host="cloud.example.com" port=5570;
/* Specify the CAS engine LIBNAME statement and use the CAS engine
libref*/
libname mycas cas sessref=casauto;
/* Enable INFO messages so that the SAS log reports the use of CAS
actions. */
options msglevel=i;
/*For demonstration purposes, move Grade to CAS and rename it to
GradeBySection.*/
proc casutil;
load data=Grade casout=GradeBySection;
```

```
quit;
/*Execute the PROC MEANS procedure*/
/* A SORT step is not needed for BY-processing when running in CAS */
proc means data=mycas.GradeBySection min max mean;
  by Section;
  var Score;
  class Status Year;
  title1 'Final Exam Scores for Student Status and Year of
Graduation';
  title2 'Within Each Section';
run;
cas casauto terminate;
```

Syntax: MEANS Procedure

Tip: You can use the ATTRIB, FORMAT, LABEL, and WHERE statements. For more information, see [“Language Concepts” on page 20](#). You can also use any global statement. For a list, see “Global Statements” in [SAS Global Statements: Reference](#).

PROC MEANS *<options>* *<statistic-keyword(s)>*;

ATTRIB *variable-list(s) attribute-list(s)*;

BY *<DESCENDING> variable-1 <<DESCENDING> variable-2 ...> <NOTSORTED>*;

CLASS *variable(s) </ options>*;

FORMAT *variable-1 <...variable-n> <format> <DEFAULT=default-format>*;

FREQ *variable*;

ID *variable(s)*;

LABEL *variable-1=label-1<...variable-n=label-n>*;

OUTPUT *<OUT=SAS-data-set> <output-statistic-specification(s)>*
<id-group-specification(s)> <maximum-id-specification(s)>
<minimum-id-specification(s)> </ options> ;

TYPES *request(s)*;

VAR *variable(s) </ WEIGHT=weight-variable>*;

WAYS *list*;

WEIGHT *variable*;

WHERE *where-expression-1*
< logical-operator where-expression-n>;

Statement	Task	Example
PROC MEANS	Compute descriptive statistics for variables	Ex. 1, Ex. 2, Ex. 3, Ex. 4, Ex. 5, Ex. 6, Ex. 7, Ex. 8, Ex. 9, Ex. 10,

Statement	Task	Example
		Ex. 12, Ex. 13
BY	Calculate separate statistics for each BY group	Ex. 3
CLASS	Identify variables whose values define subgroups for the analysis	Ex. 2, Ex. 3, Ex. 4, Ex. 5, Ex. 6, Ex. 7, Ex. 8, Ex. 9, Ex. 10, Ex. 11, Ex. 12
FREQ	Identify a variable whose values represent the frequency of each observation	
ID	Include additional identification variables in the output data set	
OUTPUT	Create an output data set that contains specified statistics and identification variables	Ex. 8, Ex. 9, Ex. 10, Ex. 11, Ex. 12
TYPES	Identify specific combinations of class variables to use to subdivide the data	Ex. 2, Ex. 5, Ex. 12
VAR	Identify the analysis variables and their order in the results	Ex. 1
WAYS	Specify the number of ways to make unique combinations of class variables	Ex. 6
WEIGHT	Identify a variable whose values weight each observation in the statistical calculations	Ex. 6

PROC MEANS Statement

Computes descriptive statistics for variables.

Supports: Threaded processing

See: [Chapter 60, “SUMMARY Procedure,” on page 2153](#)

Examples:

- [“Example 1: Computing Specific Descriptive Statistics” on page 1299](#)
- [“Example 2: Computing Descriptive Statistics with Class Variables” on page 1301](#)
- [“Example 3: Using the BY Statement with Class Variables” on page 1304](#)
- [“Example 4: Using a CLASSDATA= Data Set with Class Variables” on page 1306](#)
- [“Example 5: Using Multilabel Value Formats with Class Variables” on page 1309](#)
- [“Example 6: Using Preloaded Formats with Class Variables” on page 1314](#)
- [“Example 7: Computing a Confidence Limit for the Mean” on page 1318](#)
- [“Example 8: Computing Output Statistics” on page 1320](#)

[“Example 9: Computing Different Output Statistics for Several Variables” on page 1322](#)

[“Example 10: Computing Output Statistics with Missing Class Variable Values” on page 1325](#)

[“Example 11: Identifying an Extreme Value with the Output Statistics” on page 1327](#)

[“Example 12: Identifying the Top Three Extreme Values with the Output Statistics” on page 1330](#)

[“Example 13: Using the STACKODSOUTPUT Option to Control Data” on page 1335](#)

Syntax

PROC MEANS *<options>* *<statistic-keyword(s)>*;

Summary of Optional Arguments

DATA=SAS-data-set

specifies the input data set.

INCAS=(YES | NO)

specifies whether to allow in-CAS processing.

NOTRAP

disables floating point exception recovery.

SUMSIZE=value

specifies the amount of memory to use for data summarization with class variables.

THREADS

NOTHREADS

overrides the SAS system option **THREADS** | **NOTHREADS**.

Control the classification levels

CLASSDATA=SAS-data-set

specifies a secondary data set that contains the combinations of class variables to analyze.

COMPLETETYPES

creates all possible combinations of class variable values.

EXCLUSIVE

excludes from the analysis all combinations of class variable values that are not in the **CLASSDATA=** data set.

MISSING

uses missing values as valid values to create combinations of class variables.

Control the output

FW=field-width

specifies the field width for the statistics.

MAXDEC=number

specifies the number of decimal places for the statistics.

NOLABEL

suppresses variable labels.

NONOBS

suppresses reporting the total number of observations for each unique combination of the class variables.

NOPRINT

suppresses all displayed output.

ORDER=DATA | FORMATTED | FREQ | UNFORMATTED

orders the values of the class variables according to the specified order.

PRINT

displays the output.

PRINTALLTYPES

displays the analysis for all requested combinations of class variables.

PRINTIDVARS

displays the values of the ID variables.

STACKODSOUTPUT

produces an ODS output object

Control the output data set

CHARTYPE

specifies that the _TYPE_ variable contain character values.

DESCENDTYPES

orders the output data set by descending _TYPE_ value.

IDMIN

selects ID variables based on minimum values.

NWAY

limits the output statistics to the observations with the highest _TYPE_ value.

Control the statistical analysis

ALPHA=value

specifies the confidence level for the confidence limits.

EXCLNPWGT

excludes observations with nonpositive weights from the analysis.

QMARKERS=number

specifies the sample size to use for the P2 quantile estimation method.

QMETHOD=OS | P2

specifies the quantile estimation method.

QNTLDEF=1 | 2 | 3 | 4 | 5

specifies the mathematical definition used to compute quantiles.

statistic-keyword(s)

selects the statistics.

VARDEF=DF | N | WDF | WEIGHT

specifies the variance divisor.

Optional Arguments

ALPHA=*value*

specifies the confidence level to compute the confidence limits for the mean. The percentage for the confidence limits is $(1 - \text{value}) \times 100$. An example is (ALPHA=.05 results in a 95% confidence limit).

Default .05

Range between 0 and 1

Interaction To compute confidence limits specify the *statistic-keyword* CLM, LCLM, or UCLM.

See [“Confidence Limits” on page 1294](#)

[“Example 7: Computing a Confidence Limit for the Mean” on page 1318](#)

CHARTYPE

specifies that the _TYPE_ variable in the output data set is a character representation of the binary value of _TYPE_. The length of the variable equals the number of class variables.

Interaction When you specify more than 32 class variables, _TYPE_ automatically becomes a character variable.

See [“Output Data Set” on page 1297](#)

[“Example 10: Computing Output Statistics with Missing Class Variable Values” on page 1325](#)

CLASSDATA=*SAS-data-set*

specifies a data set that contains the combinations of values of the class variables that must be present in the output. Any combinations of values of the class variables that occur in the CLASSDATA= data set but not in the input data set appear in the output and have a frequency of zero.

Restriction The CLASSDATA= data set must contain all class variables. Their data type and format must match the corresponding class variables in the input data set.

Interaction If you use the EXCLUSIVE option, then PROC MEANS excludes any observation in the input data set whose combination of class variables is not in the CLASSDATA= data set.

Tip Use the CLASSDATA= data set to filter or to supplement the input data set.

See [“Example 4: Using a CLASSDATA= Data Set with Class Variables” on page 1306](#)

COMPLETETYPES

creates all possible combinations of class variables even if the combination does not occur in the input data set.

- Interaction The PRELOADFMT option in the CLASS statement ensures that PROC MEANS writes all user-defined format ranges or values for the combinations of class variables to the output, even when a frequency is zero.
- Tip Using COMPLETETYPES does not increase the memory requirements.
- See [“Example 6: Using Preloaded Formats with Class Variables” on page 1314](#)

DATA=SAS-data-set

identifies the input SAS data set.

See [“Input Data Sets” on page 24](#)

DESCENDTYPES

orders observations in the output data set by descending _TYPE_ value.

Aliases DESCENDING
 DESCEND

- Interaction Descending has no effect if you specify NWAY.
- Tip Use DESCENDTYPES to make the overall total (_TYPE_=0) the last observation in each BY group.
- See [“Output Data Set” on page 1297](#)
[“Example 9: Computing Different Output Statistics for Several Variables” on page 1322](#)

EXCLNPWGT

excludes observations with nonpositive weight values (zero or negative) from the analysis. By default, PROC MEANS treats observations with negative weights like observations with zero weights and counts them in the total number of observations.

Alias EXCLNPWGTS

See [“WEIGHT Statement” on page 1284](#)

[WEIGHT= option on the VAR statement on page 1282](#)

EXCLUSIVE

excludes from the analysis all combinations of the class variables that are not found in the CLASSDATA= data set.

- Requirement If a CLASSDATA= data set is not specified, then this option is ignored.
- See [“Example 4: Using a CLASSDATA= Data Set with Class Variables” on page 1306](#)

FW=field-width

specifies the field width to display the statistics in printed or displayed output. FW= has no effect on statistics that are saved in an output data set.

Default 12

Tip If PROC MEANS truncates column labels in the output, then increase the field width.

See [“Example 1: Computing Specific Descriptive Statistics” on page 1299](#)

[“Example 4: Using a CLASSDATA= Data Set with Class Variables” on page 1306](#)

[“Example 5: Using Multilabel Value Formats with Class Variables” on page 1309](#)

INCAS=(YES | NO)

specifies whether to allow in-CAS processing.

YES

Use in-CAS processing. YES is the default.

NO

Do not use in-CAS processing.

IDMIN

specifies that the output data set contain the minimum value of the ID variables.

Interaction Specify PRINTIDVARS to display the value of the ID variables in the output.

See [“ID Statement” on page 1267](#)

MAXDEC=number

specifies the maximum number of decimal places to display the statistics in the printed or displayed output. MAXDEC= has no effect on statistics that are saved in an output data set.

Default BEST. width for columnar format, typically about 7.

Range 0-8

See [“Example 2: Computing Descriptive Statistics with Class Variables” on page 1301](#)

[“Example 4: Using a CLASSDATA= Data Set with Class Variables” on page 1306](#)

MISSING

considers missing values as valid values to create the combinations of class variables. Special missing values that represent numeric values (the letters A through Z and the underscore (_) character) are each considered as a separate value.

Default If you omit MISSING, then PROC MEANS excludes the observations with a missing class variable value from the analysis.

See *SAS Programmer's Guide: Essentials* for a discussion of missing values that have special meaning.

[“Example 6: Using Preloaded Formats with Class Variables” on page 1314](#)

NOLABEL

suppresses variable labels.

```
Example  proc means data=sashelp.shoes maxdec=2 nolabels;
          var sales returns;
          run;
```

NONOBS

suppresses the column that displays the total number of observations for each unique combination of the values of the class variables. This column corresponds to the _FREQ_ variable in the output data set.

See [“The N Obs Statistic” on page 1297](#)

[“Example 5: Using Multilabel Value Formats with Class Variables” on page 1309](#)

[“Example 6: Using Preloaded Formats with Class Variables” on page 1314](#)

NOPRINT

suppresses all the output. See the `PRINT` option.

Tip Use NOPRINT when you want to create only an OUT= output data set.

NOTRAP

disables floating point exception (FPE) recovery during data processing. By default, PROC MEANS traps these errors and sets the statistic to missing.

In operating environments where the overhead of FPE recovery is significant, NOTRAP can improve performance. Note that normal SAS FPE handling is still in effect so that PROC MEANS terminates in the case of math exceptions.

NWAY

specifies that the output data set contain only statistics for the observations with the highest _TYPE_ and _WAY_ values. When you specify class variables, NWAY corresponds to the combination of all class variables.

Interaction If you specify a TYPES statement or a WAYS statement, then PROC MEANS ignores this option.

See [“Output Data Set” on page 1297](#)

[“Example 10: Computing Output Statistics with Missing Class Variable Values” on page 1325](#)

ORDER=DATA | FORMATTED | FREQ | UNFORMATTED

specifies the sort order to create the unique combinations for the values of the class variables in the output, where

DATA

orders values according to their order in the input data set.

Interaction If you use PRELOADFMT in the CLASS statement, then the order for the values of each class variable matches the order that PROC FORMAT uses to store the values of the associated user-defined format. If you use the CLASSDATA= option, then PROC MEANS uses the order of the unique values of each class variable in the CLASSDATA= data set to order the output levels. If you use both options, then PROC MEANS first uses the user-defined formats to order the output. If you omit EXCLUSIVE, then PROC MEANS appends after the user-defined format and the CLASSDATA= values the unique values of the class variables in the input data set based on the order in which they are encountered.

Tip By default, PROC FORMAT stores a format definition in sorted order. Use the NOTSORTED option to store the values or ranges of a user-defined format in the order in which you define them.

FORMATTED

orders values by their ascending formatted values. This order depends on your operating environment.

Aliases FMT

EXTERNAL

FREQ

orders values by descending frequency count so that levels with the most observations are listed first.

Interactions For multiway combinations of the class variables, PROC MEANS determines the order of a class variable combination from the individual class variable frequencies.

Use the ASCENDING option in the CLASS statement to order values by ascending frequency count.

UNFORMATTED

orders values by their unformatted values, which yields the same order as PROC SORT. This order depends on your operating environment.

Aliases UNFMT

INTERNAL

Default UNFORMATTED

See [“Ordering the Class Values” on page 1232](#)

PRINT

specifies whether PROC MEANS displays the statistical analysis. NOPRINT suppresses all the output.

Default PRINT

Tip Use NOPRINT when you want to create only an OUT= output data set.

See [“Example 8: Computing Output Statistics” on page 1320](#)

[“Example 12: Identifying the Top Three Extreme Values with the Output Statistics” on page 1330](#)

PRINTALLTYPES

displays all requested combinations of class variables (all _TYPE_ values) in the printed or displayed output. Normally, PROC MEANS shows only the NWAY type.

Alias PRINTALL

Interaction If you use the NWAY option, the TYPES statement, or the WAYS statement, then PROC MEANS ignores this option.

See [“Example 4: Using a CLASSDATA= Data Set with Class Variables” on page 1306](#)

PRINTIDVARS

displays the values of the ID variables in printed or displayed output.

Alias PRINTIDS

Interaction Specify IDMIN to display the minimum value of the ID variables.

See [“ID Statement” on page 1267](#)

QMARKERS=number

specifies the default number of markers to use for the P^2 quantile estimation method. The number of markers controls the size of fixed memory space.

Default The default value depends on which quantiles you request. For the median (P50), *number* is 7. For the quantiles (P25 and P50), *number* is 25. For the quantiles P1, P5, P10, P75 P90, P95, or P99, *number* is 105. If you request several quantiles, then PROC MEANS uses the largest value of *number*.

Range an odd integer greater than 3

Tip Increase the number of markers above the defaults settings to improve the accuracy of the estimate; reduce the number of markers to conserve memory and computing time.

See [“Quantiles” on page 1295](#)

QMETHOD=OS | P2

specifies the method that PROC MEANS uses to process the input data when it computes quantiles. If the number of observations is less than or equal to the

QMARKERS= value and QNTLDEF=5, then both methods produce the same results.

OS

uses order statistics. This method is the same method that PROC UNIVARIATE uses.

Note: This technique can be very memory-intensive.

P2

uses the P^2 method to approximate the quantile.

Default OS

Restriction When QMETHOD=P2, PROC MEANS does not compute MODE or weighted quantiles.

Tip When QMETHOD=P2, reliable estimations of some quantiles (P1,P5,P95,P99) might not be possible for some data sets.

See [“Quantiles” on page 1295](#)

QNTLDEF=1 | 2 | 3 | 4 | 5

specifies the mathematical definition that PROC MEANS uses to calculate quantiles when QMETHOD=OS. To use QMETHOD=P2, you must use QNTLDEF=5.

Alias PCTLDEF=

Default 5

See [“Quantile and Related Statistics” on page 2432](#)

statistic-keyword(s)

specifies which statistics to compute and the order to display them in the output. The available keywords in the PROC statement are listed in [“Statistic Keywords” on page 1252](#).

Default N, MEAN, STD, MIN, and MAX

Requirement To compute standard error, confidence limits for the mean, and the Student's *t*-test, you must use the default value of the VARDEF= option, which is DF. To compute skewness or kurtosis, you must use VARDEF=N or VARDEF=DF.

Tip Use CLM or both LCLM and UCLM to compute a two-sided confidence limit for the mean. Use only LCLM or UCLM, to compute a one-sided confidence limit.

See The definitions of the keywords and the formulas for the associated statistics are listed in [“Keywords and Formulas” on page 2426](#).

[“Example 1: Computing Specific Descriptive Statistics” on page 1299](#)

[“Example 3: Using the BY Statement with Class Variables” on page 1304](#)

STACKODSOUTPUT

produces an ODS output object whose data set resembles the printed output.

The STACKODSOUTPUT option affects output data sets created by ODS OUTPUT statements, not the PROC MEANS OUTPUT statement.

Alias STACKODS

See [“Example 13: Using the STACKODSOUTPUT Option to Control Data” on page 1335](#)

SUMSIZE=*value*

specifies the amount of memory that is available for data summarization when you use class variables. *value* might be one of the following:

n

nK

nM

nG

specifies the amount of memory available in bytes, kilobytes, megabytes, or gigabytes, respectively. If *n* is 0, then PROC MEANS use the value of the SAS system option SUMSIZE=.

MAXIMUM

MAX

specifies the maximum amount of memory that is available.

Default The value of the SUMSIZE= system option.

Note Specifying SUMSIZE=0 enables PROC MEANS to use the preferred global REALMEMSIZE option.

Tip For best results, do not make SUMSIZE= larger than the amount of physical memory that is available for the PROC step. If additional space is needed, then PROC MEANS uses utility files.

See The SAS system option SUMSIZE= in *SAS System Options: Reference*.

THREADS | NOTHEADS

enables or disables parallel processing of the input data set. This option overrides the SAS system option THREADS | NOTHEADS unless the system option is restricted. (See Restriction.) For more information, see [Using Parallel Processing](#).

Default value of SAS system option THREADS | NOTHEADS.

Restriction Your site administrator can create a restricted options table. A restricted options table specifies SAS system option values that are established at start-up and cannot be overridden. If the THREADS |

NOTHREADS system option is listed in the restricted options table, any attempt to set these system options is ignored and a warning message is written to the SAS log.

- Interaction** PROC MEANS honors the SAS system option THREADS except when a BY statement is specified or the value of the SAS system option CPUCOUNT is less than 2. You can use THREADS in the PROC MEANS statement to force PROC MEANS to use parallel processing in these situations.
- Note** If THREADS is specified (either as a SAS system option or in the PROC MEANS statement) and another program has the input data set open for reading, writing, or updating, then PROC MEANS might fail to open the input data set. In this case, PROC MEANS stops processing and writes a message to the SAS log.

VARDEF=DF | N | WDF | WEIGHT

specifies the divisor to use in the calculation of the variance and standard deviation. The following table shows the possible values for *divisor* and associated divisors.

Table 35.1 Possible Values for VARDEF=

Value	Divisor	Formula for Divisor
DF	Degrees of freedom	$n - 1$
N	Number of observations	n
WDF	Sum of weights minus one	$(\sum_i w_i) - 1$
WEIGHT WGT	Sum of weights	$\sum_i w_i$

The procedure computes the variance as $CSS/divisor$, where CSS is the corrected sums of squares and equals $\sum (x_i - \bar{x})^2$. When you weight the analysis variables, CSS equals $\sum w_i(x_i - \bar{x}_w)^2$, where $\bar{x}_w$ is the weighted mean.

- Default** DF
- Requirement** To compute the standard error of the mean, confidence limits for the mean, or the Student's t -test, use the default value of VARDEF=.
- Tips** When you use the WEIGHT statement and VARDEF=DF, the variance is an estimate of σ^2 , where the variance of the i th observation is $var(x_i) = \sigma^2/w_i$ and w_i is the weight for the i th observation. This method yields an estimate of the variance of an observation with unit weight.
- When you use the WEIGHT statement and VARDEF=WGT, the computed variance is asymptotically (for large n) an estimate of

$\sigma^2/\bar{w}$, where $\bar{w}$ is the average weight. This method yields an asymptotic estimate of the variance of an observation with average weight.

See [“Keywords and Formulas” on page 2426](#)

[“WEIGHT Statement” on page 1284](#)

Statistic Keywords

CLM

is a descriptive statistic keyword. For details, see [“*statistic-keyword\(s\)*” on page 1249](#).

CSS

is a descriptive statistic keyword. For details, see [“*statistic-keyword\(s\)*” on page 1249](#).

KURTOSIS

is a descriptive statistic keyword. For details, see [“*statistic-keyword\(s\)*” on page 1249](#).

Alias KURT

LCLM

is a descriptive statistic keyword. For details, see [“*statistic-keyword\(s\)*” on page 1249](#).

MAX

is a descriptive statistic keyword. For details, see [“*statistic-keyword\(s\)*” on page 1249](#).

MEAN

is a descriptive statistic keyword. For details, see [“*statistic-keyword\(s\)*” on page 1249](#).

MIN

is a descriptive statistic keyword. For details, see [“*statistic-keyword\(s\)*” on page 1249](#).

MODE

is a descriptive statistic keyword. For details, see [“*statistic-keyword\(s\)*” on page 1249](#).

N

is a descriptive statistic keyword. For details, see [“*statistic-keyword\(s\)*” on page 1249](#).

NMISS

is a descriptive statistic keyword. For details, see [“*statistic-keyword\(s\)*” on page 1249](#).

RANGE

is a descriptive statistic keyword. For details, see [“*statistic-keyword\(s\)*” on page 1249](#).

SKEWNESS

is a descriptive statistic keyword. For details, see “[statistic-keyword\(s\)](#)” on page 1249.

Alias SKEW

STDDEV

is a descriptive statistic keyword. For details, see “[statistic-keyword\(s\)](#)” on page 1249.

Alias STD

STDERR

is a descriptive statistic keyword. For details, see “[statistic-keyword\(s\)](#)” on page 1249.

SUM

is a descriptive statistic keyword. For details, see “[statistic-keyword\(s\)](#)” on page 1249.

SUMWGT

is a descriptive statistic keyword. For details, see “[statistic-keyword\(s\)](#)” on page 1249.

UCLM

is a descriptive statistic keyword. For details, see “[statistic-keyword\(s\)](#)” on page 1249.

USS

is a descriptive statistic keyword. For details, see “[statistic-keyword\(s\)](#)” on page 1249.

VAR

is a descriptive statistic keyword. For details, see “[statistic-keyword\(s\)](#)” on page 1249.

MEDIAN

is a quantile statistic keyword. For details, see “[statistic-keyword\(s\)](#)” on page 1249.

Alias P50

P1

is a quantile statistic keyword. For details, see “[statistic-keyword\(s\)](#)” on page 1249.

P5

is a quantile statistic keyword. For details, see “[statistic-keyword\(s\)](#)” on page 1249.

P10

is a quantile statistic keyword. For details, see “[statistic-keyword\(s\)](#)” on page 1249.

P20

is a quantile statistic keyword. For details, see “[statistic-keyword\(s\)](#)” on page 1249.

P40

is a quantile statistic keyword. For details, see “[statistic-keyword\(s\)](#)” on page 1249.

P70

is a quantile statistic keyword. For details, see “[statistic-keyword\(s\)](#)” on page 1249.

Q1

is a quantile statistic keyword. For details, see “[statistic-keyword\(s\)](#)” on page 1249.

Alias P25

Q3

is a quantile statistic keyword. For details, see “[statistic-keyword\(s\)](#)” on page 1249.

Alias P75

P90

is a quantile statistic keyword. For details, see “[statistic-keyword\(s\)](#)” on page 1249.

P95

is a quantile statistic keyword. For details, see “[statistic-keyword\(s\)](#)” on page 1249.

P99

is a quantile statistic keyword. For details, see “[statistic-keyword\(s\)](#)” on page 1249.

P30

is a quantile statistic keyword. For details, see “[statistic-keyword\(s\)](#)” on page 1249.

P60

is a quantile statistic keyword. For details, see “[statistic-keyword\(s\)](#)” on page 1249.

P80

is a quantile statistic keyword. For details, see “[statistic-keyword\(s\)](#)” on page 1249.

QRANGE

is a quantile statistic keyword. For details, see “[statistic-keyword\(s\)](#)” on page 1249.

PROBT

is a hypothesis testing keyword. For details, see “[statistic-keyword\(s\)](#)” on page 1249.

Alias PBT

T

is a hypothesis testing keyword. For details, see “[statistic-keyword\(s\)](#)” on page 1249.

ATTRIB Statement

Associates a format, informat, label, and length with one or more variables.

Syntax

ATTRIB *variable-list(s) attribute-list(s);*

Arguments

variable-list(s)

names the variables that you want to associate with the attributes.

TIP List the variables in any form that SAS allows.

attribute-list(s)

specifies one or more attributes to assign to *variable-list*. Specify one or more of these attributes in the ATTRIB statement:

FORMAT=*format*

associates a format with variables in *variable-list*.

Tip The format can be either a standard SAS format or a format that is defined with the FORMAT procedure.

INFORMAT=*informat*

associates an informat with variables in *variable-list*.

Tip The informat can be either a standard SAS informat or an informat that is defined with the FORMAT procedure.

LABEL=*'label'*

associates a label with variables in *variable-list*.

See Also

[“ATTRIB Statement” in SAS DATA Step Statements: Reference.](#)

BY Statement

Orders the output according to the BY groups.

Restriction: Do not specify a group variable in the BY statement. The BY variable cannot be the same as the group variable. A *group variable* is a variable in the input data set that is used to categorize chart variable values into groups. For more information, see “Group Variables” on page 1760.

Syntax

```
BY <DESCENDING> variable-1
    <... <DESCENDING> variable-n>
    <NOTSORTED>;
```

Required Argument

variable

specifies the variable that the procedure uses to form BY groups. You can specify more than one variable. If you do not use the NOTSORTED option in the BY statement, then either the observations in the data set must be sorted by all the variables that you specify, or they must be indexed appropriately. Variables in a BY statement are called *BY variables*.

Optional Arguments

DESCENDING

specifies that the observations are sorted in descending order by the variable that immediately follows the word DESCENDING in the BY statement.

NOTSORTED

specifies that observations are not necessarily sorted in alphabetic or numeric order. The observations are grouped in another way (for example, chronological order).

The requirement for ordering or indexing observations according to the values of BY variables is suspended for BY-group processing when you use the NOTSORTED option. In fact, the procedure does not use an index if you specify NOTSORTED. The procedure defines a BY group as a set of contiguous observations that have the same values for all BY variables. If observations with the same values for the BY variables are not contiguous, then the procedure treats each contiguous set as a separate BY group.

You cannot use the NOTSORTED option in a PROC SORT step.

Note: You cannot use the GROUPFORMAT option, which is available in the BY statement in a DATA step, in a BY statement in any PROC step.

Details

Using the BY Statement with the SAS System Option NOBYLINE

If you use the BY statement with the SAS system option NOBYLINE, which suppresses the BY line that normally appears in output that is produced with BY-group processing, then PROC MEANS always starts a new page for each BY group. This behavior ensures that if you create customized BY lines by putting BY-group information in the title and suppressing the default BY lines with NOBYLINE, then the information in the titles matches the report on the pages. See [“Creating Titles That Contain BY-Group Information” on page 55](#).

BY-Group Processing

Procedures create output for each BY group. For example, the elementary statistics procedures and the scoring procedures perform separate analyses for each BY group. The reporting and ODS procedures produce a report for each BY group.

Note: All Base SAS procedures except PROC PRINT process BY groups independently. PROC PRINT can report the number of observations in each BY group as well as the number of observations in all BY groups. Similarly, PROC PRINT can sum numeric variables in each BY group and across all BY groups.

You can use only one BY statement in each PROC step. When you use a BY statement, the procedure expects an input data set that is sorted by the order of the BY variables or one that has an appropriate index. If your input data set does not meet these criteria, then an error occurs. Either sort it with the SORT procedure or create an appropriate index on the BY variables.

Depending on the order of your data, you might need to use the NOTSORTED or DESCENDING option in the BY statement in the PROC step.

- For more information about the BY statement, see [SAS DATA Step Statements: Reference](#).
- For more information about PROC SORT, see [Chapter 55, “SORT Procedure,” on page 2039](#).
- For more information about BY-group processing in CAS, see [“Grouping and Ordering” in SAS Cloud Analytic Services: DATA Step Programming](#).
- For more information about creating indexes, see [“INDEX CREATE Statement” on page 440](#).

Formatting BY-Variable Values

When a procedure is submitted with a BY statement, the following actions are taken with respect to processing of BY groups:

- 1 The procedure determines whether the data is sorted by the internal (unformatted) values of the BY variable(s).

- 2 The procedure determines whether a format has been applied to the BY variable(s). If the BY variable is numeric and has no user-applied format, then the BEST12. format is applied for the purpose of BY-group processing.
- 3 The procedure continues adding observations to the current BY group until both the internal and formatted values of the BY variable or variables change.

This process can have unexpected results if, for example, nonconsecutive internal BY values share the same formatted value. In this case, the formatted value is represented in different BY groups. Alternatively, if different consecutive internal BY values share the same formatted value, then these observations are grouped into the same BY group.

BY Variables That Have Different Lengths in Two Data Sets

When a procedure has a BY statement and two input data sets, one of the input data sets is called the primary data set and the other is called the secondary data set. The primary data set is usually, but not always, the DATA= data set. A BY statement always applies to the primary data set. The variables in the BY statement must appear in the primary data set.

Each procedure determines whether a BY statement applies to a secondary data set, and performs one of the following actions:

- The procedure might always apply the BY statement to the secondary data set. In this case, one or more variables in the BY statement must appear in the secondary data set.
- The procedure might never apply the BY statement to the secondary data set. In this case, the variables in the BY statement are not required in the secondary data set.
- The procedure might check whether the BY variables are in the secondary data set. If none of the BY variables are in the secondary data set, then BY processing does not occur for the secondary data set. If one or more of the BY variables are in the secondary data set, and they match the BY variables in the primary data set, then BY processing is done for the secondary data set. If some but not all BY variables are in the secondary data set, then the procedure might issue an error message and quit. Or, it might take some other action described in the documentation for that particular procedure.

If the BY statement is applied to the secondary data set, then each BY variable that exists on both the data sets must have the same type, character or numeric, in both data sets. The BY variables are required to have either the same formatted value or the same unformatted value. Formatted values match only if both the formatted lengths and the formatted values are the same. Unformatted values are not required to have the same length in order to match. The unformatted character values match if the unformatted values are the same after stripping the trailing blanks. The unformatted doubles match if they have the same value.

A secondary data set does not need to have all of the BY variables that are in the primary data set. A procedure can define a subset of the BY variables for the secondary data set. For example, if the primary data set has the BY variables A,B,C,D, then the procedure can define the following BY variables on the secondary data set:

- A
- A,B
- A,B,C
- A,B,C,D

If both the primary and secondary data sets have the same number of BY variables, and all the BY variables have the same byte lengths and format lengths, then either the unformatted values or the formatted values in the BY buffer (for all of the BY variables) have to match. If they do not match, then each variable is compared. The formatted values of each variable are compared first. The formatted lengths have to match, and the formatted values have to match. If the formatted lengths and values do not match, then the unformatted values are compared even if the byte lengths are different.

If corresponding character variable lengths differ, then the longer character variable can contain only trailing blanks for the extra characters. If the lengths of the character variables are different, then the values match as long as they are the same after stripping the trailing blanks. For example, 'ABCD' in the primary data set matches 'ABCD ' in the secondary data set. If the secondary data set contained 'ABCDEF', then they would not match.

Base SAS Procedures That Support the BY Statement

COMPARE	SORT (required)
CORR	STANDARD
FREQ	SUMMARY
MEANS	TABULATE
PRINT	TRANPOSE
RANK	UNIVARIATE
REPORT (nonwindowing environment only)	ODSTEXT
ODSLIST	ODSTABLE

Note: In the SORT procedure, the BY statement specifies how to sort the data. In the other procedures, the BY statement specifies how the data is currently sorted.

See Also

[“BY Statement” in SAS DATA Step Statements: Reference](#)

CLASS Statement

Specifies the variables whose values define the subgroup combinations for the analysis.

Alias: CLASSES

Note: CLASS statements without options use ORDER=INTERNAL, which is the default, or the value specified by the ORDER= option in the PROC MEANS statement. For example, in the following code, variables c and d would use ORDER=INTERNAL. If an ORDER= option had been specified in the PROC MEANS statement, then variables c and d would use the value specified by the ORDER= option in the PROC MEANS statement.

```
class a b / order=data;
class c d;
```

Tips: You can use multiple CLASS statements.

Some CLASS statement options are also available in the PROC MEANS statement. They affect all CLASS variables. Options that you specify in a CLASS statement apply only to the variables in that CLASS statement.

See: For information about how the CLASS statement groups formatted values, see [“Formatted Values” on page 62](#).

Examples: [“Example 2: Computing Descriptive Statistics with Class Variables” on page 1301](#)
[“Example 3: Using the BY Statement with Class Variables” on page 1304](#)
[“Example 4: Using a CLASSDATA= Data Set with Class Variables” on page 1306](#)
[“Example 5: Using Multilabel Value Formats with Class Variables” on page 1309](#)
[“Example 6: Using Preloaded Formats with Class Variables” on page 1314](#)
[“Example 7: Computing a Confidence Limit for the Mean” on page 1318](#)
[“Example 8: Computing Output Statistics” on page 1320](#)
[“Example 9: Computing Different Output Statistics for Several Variables” on page 1322](#)
[“Example 10: Computing Output Statistics with Missing Class Variable Values” on page 1325](#)
[“Example 11: Identifying an Extreme Value with the Output Statistics” on page 1327](#)
[“Example 12: Identifying the Top Three Extreme Values with the Output Statistics” on page 1330](#)

Syntax

CLASS *variable(s)* *</ options>*;

Required Argument

variable(s)

specifies one or more variables that the procedure uses to group the data. Variables in a CLASS statement are referred to as *class variables*. Class variables are numeric or character. Class variables can have continuous values, but they typically have a few discrete values that define levels of the variable. You do not have to sort the data by class variables.

Interaction Use the TYPES statement or the WAYS statement to control which class variables PROC MEANS uses to group the data.

Tip To reduce the number of class variable levels, use a FORMAT statement to combine variable values. When a format combines several internal values into one formatted value, PROC MEANS outputs the lowest internal value.

See [“Using Class Variables ” on page 1231](#)

Optional Arguments

ASCENDING

specifies to sort the class variable levels in ascending order.

Alias ASCEND

Interaction PROC MEANS issues a warning message if you specify both ASCENDING and DESCENDING and ignores both options.

See [“Example 10: Computing Output Statistics with Missing Class Variable Values” on page 1325](#)

DESCENDING

specifies to sort the class variable levels in descending order.

Alias DESCEND

Interaction PROC MEANS issues a warning message if you specify both ASCENDING and DESCENDING and ignores both options.

EXCLUSIVE

excludes from the analysis all combinations of the class variables that are not found in the preloaded range of user-defined formats.

Requirement You must specify PRELOADFMT to preload the class variable formats.

See [“Example 6: Using Preloaded Formats with Class Variables” on page 1314](#)

GROUPINTERNAL

specifies not to apply formats to the class variables when PROC MEANS groups the values to create combinations of class variables.

Interactions If you specify the PRELOADFMT option, then PROC MEANS ignores the GROUPINTERNAL option and uses the formatted values.

If you specify the ORDER=FORMATTED option, then PROC MEANS ignores the GROUPINTERNAL option and uses the formatted values.

Tip This option saves computer resources when the numeric class variables contain discrete values.

See [“Computer Resources” on page 1265](#)

MISSING

considers missing values as valid values for the class variable levels. Special missing values that represent numeric values (the letters A through Z and the underscore (_) character) are each considered as a separate value.

Default If you omit MISSING, then PROC MEANS excludes the observations with a missing class variable value from the analysis.

See *SAS Programmer's Guide: Essentials* for a discussion of missing values with special meanings.

[“Example 10: Computing Output Statistics with Missing Class Variable Values” on page 1325](#)

MLF

enables PROC MEANS to use the primary and secondary format labels for a given range or overlapping ranges to create subgroup combinations when a multilabel format is assigned to a class variable.

Requirement You must use PROC FORMAT and the MULTILABEL option in the VALUE statement to create a multilabel format.

Interactions If you use the OUTPUT statement with MLF, then the class variable contains a character string that corresponds to the formatted value. Because the formatted value becomes the internal value, the length of this variable is the number of characters in the longest format label.

Using MLF with ORDER=FREQ might not produce the order that you expect for the formatted values. You might not get the expected results when you use MLF with CLASSDATA and EXCLUSIVE because MLF processing requires that each TYPE be computed independently. Types other than NWAY might contain more levels than expected.

Note When the formatted values overlap, one internal class variable value maps to more than one class variable subgroup combination. Therefore, the sum of the N statistics for all subgroups is greater than the number of observations in the data set (the overall N statistic).

- Tip** If you omit MLF, then PROC MEANS uses the primary format labels. This action corresponds to using the first external format value to determine the subgroup combinations.
- See** The MULTILABEL option in the VALUE statement of the FORMAT procedure [“Optional Arguments” on page 889](#).
- [“Example 5: Using Multilabel Value Formats with Class Variables” on page 1309](#)

ORDER=DATA | FORMATTED | FREQ | UNFORMATTED

specifies the order to group the levels of the class variables in the output, where

DATA

orders values according to their order in the input data set.

- Interaction** If you use PRELOADFMT, then the order of the values of each class variable matches the order that PROC FORMAT uses to store the values of the associated user-defined format. If you use the CLASSDATA= option in the PROC statement, then PROC MEANS uses the order of the unique values of each class variable in the CLASSDATA= data set to order the output levels. If you use both options, then PROC MEANS first uses the user-defined formats to order the output. If you omit EXCLUSIVE in the PROC statement, then PROC MEANS appends after the user-defined format and the CLASSDATA= values the unique values of the class variables in the input data set based on the order in which they are encountered.
- Tip** By default, PROC FORMAT stores a format definition in sorted order. Use the NOTSORTED option to store the values or ranges of a user-defined format in the order in which you define them.
- See** [“Example 10: Computing Output Statistics with Missing Class Variable Values” on page 1325](#)

FORMATTED

orders values by their ascending formatted values. This order depends on your operating environment. If no format has been assigned to a class variable, then the default format, BEST12., is used.

- Aliases** FMT
- EXTERNAL

- See** [“Example 5: Using Multilabel Value Formats with Class Variables” on page 1309](#)

FREQ

orders values by descending frequency count so that levels with the most observations are listed first.

Interactions For multiway combinations of the class variables, PROC MEANS determines the order of a level from the individual class variable frequencies.

Use the ASCENDING option to order values by ascending frequency count.

See [“Example 5: Using Multilabel Value Formats with Class Variables” on page 1309](#)

UNFORMATTED

orders values by their unformatted values, which yields the same order as PROC SORT. This order depends on your operating environment. This sort sequence is particularly useful for displaying dates chronologically.

Aliases UNFMT

INTERNAL

Default UNFORMATTED

Tip By default, all orders except FREQ are ascending. For descending orders, use the DESCENDING option.

See [“Ordering the Class Values” on page 1232](#)

PRELOADFMT

specifies that all formats are preloaded for the class variables.

Requirement PRELOADFMT has no effect unless you specify either COMPLETETYPES, EXCLUSIVE, or ORDER=DATA and you assign formats to the class variables.

Interactions The combination of PRELOADFMT and COMPLETETYPES generates values that might not appear within the input data set. In some cases, for a character class variable, extremely low or high sentinel values are generated. These values might not represent valid characters of the session encoding. These values should not be printed in their raw form but should be printed using the associated format. Use of the OTHER= value in the format definition can prevent the raw sentinel values from printing when using the format.

To limit PROC MEANS output to the combinations of formatted class variable values present in the input data set, use the EXCLUSIVE option in the CLASS statement.

To include all ranges and values of the user-defined formats in the output, even when the frequency is zero, use COMPLETETYPES in the PROC statement.

Note When preloading a character format, there must be at least one representative raw value that produces the label. If a range does not have any raw values, the range is considered unreachable and an error is generated. For example, you cannot preload a multi-

label character format with both Other= and Low-High ranges. Other is a special keyword that refers to values or ranges that remain after all labels have been defined. For character formats, missing values are included in the low range so that there are no raw values left to occupy the Other range.

See [“Example 6: Using Preloaded Formats with Class Variables” on page 1314](#)

Details

Comparison of the BY and CLASS Statements

Using the BY statement is similar to using the CLASS statement and the NWAY option in that PROC MEANS summarizes each BY group as an independent subset of the input data. Therefore, no overall summarization of the input data is available. However, unlike the CLASS statement, the BY statement requires that the data is pre-sorted or indexed by the variables in the BY statement.

When you use the NWAY option, PROC MEANS might encounter insufficient memory for the summarization of all the class variables. You can move some class variables to the BY statement. For maximum benefit, move class variables to the BY statement that are already sorted or that have the greatest number of unique values.

You can use the CLASS and BY statements together to analyze the data by the levels of class variables within BY groups. See [“Example 3: Using the BY Statement with Class Variables” on page 1304](#).

How PROC MEANS Handles Missing Values for Class Variables

By default, if an observation contains a missing value for any class variable, then PROC MEANS excludes that observation from the analysis. If you specify the MISSING option in the PROC statement, then the procedure considers missing values as valid levels for the combination of class variables.

Specifying the MISSING option in the CLASS statement enables you to control the acceptance of missing values for individual class variables.

Computer Resources

The total of unique class variable values that PROC MEANS allows depends on the amount of computer memory that is available. See [“Computational Resources” on page 1233](#) for more information.

The GROUPINTERNAL option can improve computer performance because the grouping process is based on the internal values of the class variables. If a numeric class variable is not assigned a format and you do not specify GROUPINTERNAL, then PROC MEANS uses the default format, BEST12., to format numeric values as character strings. Then PROC MEANS groups these numeric variables by their character values, which takes additional time and computer memory.

FORMAT Statement

Associates formats with variables.

Syntax

FORMAT *variable-1* <...*variable-n*> *format variable-1* <...*variable-n*> *format*;

Arguments

variable

names one or more variables for SAS to associate with a format. You must specify at least one *variable*.

Tip To disassociate a format from a variable, use the variable in a FORMAT statement without specifying a format in a DATA step or in PROC DATASETS. In a DATA step, place this FORMAT statement after the SET statement. See [“Removing a Format” in SAS DATA Step Statements: Reference](#). You can also use PROC DATASETS.

format

specifies the format that is listed for writing the values of the variables.

Tip Formats that are associated with variables by using a FORMAT statement behave like formats that are used with a colon modifier in a subsequent PUT statement. For more information about using a colon modifier, see [“PUT Statement: List” in SAS DATA Step Statements: Reference](#).

See [SAS Formats and Informats: Reference](#)

FREQ Statement

Specifies a numeric variable that contains the frequency of each observation.

Alias: FREQUENCY

Syntax

FREQ *variable*;

Required Argument

variable

specifies a numeric variable whose value represents the frequency of the observation. If you use the FREQ statement, then the procedure assumes that each observation represents n observations, where n is the value of *variable*. If n is not an integer, then SAS truncates it. If n is less than 1 or is missing, then the procedure does not use that observation to calculate statistics.

The sum of the frequency variable represents the total number of observations.

Note: The FREQ variable does not affect how PROC MEANS identifies multiple extremes when you use the IDGROUP syntax in the OUTPUT statement.

ID Statement

Includes additional variables in the output data set.

See: [Discussion of id-group-specification in “OUTPUT Statement” on page 1270.](#)

Syntax

ID *variable(s)*;

Required Argument

variable(s)

identifies one or more variables from the input data set whose maximum values for groups of observations PROC MEANS includes in the output data set.

Interaction Use IDMIN in the PROC statement to include the minimum value of the ID variables in the output data set.

Tip Use the PRINTIDVARS option in the PROC statement to include the value of the ID variable in the displayed output.

Details

Selecting the Values of the ID Variables

When you specify only one variable in the ID statement, the value of the ID variable for a given observation is the maximum (minimum) value found in the corresponding group of observations in the input data set. When you specify multiple variables in the ID statement, PROC MEANS selects the maximum value

by processing the variables in the ID statement in the order in which you list them. PROC MEANS determines which observation to use from all the ID variables by comparing the values of the first ID variable. If more than one observation contains the same maximum (minimum) ID value, then PROC MEANS uses the second and subsequent ID variable values as “tiebreakers.” In any case, all ID values are taken from the same observation for any given BY group or classification level within a type.

See [“Sorting Orders for Character Variables” on page 2043](#) for information about how PROC MEANS compares character values to determine the maximum value.

LABEL Statement

Assigns descriptive labels to variables.

Syntax

LABEL *variable-1* = '*text-string*' <...*variable-n* = '*text-string*'>;

LABEL *variable-1* = '' <...*variable-n*=''>; | *variable-1* = <...*variable-n*=>;

Arguments

variable

specifies the variable that you want to label. You can specify labels for multiple variables in a single LABEL statement:

```
data test;
  L = 5;
  W = 10;
  label L = 'Length' W = 'Width';
run;
```

text-string

specifies a label of up to 256 bytes.

```
data test;
  L = 5;
  label L = 'Length';
run;
```

Restrictions If the label includes a semicolon (;) or an equal sign (=), you must enclose the label in either single or double quotation marks.

```
label N = 'Number = ';
```

If the label includes single quotation marks ('), you must enclose the label in double quotation marks.

```
label Name = "Owner's Last Name";
```

There is partial support for variable name substitution with the JAVAIMG device. When the label text for the variable exceeds the space allowed, the variable name is used instead by procedures that are labeling an axis or a legend title. With the exception of the SAS/GRAPH GPLOT procedure, the variable name is not used when the JAVAIMG device is specified.

Notes

Quotation marks are not required unless the label includes a semicolon, quotation marks, or an equal sign.

The LABEL statement expects an equal sign (=) after the first variable. If any other character is found, an error occurs. The LABEL statement considers everything after the equal sign, following the first variable, as part of the label until the statement comes to another variable followed by an equal sign, regardless of quotation marks. In this example, the label for x is **this is x y 4 = this is y** because the next variable that is followed by an equal sign is z.

```
data test;
  x = 1;
  y = 1;
  z = 1;
  label x = 'this is x' y 4 = 'this is y' z = 'This is z';
run;
```

In this LABEL statement, the x variable has the same label:

```
label x = this is x y 4 = this is y z = This is z;
```

See

[“Character Constants and Character Variables” in SAS Programmer’s Guide: Essentials.](#)

”

removes a label from a variable. Enclose a single blank space in quotation marks to remove an existing label. You can remove labels from multiple variables in a single LABEL statement:

```
data test;
  L=5; W=10;
  label L = ' ' W = ' ';
run;
```

Alternatively, you can remove a label by specifying nothing after the equal sign:

```
data test;
  L=5; W=10;
  label L = W = ;
run;
```

OUTPUT Statement

Writes statistics to a new SAS data set.

Tip: You can use multiple OUTPUT statements to create several OUT= data sets.

Examples: [“Example 8: Computing Output Statistics” on page 1320](#)
[“Example 9: Computing Different Output Statistics for Several Variables” on page 1322](#)
[“Example 10: Computing Output Statistics with Missing Class Variable Values” on page 1325](#)
[“Example 11: Identifying an Extreme Value with the Output Statistics” on page 1327](#)
[“Example 12: Identifying the Top Three Extreme Values with the Output Statistics” on page 1330](#)

Syntax

```
OUTPUT <OUT=SAS-data-set>
<statistic-specifications>
<id-group-specifications> <maximum-id-specifications>
<minimum-id-specifications> </ options>;
```

Optional Arguments

OUT=SAS-data-set

names the new output data set. If SAS-data-set does not exist, then PROC MEANS creates it. If you omit OUT=, then the data set is named DATA n , where n is the smallest integer that makes the name unique.

Default DATA n

Tip You can use data set options with the OUT= option.

statistic-keyword<(variable-list)><=name(s)>

specifies the statistics to store in the OUT= data set and names one or more variables that contain the statistics.

statistic-keyword

specifies which statistic to store in the output data set.

The available statistic keywords are listed in [“Statistic Keywords” on page 1277](#).

By default the statistics in the output data set automatically inherit the analysis variable's format, informat, and label. However, statistics computed for N, NMISS, SUMWGT, USS, CSS, VAR, CV, T, PROBT, PRT, SKEWNESS, and KURTOSIS do not inherit the analysis variable's format because this

format might be invalid for these statistics (for example, dollar or datetime formats).

Default N, MEAN, STD, MIN, and MAX

Restriction If you omit *variable* and *name(s)*, then PROC MEANS allows the *statistic-keyword* only once in a single OUTPUT statement, unless you also use the AUTONAME option.

See The definitions of the keywords and the formulas for the associated statistics are listed in [“Keywords and Formulas” on page 2426](#).

[“Example 8: Computing Output Statistics” on page 1320](#)

[“Example 9: Computing Different Output Statistics for Several Variables” on page 1322](#)

[“Example 11: Identifying an Extreme Value with the Output Statistics” on page 1327](#)

[“Example 12: Identifying the Top Three Extreme Values with the Output Statistics” on page 1330](#)

variable-list

specifies the names of one or more numeric analysis variables whose statistics you want to store in the output data set.

Default all numeric analysis variables

names

specifies one or more names for the variables in the output data set that contain the analysis variable statistics. The first name contains the statistic for the first analysis variable; the second name contains the statistic for the second analysis variable; and so on.

Default the analysis variable name. If you specify AUTONAME, then the default is the combination of the analysis variable name and the *statistic-keyword*. If you use the CLASS statement and an OUTPUT statement without an *output-statistic-specification*, then the output data set contains five observations for each combination of class variables: the value of N, MIN, MAX, MEAN, and STD. If you use the WEIGHT statement or the WEIGHT option in the VAR statement, then the output data set also contains an observation with the sum of weights (SUMWGT) for each combination of class variables.

Interaction If you specify *variable-list*, then PROC MEANS uses the order in which you specify the analysis variables to store the statistics in the output data set variables.

Tip Use the AUTONAME option to have PROC MEANS generate unique names for multiple variables and statistics.

See [“Example 8: Computing Output Statistics” on page 1320](#)

IDGROUP (<MIN (*variable-list-1*) | MAX (*variable-list-1*) <...MIN (*variable-list-n*) | MAX (*variable-list-n*) >> <MISSING> <OBS> <LAST> OUT <[*n*]> (*id-variable-list*)=<names>)

combines the features and extends the ID statement, the IDMIN option in the PROC statement, and the MAXID and MINID options in the OUTPUT statement to create an OUT= data set that identifies multiple extreme values.

LAST

specifies that the OUT= data set contains values from the last observation (or the last *n* observations, if *n* is specified).

If you do not specify LAST, then the OUT= data set contains values from the first observation (or the first *n* observations, if *n* is specified). The OUT= data set might contain several observations because in addition to the value of the last (first) observation, the OUT= data set contains values from the last (first) observation of each subgroup level that is defined by combinations of class variable values.

Interaction When you specify MIN or MAX and when multiple observations contain the same extreme values, PROC MEANS uses the observation number to resolve which observation to save to the OUT= data set. If you specify LAST, then PROC MEANS uses the later observations to resolve any ties. If you do not specify LAST, then PROC MEANS uses the earlier observations to resolve any ties.

MAX(*variable-list*)

specifies the selection criteria to determine the maximum extreme values of one or more input data set variables specified in *variable-list*.

When you specify multiple selection variables, the ordering of observations for the selection of *n* extremes is done the same way that PROC SORT sorts data with multiple BY variables. PROC MEANS concatenates the variable values into a single key. The MAX(*variable-list*) selection criterion is similar to using PROC SORT and the DESCENDING option in the BY statement.

Default If you do not specify MIN or MAX, then PROC MEANS uses the observation number as the selection criterion to output observations.

Restriction If you specify criteria that are contradictory, then PROC MEANS uses only the first selection criterion.

Interaction When multiple observations contain the same extreme values in all the MIN or MAX variables, PROC MEANS uses the observation number to resolve which observation to write to the output. By default, PROC MEANS uses the first observation to resolve any ties. However, if you specify the LAST option, then PROC MEANS uses the last observation to resolve any ties.

MIN(*variable-list*)

specifies the selection criteria to determine the minimum extreme values of one or more input data set variables specified in *variable-list*.

When you specify multiple selection variables, the ordering of observations for the selection of n extremes is done the same way that PROC SORT sorts data with multiple BY variables. PROC MEANS concatenates the variable values into a single key.

- | | |
|-------------|--|
| Default | If you do not specify MIN or MAX, then PROC MEANS uses the observation number as the selection criterion to output observations. |
| Restriction | If you specify criteria that are contradictory, then PROC MEANS uses only the first selection criterion. |
| Interaction | When multiple observations contain the same extreme values in all the MIN or MAX variables, PROC MEANS uses the observation number to resolve which observation to write to the output. By default, PROC MEANS uses the first observation to resolve any ties. However, if you specify the LAST option, then PROC MEANS uses the last observation to resolve any ties. |

MISSING

specifies that missing values be used in selection criteria.

Alias MISS

OBS

includes an `_OBS_` variable in the `OUT=` data set that contains the number of the observation in the input data set where the extreme value was found.

Interactions If you use WHERE processing, then the value of `_OBS_` might not correspond to the location of the observation in the input data set.

If you use `[n]` to write multiple extreme values to the output, then PROC MEANS creates n `_OBS_` variables and uses the suffix n to create the variable names, where n is a sequential integer from 1 to n .

`[n]`

specifies the number of extreme values for each variable in *id-variable-list* to include in the `OUT=` data set. PROC MEANS creates n new variables and uses the suffix `_n` to create the variable names, where n is a sequential integer from 1 to n .

By default, PROC MEANS determines one extreme value for each level of each requested type. If n is greater than one, then n extremes are generated for each level of each type. When n is greater than one and you request extreme value selection, the time complexity is $O(T * N \log_2 n)$, where T is the number of types requested and N is the number of observations in the input data set. By comparison, to group the entire data set, the time complexity is $O(N \log_2 N)$.

Default 1

Range an integer between 1 and 100

Example For example, to generate two minimum extreme values for each variable, use

```
idgroup(min(x) out[2] (x y z)=MinX MinY MinZ);
```

The OUT= data set contains the variables MinX_1, MinX_2, MinY_1, MinY_2, MinZ_1, and MinZ_2.

(id-variable-list)

identifies one or more input data set variables whose values PROC MEANS includes in the OUT= data set. PROC MEANS determines which observations to generate by the selection criteria that you specify (MIN, MAX, and LAST).

Alias IDGRP

Requirement You must specify the MIN | MAX selection criteria first and OUT(*id-variable-list*)= after the suboptions MISSING, OBS, and LAST.

Tips You can use *id-group-specification* to mimic the behavior of the ID statement and a *maximum-id-specification* or *minimum-id-specification* in the OUTPUT statement.

When you want the output data set to contain extreme values along with other ID variables, it is more efficient to include them in the *id-variable-list* than to request separate statistics. For example, the statement `output idgrp(max(x) out(x a b)= );` is more efficient than the statement `output idgrp(max(x) out(a b)= ) max(x)=;`

See [“Example 8: Computing Output Statistics” on page 1320](#)
[“Example 12: Identifying the Top Three Extreme Values with the Output Statistics” on page 1330](#)

names

specifies one or more names for variables in the OUT= data set.

Default If you omit *name*, then PROC MEANS uses the names of variables in the *id-variable-list*.

Tip Use the AUTONAME option to automatically resolve naming conflicts.

CAUTION

The IDGROUP syntax enables you to create output variables with the same name. When this action happens, only the first variable appears in the output data set. Use the AUTONAME option to automatically resolve these naming conflicts.

Note: If you specify fewer new variable names than the combination of analysis variables and identification variables, then the remaining output variables use

the corresponding names of the ID variables as soon as PROC MEANS exhausts the list of new variable names.

Alias IDGRP

MAXID *<(variable-1 <(id-variable-list-1)> <...variable-n <(id-variable-list-n)>>> = names*

specifies that one or more identification variables be associated with the maximum values of the analysis variables.

variable

identifies the numeric analysis variable whose maximum values PROC MEANS determines. PROC MEANS can determine several maximum values for a variable because, in addition to the overall maximum value, subgroup levels, which are defined by combinations of class variables values, also have maximum values.

Tip If you use an ID statement and omit *variable*, then PROC MEANS uses all analysis variables.

id-variable-list

identifies one or more variables whose values identify the observations with the maximum values of the analysis variable.

Default the ID statement variables

names

specifies the names for new variables that contain the values of the identification variable associated with the maximum value of each analysis variable.

Note If multiple observations contain the maximum value within a class level, then PROC MEANS saves the value of the ID variable for only the first of those observations in the output data set.

Tips If you use an ID statement, and omit *variable* and *id-variable*, then PROC MEANS associates all ID statement variables with each analysis variable. Thus, for each analysis variable, the number of variables that are created in the output data set equals the number of variables that you specify in the ID statement.

Use the AUTONAME option to automatically resolve naming conflicts.

See [“Example 11: Identifying an Extreme Value with the Output Statistics” on page 1327](#)

CAUTION

The MAXID syntax enables you to create output variables with the same name. When this action happens, only the first variable appears in the output data set. Use the AUTONAME option to automatically resolve these naming conflicts.

Note: If you specify fewer new variable names than the combination of analysis variables and identification variables, then the remaining output variables use the corresponding names of the ID variables as soon as PROC MEANS exhausts the list of new variable names.

MINID<(variable-1 <(id-variable-list-1)> <...variable-n <(id-variable-list-n)>>> = name(s)

see the description of maximum-id-specification. This option behaves in exactly the same way, except that PROC MEANS determines the minimum values instead of the maximum values.

When MINID is used without an explicit variable list, it is similar to the following more advanced IDGROUP syntax example:

```
IDGRP( min(x) missing out(id_variable)=idminx) idgrp( min(y) missing
out(id_variable)=idminy)
```

If one or more of the analysis variables has a missing value, the id_variable value corresponds to the observation with the missing value not the observation with the value for the MIN statistic.

Options to Use after a Slash Delimiter

AUTOLABEL

specifies that PROC MEANS appends the statistic name to the end of the variable label. If an analysis variable has no label, then PROC MEANS creates a label by appending the statistic name to the analysis variable name.

See [“Example 12: Identifying the Top Three Extreme Values with the Output Statistics” on page 1330](#)

AUTONAME

specifies that PROC MEANS creates a unique variable name for an output statistic when you do not assign the variable name in the OUTPUT statement. This action is accomplished by appending to the *statistic-keyword* to the end of the input variable name from which the statistic was derived.

For example, the statement `output min(x)=/autoname;` produces the x_Min variable in the output data set.

AUTONAME activates the SAS internal mechanism to automatically resolve conflicts in the variable names in the output data set. Duplicate variables do not generate errors.

As a result, the statement `output min(x)= min(x)=/autoname;` produces two variables, x_Min and x_Min2, in the output data set.

If the new variable name exceeds 32 characters, then the variable-name portion is truncated.

See [“Example 12: Identifying the Top Three Extreme Values with the Output Statistics” on page 1330](#)

KEEPLEN

specifies that statistics in the output data set inherit the length of the analysis variable that PROC MEANS uses to derive them.

CAUTION

You permanently lose numeric precision when the length of the analysis variable causes PROC MEANS to truncate or round the value of the statistic. However, the precision of the statistic matches that of the input.

LEVELS

includes a variable named `_LEVEL_` in the output data set. This variable contains a value from 1 to n that indicates a unique combination of the values of class variables (the values of `_TYPE_` variable).

See [“Output Data Set” on page 1297](#)

[“Example 8: Computing Output Statistics” on page 1320](#)

NOINHERIT

specifies that the variables in the output data set that contain statistics do not inherit the attributes (label and format) of the analysis variables that are used to derive them.

Interaction When no option is used (implied INHERIT) then the statistics inherit the attributes, label and format, of the input analysis variable(s). If the INHERIT option is used in the OUTPUT statement, then the statistics inherit the length of the input analysis variable(s), the label and format.

Tip By default, the output data set includes an output variable for each analysis variable and for five observations that contain N, MIN, MAX, MEAN, and STDDEV. Unless you specify NOINHERIT, this variable inherits the format of the analysis variable, which can be invalid for the N statistic (for example, datetime formats).

WAYS

includes a variable named `_WAY_` in the output data set. This variable contains a value from 1 to the maximum number of class variables that indicates how many class variables PROC MEANS combines to create the TYPE value.

See [“Output Data Set” on page 1297](#)

[“WAYS Statement” on page 1283](#)

[“Example 8: Computing Output Statistics” on page 1320](#)

Statistic Keywords

CLM

is a descriptive statistic keyword. For details, see [“*statistic-keyword*<\(variable-list\)><=name\(s\)>” on page 1270](#).

CSS

is a descriptive statistic keyword. For details, see “*statistic-keyword<(variable-list)><=name(s)>*” on page 1270.

KURTOSIS

is a descriptive statistic keyword. For details, see “*statistic-keyword<(variable-list)><=name(s)>*” on page 1270.

Alias KURT

LCLM

is a descriptive statistic keyword. For details, see “*statistic-keyword<(variable-list)><=name(s)>*” on page 1270.

MAX

is a descriptive statistic keyword. For details, see “*statistic-keyword<(variable-list)><=name(s)>*” on page 1270.

MEAN

is a descriptive statistic keyword. For details, see “*statistic-keyword<(variable-list)><=name(s)>*” on page 1270.

MIN

is a descriptive statistic keyword. For details, see “*statistic-keyword<(variable-list)><=name(s)>*” on page 1270.

MODE

is a descriptive statistic keyword. For details, see “*statistic-keyword<(variable-list)><=name(s)>*” on page 1270.

N

is a descriptive statistic keyword. For details, see “*statistic-keyword<(variable-list)><=name(s)>*” on page 1270.

NMISS

is a descriptive statistic keyword. For details, see “*statistic-keyword<(variable-list)><=name(s)>*” on page 1270.

RANGE

is a descriptive statistic keyword. For details, see “*statistic-keyword<(variable-list)><=name(s)>*” on page 1270.

SKEWNESS

is a descriptive statistic keyword. For details, see “*statistic-keyword<(variable-list)><=name(s)>*” on page 1270.

Alias SKEW

STDDEV

is a descriptive statistic keyword. For details, see “*statistic-keyword<(variable-list)><=name(s)>*” on page 1270.

Alias STD

STDERR

is a descriptive statistic keyword. For details, see “*statistic-keyword<(variable-list)><=name(s)>*” on page 1270.

SUM

is a descriptive statistic keyword. For details, see “[statistic-keyword<\(variable-list\)><=name\(s\)>](#)” on page 1270.

SUMWGT

is a descriptive statistic keyword. For details, see “[statistic-keyword<\(variable-list\)><=name\(s\)>](#)” on page 1270.

UCLM

is a descriptive statistic keyword. For details, see “[statistic-keyword<\(variable-list\)><=name\(s\)>](#)” on page 1270.

USS

is a descriptive statistic keyword. For details, see “[statistic-keyword<\(variable-list\)><=name\(s\)>](#)” on page 1270.

VAR

is a descriptive statistic keyword. For details, see “[statistic-keyword<\(variable-list\)><=name\(s\)>](#)” on page 1270.

MEDIAN

is a quantile statistic keyword. For details, see “[statistic-keyword<\(variable-list\)><=name\(s\)>](#)” on page 1270.

Alias P50

P1

is a quantile statistic keyword. For details, see “[statistic-keyword<\(variable-list\)><=name\(s\)>](#)” on page 1270.

P5

is a quantile statistic keyword. For details, see “[statistic-keyword<\(variable-list\)><=name\(s\)>](#)” on page 1270.

P10

is a quantile statistic keyword. For details, see “[statistic-keyword<\(variable-list\)><=name\(s\)>](#)” on page 1270.

P20

is a quantile statistic keyword. For details, see “[statistic-keyword<\(variable-list\)><=name\(s\)>](#)” on page 1270.

P40

is a quantile statistic keyword. For details, see “[statistic-keyword<\(variable-list\)><=name\(s\)>](#)” on page 1270.

P70

is a quantile statistic keyword. For details, see “[statistic-keyword<\(variable-list\)><=name\(s\)>](#)” on page 1270.

Q1

is a quantile statistic keyword. For details, see “[statistic-keyword<\(variable-list\)><=name\(s\)>](#)” on page 1270.

Alias P25

Q3

is a quantile statistic keyword. For details, see [“*statistic-keyword*<\(variable-list\)><=name\(s\)>” on page 1270](#).

Alias P75

P90

is a quantile statistic keyword. For details, see [“*statistic-keyword*<\(variable-list\)><=name\(s\)>” on page 1270](#).

P95

is a quantile statistic keyword. For details, see [“*statistic-keyword*<\(variable-list\)><=name\(s\)>” on page 1270](#).

P99

is a quantile statistic keyword. For details, see [“*statistic-keyword*<\(variable-list\)><=name\(s\)>” on page 1270](#).

P30

is a quantile statistic keyword. For details, see [“*statistic-keyword*<\(variable-list\)><=name\(s\)>” on page 1270](#).

P60

is a quantile statistic keyword. For details, see [“*statistic-keyword*<\(variable-list\)><=name\(s\)>” on page 1270](#).

P80

is a quantile statistic keyword. For details, see [“*statistic-keyword*<\(variable-list\)><=name\(s\)>” on page 1270](#).

QRANGE

is a quantile statistic keyword. For details, see [“*statistic-keyword*<\(variable-list\)><=name\(s\)>” on page 1270](#).

PROBT

is a hypothesis testing keyword. For details, see [“*statistic-keyword*<\(variable-list\)><=name\(s\)>” on page 1270](#).

Alias PBT

T

is a hypothesis testing keyword. For details, see [“*statistic-keyword*<\(variable-list\)><=name\(s\)>” on page 1270](#).

TYPES Statement

Identifies which of the possible combinations of class variables to generate.

Requirement: CLASS statement

See: [“Output Data Set” on page 1297](#)

Examples: [“Example 2: Computing Descriptive Statistics with Class Variables” on page 1301](#)
[“Example 5: Using Multilabel Value Formats with Class Variables” on page 1309](#)

“Example 12: Identifying the Top Three Extreme Values with the Output Statistics”
on page 1330

Syntax

TYPES *request(s)*;

Required Argument

request(s)

specifies which of the 2^k combinations of class variables PROC MEANS uses to create the types, where k is the number of class variables. A request includes one class variable name, several class variable names separated by asterisks, or ().

To request class variable combinations quickly, use a grouping syntax by placing parentheses around several variables and joining other variables or variable combinations. For example, the following statements illustrate grouping syntax:

Request	Equivalent to
types A*(B C) ;	types A*B A*C;
types (A B)*(C D) ;	types A*C A*D B*C B*D;
types (A B C)*D;	types A*D B*D C*D;

- Interaction

The CLASSDATA= option places constraints on the NWAY type. PROC MEANS generates all other types as if derived from the resulting NWAY type.
- Tips

Use () to request the overall total (_TYPE_=0).

If you do not need all types in the output data set, then use the TYPES statement to specify particular subtypes rather than applying a WHERE clause to the data set. Doing so saves time and computer memory.

Details

Order of Analyses in the Output

The analyses are written to the output in order of increasing values of the _TYPE_ variable, which is calculated by PROC MEANS. The _TYPE_ variable has a unique value for each combination of class variables; the values are determined by how

you specify the CLASS statement, not the TYPES statement. Therefore, if you specify

```
class A B C;
types (A B)*C;
```

then the B*C analysis (_TYPE_=3) is written first, followed by the A*C analysis (_TYPE_=5). However, if you specify

```
class B A C;
types (A B)*C;
```

then the A*C analysis comes first.

The _TYPE_ variable is calculated even if no output data set is requested. For more information about the _TYPE_ variable, see [“Output Data Set” on page 1297](#).

VAR Statement

Identifies the analysis variables and their order in the output.

Aliases:	VARS VARIABLES
Default:	If you omit the VAR statement, then PROC MEANS analyzes all numeric variables that are not listed in the other statements. When all variables are character variables, PROC MEANS produces a simple count of observations.
Tip:	You can use multiple VAR statements.
See:	Chapter 60, “SUMMARY Procedure,” on page 2153
Example:	“Example 1: Computing Specific Descriptive Statistics” on page 1299

Syntax

```
VAR variable(s) </ WEIGHT=weight-variable>;
```

Required Argument

variable(s)

identifies the analysis variables and specifies their order in the results.

Optional Argument

WEIGHT=*weight-variable*

specifies a numeric variable whose values weight the values of the variables that are specified in the VAR statement. The variable does not have to be an integer. The following table describes how PROC MEANS treats various values of the WEIGHT variable.

Weight Value	PROC MEANS Response
0	Counts the observation in the total number of observations
Less than 0	Converts the value to zero and counts the observation in the total number of observations
Missing	Excludes the observation

To exclude observations that contain negative and zero weights from the analysis, use EXCLNPWGT. Note that most SAS/STAT procedures, such as PROC GLM, exclude negative and zero weights by default.

The weight variable does not change how the procedure determines the range, extreme values, or number of missing values.

Restrictions To compute weighted quantiles, use QMETHOD=OS in the PROC statement.

Skewness and kurtosis are not available with the WEIGHT option.

Note Prior to Version 7 of SAS, the procedure did not exclude the observations with missing weights from the count of observations.

Tips When you use the WEIGHT option, consider which value of the VARDEF= option is appropriate. See the discussion of VARDEF=.

Use the WEIGHT option in multiple VAR statements to specify different weights for the analysis variables.

WAYS Statement

Specifies the number of ways to make unique combinations of class variables.

Tip: Use the TYPES statement to specify additional combinations of class variables.

Example: [“Example 6: Using Preloaded Formats with Class Variables” on page 1314](#)

Syntax

WAYS *list*;

Required Argument

list

specifies one or more integers that define the number of class variables to combine to form all the unique combinations of class variables. For example, you can specify 2 for all possible pairs and 3 for all possible triples. The *list* can be specified in the following ways:

- *m*
- *m1 m2 ... mn*
- *m1,m2,...,mn*
- *m TO n <BY increment>*
- *m1,m2 TO m3 <BY increment>, m4*

Range 0 to maximum number of class variables

See WAYS option

Example The following code is an example of creating two-way types for the classification variables A, B, and C. This WAYS statement is equivalent to specifying a*b, a*c, and b*c in the TYPES statement.

```
class A B C ; ways 2;
```

WEIGHT Statement

Specifies weights for analysis variables in the statistical calculations. You can use a WEIGHT statement and a FREQ statement in the same step of any procedure that supports both statements.

Alias: WGT

Restrictions: To compute weighted quantiles, use QMETHOD=OS in the PROC statement. Skewness and kurtosis are not available with the WEIGHT statement. PROC MEANS does not compute MODE when a weight variable is active. Instead, try using the UNIVARIATE procedure when MODE needs to be computed and a weight variable is active.

Interaction: If you use the WEIGHT= option in a VAR statement to specify a weight variable, then PROC MEANS uses this variable instead to weight those VAR statement variables.

Syntax

WEIGHT *variable*;

Required Argument

variable

specifies a numeric variable whose values weight the values of the analysis variables. The values of the variable do not have to be integers.

Different behavior for nonpositive values is discussed in the WEIGHT statement syntax under the individual procedure.

You can exclude the observations with missing weights from the analysis. Most SAS/STAT procedures, such as PROC GLM, have always excluded not only missing weights but also negative and zero weights from the analysis. You can achieve this same behavior in a Base SAS procedure that supports the WEIGHT statement by using the EXCLNPWGT option in the PROC statement.

Weight value	Procedure
0	Counts the observation in the total number of observations
Less than 0	Converts the weight value to zero and counts the observation in the total number of observations
Missing	Excludes the observation from the analysis

To exclude observations that contain negative and zero weights from the analysis, use EXCLNPWGT. Note that most SAS/STAT procedures, such as PROC GLM, exclude negative and zero weights by default.

CAUTION

Single extreme weight values can cause inaccurate results. When one (and only one) weight value is many orders of magnitude larger than the other weight values (for example, 49 weight values of 1 and one weight value of 1×10^{14}), certain statistics might not be within acceptable accuracy limits. The affected statistics are based on the second moment (such as standard deviation, corrected sum of squares, variance, and standard error of the mean). Under certain circumstances, no warning is written to the SAS log.

The procedure substitutes the value of the WEIGHT variable for w_i , which appears in [“Keywords and Formulas” on page 2426](#).

Example

Procedures That Support the WEIGHT Statement

- CORR
- FREQ
- MEANS or SUMMARY
- REPORT
- STANDARD
- TABULATE
- UNIVARIATE

Note: In PROC FREQ, the value of the variable in the WEIGHT statement represents the frequency of occurrence for each observation. For more information, see the information about PROC FREQ in the *Base SAS(R) 9.3 Procedures Guide: Statistical Procedures*.

Calculating Weighted Statistics

The procedures that support the WEIGHT statement also support the VARDEF= option, which lets you specify a divisor to use in the calculation of the variance and standard deviation.

By using a WEIGHT statement to compute moments, you assume that the i th observation has a variance that is equal to σ^2/w_i . When you specify VARDEF=DF (the default), the computed variance is a weighted least squares estimate of σ^2 . Similarly, the computed standard deviation is an estimate of σ . Note that the computed variance is not an estimate of the variance of the i th observation, because this variance involves the observation's weight, which varies from observation to observation.

If the values of your variable are counts that represent the number of occurrences of each observation, then use this variable in the FREQ statement rather than in the WEIGHT statement. In this case, because the values are counts, they should be integers. (The FREQ statement truncates any noninteger values.) The variance that is computed with a FREQ variable is an estimate of the common variance σ^2 of the observations.

Note: If your data comes from a stratified sample where the weights w_i represent the strata weights, then neither the WEIGHT statement nor the FREQ statement

provides appropriate stratified estimates of the mean, variance, or variance of the mean. To perform the appropriate analysis, consider using PROC SURVEYMEANS, which is a SAS/STAT procedure that is documented in the *Base SAS(R) 9.3 Procedures Guide: Statistical Procedures*.

Use Weighted Statistics

As an example of the WEIGHT statement, suppose 20 people are asked to estimate the size of an object 30 cm wide. Each person is placed at a different distance from the object. As the distance from the object increases, the estimates should become less precise. The SAS data set SIZE contains the estimate (ObjectSize) in centimeters at each distance (Distance) in meters and the precision (Precision) for each estimate. Notice that the largest deviation (an overestimate by 20 cm) came at the greatest distance (7.5 meters from the object). As a measure of precision, $1/\text{Distance}$, gives more weight to estimates that were made closer to the object and less weight to estimates that were made at greater distances. The PROC MEANS step computes the average estimate of the object size while ignoring the weights. Without a WEIGHT variable, PROC MEANS uses the default weight of 1 for every observation. Thus, the estimates of object size at all distances are given equal weight. The average estimate of the object size exceeds the actual size by 3.55 cm.

```
options nodate pageno=1 linesize=64 pagesize=60;

data size;
    input Distance ObjectSize @@;
    Precision=1/distance;
    datalines;
1.5 30 1.5 20 1.5 30 1.5 25
3   43 3   33 3   25 3   30
4.5 25 4.5 36 4.5 48 4.5 33
6   43 6   36 6   23 6   48
7.5 30 7.5 25 7.5 50 7.5 38
;

proc means data=size maxdec=3 n mean var stddev;
    var objectsize;
    title1 'Unweighted Analysis of the SIZE Data Set';
run;
```

Output

Unweighted Analysis of the SIZE Data Set

The MEANS Procedure

Analysis Variable : ObjectSize			
N	Mean	Variance	Std Dev
20	33.550	80.892	8.994

Use the WEIGHT Statement with Precision in PROC MEANS

The next two PROC MEANS steps use the precision measure (Precision) in the WEIGHT statement and show the effect of using different values of the VARDEF= option. The first PROC step creates an output data set that contains the variance and standard deviation. If you reduce the weighting of the estimates that are made at greater distances, the weighted average estimate of the object size is closer to the actual size. The variance of the i th observation is assumed to be $\text{var}(x_i) = \sigma^2/w_i$ and w_i is the weight for the i th observation. In the first PROC MEANS step, the computed variance is an estimate of σ^2 . In the second PROC MEANS step, the computed variance is an estimate of $(n - 1/n)\sigma^2/\bar{w}$, where $\bar{w}$ is the average weight. For large n , this value is an approximate estimate of the variance of an observation with average weight.

```
proc means data=size maxdec=3 n mean var stddev;
  weight precision;
  var objectsize;
  output out=wtstats var=Est_SigmaSq std=Est_Sigma;
  title1 'Weighted Analysis Using Default VARDEF=DF';
run;

proc means data=size maxdec=3 n mean var std
           vardef=weight;
  weight precision;
  var objectsize;
  title1 'Weighted Analysis Using VARDEF=WEIGHT';
run;
```

Output

Weighted Analysis Using Default VARDEF=DF

The MEANS Procedure

Analysis Variable : ObjectSize			
N	Mean	Variance	Std Dev
20	31.088	20.678	4.547

Weighted Analysis Using VARDEF=WEIGHT

The MEANS Procedure

Analysis Variable : ObjectSize			
N	Mean	Variance	Std Dev
20	31.088	64.525	8.033

Create a Data Set with the Weighted Variance and Weighted Standard Deviation of Each Observation

In the following statements, DATA step combines the output data set that contains the variance and the standard deviation from the weighted analysis with the original data set. The variance of each observation is computed by dividing Est_SigmaSq (the estimate of σ^2 from the weighted analysis when VARDEF=DF) by each observation's weight (Precision). The standard deviation of each observation is computed by dividing Est_Sigma (the estimate of σ from the weighted analysis when VARDEF=DF) by the square root of each observation's weight (Precision).

```
data wtsize(drop=_freq_ _type_);
  set size;
  if _n_=1 then set wtstats;
  Est_VarObs=est_sigmasq/precision;
  Est_StdObs=est_sigma/sqrt(precision);

proc print data=wtsize noobs;
  title 'Weighted Statistics';
  by distance;
  format est_varobs est_stdobs
         est_sigmasq est_sigma precision 6.3;
run;
```

Output

Weighted Statistics

Distance=1.5

ObjectSize	Precision	Est_SigmaSq	Est_Sigma	Est_VarObs	Est_StdObs
30	0.667	20.678	4.547	31.017	5.569
20	0.667	20.678	4.547	31.017	5.569
30	0.667	20.678	4.547	31.017	5.569
25	0.667	20.678	4.547	31.017	5.569

Distance=3

ObjectSize	Precision	Est_SigmaSq	Est_Sigma	Est_VarObs	Est_StdObs
43	0.333	20.678	4.547	62.035	7.876
33	0.333	20.678	4.547	62.035	7.876
25	0.333	20.678	4.547	62.035	7.876
30	0.333	20.678	4.547	62.035	7.876

Distance=4.5

ObjectSize	Precision	Est_SigmaSq	Est_Sigma	Est_VarObs	Est_StdObs
25	0.222	20.678	4.547	93.052	9.646
36	0.222	20.678	4.547	93.052	9.646
48	0.222	20.678	4.547	93.052	9.646
33	0.222	20.678	4.547	93.052	9.646

Distance=6					
ObjectSize	Precision	Est_SigmaSq	Est_Sigma	Est_VarObs	Est_StdObs
43	0.167	20.678	4.547	124.07	11.139
36	0.167	20.678	4.547	124.07	11.139
23	0.167	20.678	4.547	124.07	11.139
48	0.167	20.678	4.547	124.07	11.139

Distance=7.5					
ObjectSize	Precision	Est_SigmaSq	Est_Sigma	Est_VarObs	Est_StdObs
30	0.133	20.678	4.547	155.09	12.453
25	0.133	20.678	4.547	155.09	12.453
50	0.133	20.678	4.547	155.09	12.453
38	0.133	20.678	4.547	155.09	12.453

WHERE Statement

Subsets the input data set by specifying certain conditions that each observation must meet before it is available for processing.

Syntax

WHERE *where-expression-1*
 < *logical-operator where-expression-n*>;

Arguments

where-expression

is an arithmetic or logical expression that generally consists of a sequence of operands and operators.

Tips The operands and operators described in the next several sections are also valid for the WHERE= data set option.

You can specify multiple where-expressions.

logical-operator

can be AND, AND NOT, OR, or OR NOT.

Example

Procedures That Support the WHERE Statement

You can use the WHERE statement with any of the following Base SAS procedures that read a SAS data set:

COMPARE	REPORT
CORR	SORT
DATASETS (APPEND statement)	SQL
FREQ	STANDARD
MEANS or SUMMARY	TABULATE
PRINT	TRANSPOSE
RANK	UNIVARIATE

Details

- The COMPARE procedure and the APPEND statement in PROC DATASETS accept more than one input data set. For more information, see the documentation for the specific procedure.
- To subset the output data set, use the WHERE= data set option:

```
proc report data=debate nowd  
            out=onlyfr (where= (year='1')) ;  
run;
```

For more information about WHERE=, see [SAS DATA Step Statements: Reference](#).

Temporarily Dissociating a Format from a Variable

In this example, PROC PRINT prints only those observations that meet the condition of the WHERE expression.

```
options nodate pageno=1 linesize=64
      pagesize=40;

proc print data=debate noobs;
  where gpa>3.5;
  title 'Team Members with a GPA';
  title2 'Greater than 3.5';
run;
```

Output

Team Members with a GPA Greater than 3.5			
Name	Gender	Year	GPA
Capiccio	m	Freshman	3.598
Tucker	m	Freshman	3.901
Bagwell	f	Sophomore	3.722
Gold	f	Junior	3.609
Syme	f	Junior	3.883
Baglione	f	Senior	4.000
Carr	m	Senior	3.750
Hall	m	Senior	3.574

Usage: MEANS Procedure

Computation of Moment Statistics

PROC MEANS uses single-pass algorithms to compute the moment statistics (such as mean, variance, skewness, and kurtosis). See [“Keywords and Formulas” on page 2426](#) for the statistical formulas.

The computational details for confidence limits, hypothesis test statistics, and quantile statistics follow.

Confidence Limits

With the keywords CLM, LCLM, and UCLM, you can compute confidence limits for the mean. A confidence limit is a range, constructed around the value of a sample statistic, that contains the corresponding true population value with given probability (ALPHA=) in repeated sampling.

A two-sided $100(1 - \alpha)$ % confidence interval for the mean has upper and lower limits

$$\bar{x} \pm t_{(1 - \alpha/2; n - 1)} \frac{s}{\sqrt{n}}$$

where s is $\sqrt{\frac{1}{n-1} \sum (x_i - \bar{x})^2}$ and $t_{(1 - \alpha/2; n - 1)}$ is the $(1 - \alpha/2)$ critical value of the Student's t statistics with $n - 1$ degrees of freedom.

A one-sided $100(1 - \alpha)$ % confidence interval is computed as

$$\bar{x} + t_{(1 - \alpha; n - 1)} \frac{s}{\sqrt{n}} \quad (\text{upper})$$

$$\bar{x} - t_{(1 - \alpha; n - 1)} \frac{s}{\sqrt{n}} \quad (\text{lower})$$

If you use the WEIGHT statement or WEIGHT= in a VAR statement and the default value of VARDEF=, which is DF, the $100(1 - \alpha)$ % confidence interval for the weighted mean has upper and lower limits

$$\bar{y}_w \pm t_{(1 - \alpha/2)} \frac{s_w}{\sqrt{\sum_{i=1}^n w_i}}$$

where $\bar{y}_w$ is the weighted mean, s_w is the weighted standard deviation, w_i is the weight for i th observation, and $t_{(1 - \alpha/2)}$ is the $(1 - \alpha/2)$ critical value for the Student's t distribution with $n - 1$ degrees of freedom.

Student's t Test

PROC MEANS calculates the t statistic as

$$t = \frac{\bar{x} - \mu_0}{s/\sqrt{n}}$$

where $\bar{x}$ is the sample mean, n is the number of nonmissing values for a variable, and s is the sample standard deviation. Under the null hypothesis, the population mean equals μ_0 . When the data values are approximately normally distributed, the probability under the null hypothesis of a t statistic as extreme as, or more extreme than, the observed value (the p -value) is obtained from the t distribution with $n - 1$

degrees of freedom. For large n , the t statistic is asymptotically equivalent to a z test.

When you use the WEIGHT statement or WEIGHT= in a VAR statement and the default value of VARDEF=, which is DF, the Student's t statistic is calculated as

$$t_w = \frac{\bar{y}_w - \mu_0}{s_w / \sqrt{\frac{n}{\sum_{i=1}^n w_i}}}$$

where $\bar{y}_w$ is the weighted mean, s_w is the weighted standard deviation, and w_i is the weight for i th observation. The t_w statistic is treated as having a Student's t distribution with $n - 1$ degrees of freedom. If you specify the EXCLNPWGT option in the PROC statement, then n is the number of nonmissing observations when the value of the WEIGHT variable is positive. By default, n is the number of nonmissing observations for the WEIGHT variable.

Quantiles

The options QMETHOD=, QNTLDEF=, and QMARKERS= determine how PROC MEANS calculates quantiles. QNTLDEF= deals with the mathematical definition of a quantile. See [“Quantile and Related Statistics” on page 2432](#). QMETHOD= deals with the mechanics of how PROC MEANS handles the input data. The two methods are

OS

reads all data into memory and sorts it by unique value.

P2

accumulates all data into a fixed sample size that is used to approximate the quantile.

If data set A has 100 unique values for a numeric variable X and data set B has 1000 unique values for numeric variable X, then QMETHOD=OS for data set B takes 10 times as much memory as it does for data set A. If QMETHOD=P2, then both data sets A and B requires the same memory space to generate quantiles.

The QMETHOD=P2 technique is based on the piecewise-parabolic (P^2) algorithm invented by Jain and Chlamtac (1985). P^2 is a one-pass algorithm to determine quantiles for a large data set. It requires a fixed amount of memory for each variable for each level within the type. However, using simulation studies, reliable estimations of some quantiles (P1, P5, P95, P99) cannot be possible for some data sets such as data sets with heavily tailed or skewed distributions.

If the number of observations is less than the QMARKERS= value, then QMETHOD=P2 produces the same results as QMETHOD=OS when QNTLDEF=5. To compute weighted quantiles, you must use QMETHOD=OS.

Results: MEANS Procedure

Missing Values

PROC MEANS excludes missing values for the analysis variables before calculating statistics. Each analysis variable is treated individually; a missing value for an observation in one variable does not affect the calculations for other variables. The statements handle missing values as follows:

- If a class variable has a missing value for an observation, then PROC MEANS excludes that observation from the analysis unless you use the MISSING option in the PROC statement or CLASS statement.
- If a BY or ID variable value is missing, then PROC MEANS treats it like any other BY or ID variable value. The missing values form a separate BY group.
- If a FREQ variable value is missing or nonpositive, then PROC MEANS excludes the observation from the analysis.
- If a WEIGHT variable value is missing, then PROC MEANS excludes the observation from the analysis.

PROC MEANS tabulates the number of the missing values. Before the number of missing values are tabulated, PROC MEANS excludes observations with frequencies that are nonpositive when you use the FREQ statement and observations with weights that are missing or nonpositive (when you use the EXCLNPWGT option) when you use the WEIGHT statement. To report this information in the procedure output use the NMISS statistical keyword in the PROC statement.

Column Width for the Output

You control the column width for the displayed statistics with the FW= option in the PROC statement. Unless you assign a format to a numeric class or an ID variable, PROC MEANS uses the value of the FW= option. When you assign a format to a numeric class or an ID variable, PROC MEANS determines the column width directly from the format. If you use the PRELOADFMT option in the CLASS statement, then PROC MEANS determines the column width for a class variable from the assigned format.

The N Obs Statistic

By default when you use a CLASS statement, PROC MEANS displays an additional statistic called N Obs. This statistic reports the total number of observations or the sum of the observations of the FREQ variable that PROC MEANS processes for each class level. PROC MEANS might omit observations from this total because of missing values in one or more class variables or because of the effect of the EXCLUSIVE option when you use it with the PRELOADFMT option or the CLASSDATA= option. Because of this action and the exclusion of observations when the WEIGHT variable contains missing values, there is not always a direct relationship between N Obs, N, and NMISS.

In the output data set, the value of N Obs is stored in the _FREQ_ variable. Use the NONOBS option in the PROC statement to suppress this information in the displayed output.

Output Data Set

PROC MEANS can create one or more output data sets. The procedure does not print the output data set. Use PROC PRINT, PROC REPORT, or another SAS reporting tool to display the output data set.

Note: By default the statistics in the output data set automatically inherit the analysis variable's format and label. However, statistics computed for N, NMISS, SUMWGT, USS, CSS, VAR, CV, T, PROBT, PRT, SKEWNESS, and KURTOSIS do not inherit the analysis variable's format because this format can be invalid for these statistics. Use the NOINHERIT option in the OUTPUT statement to prevent the other statistics from inheriting the format and label attributes.

The output data set can contain these variables:

- the variables specified in the BY statement.
- the variables specified in the ID statement.
- the variables specified in the CLASS statement.
- the variable _TYPE_ that contains information about the class variables. By default _TYPE_ is a numeric variable. If you specify CHARTYPE in the PROC statement, then _TYPE_ is a character variable. When you use more than 32 class variables, _TYPE_ is automatically a character variable.
- the variable _FREQ_ that contains the number of observations that a given output level represents.
- the variables requested in the OUTPUT statement that contain the output statistics and extreme values.

- the variable `_STAT_` that contains the names of the default statistics if you omit statistic keywords.
- the variable `_LEVEL_` if you specify the `LEVEL` option.
- the variable `_WAY_` if you specify the `WAYS` option.

The value of `_TYPE_` indicates which combination of the class variables PROC MEANS uses to compute the statistics. The character value of `_TYPE_` is a series of zeros and ones, where each value of one indicates an active class variable in the type. For example, with three class variables, PROC MEANS represents type 1 as 001, type 5 as 101, and so on.

Usually, the output data set contains one observation per level per type. However, if you omit statistical keywords in the `OUTPUT` statement, then the output data set contains five observations per level (six if you specify a `WEIGHT` variable). Therefore, the total number of observations in the output data set is equal to the sum of the levels for all the types that you request multiplied by 1, 5, or 6, whichever is applicable.

If you omit the `CLASS` statement (`_TYPE_ = 0`), then there is always exactly one level of output per BY group. If you use a `CLASS` statement, then the number of levels for each type that are requested have an upper bound equal to the number of observations in the input data set. By default, PROC MEANS generates all possible types. In this case the total number of levels for each BY group has an upper bound equal to

$$m \cdot (2^k - 1) \cdot n + 1$$

where k is the number of class variables and n is the number of observations for the given BY group in the input data set and m is 1, 5, or 6.

PROC MEANS determines the actual number of levels for a given type from the number of unique combinations of each active class variable. A single level consists of all input observations whose formatted class values match.

The following figure shows the values of `_TYPE_` and the number of observations in the data set when you specify one, two, and three class variables.

Figure 35.1 The Effect of Class Variables on the OUTPUT Data Set

					three CLASS variables	two CLASS variables	one CLASS variable	
C	B	A	WA	Y	_TYPE_	Subgroup defined by	Number of observations of this _TYPE_ and _WAY_ in the data set	Total number of observations in the data set
0	0	0	0	0	0	Total	1	
0	0	1	1	1	1	A	a	1+a
0	1	0	1	2	2	B	b	
0	1	1	2	3	3	A*B	a*b	1+a+b+a*b
1	0	0	1	4	4	C	c	
1	0	1	2	5	5	A*C	a*c	
1	1	0	2	6	6	B*C	b*c	1+a+b+a*b+c
1	1	1	3	7	7	A*B*C	a*b*c	+a*c+b*c+a*b*c
Character equivalent _TYPE_ (CHARTYPE option)		binary of				A , B ,C=CLASS variables	a, b, c,=number of levels of A, B, C, respectively	

- specifies the analysis variables
- computes the statistics for the specified keywords and displays them in order
- specifies the field width of the statistics

Program

```
options nodate pageno=1 linesize=80 pagesize=60;

data cake;
    input LastName $ 1-12 Age 13-14 PresentScore 16-17
           TasteScore 19-20 Flavor $ 23-32 Layers 34 ;
    datalines;
Orlando      27 93 80  Vanilla      1
Ramey        32 84 72  Rum          2
Goldston     46 68 75  Vanilla      1
Roe          38 79 73  Vanilla      2
Larsen       23 77 84  Chocolate   .
Davis        51 86 91  Spice        3
Strickland   19 82 79  Chocolate   1
Nguyen       57 77 84  Vanilla      .
Hildenbrand  33 81 83  Chocolate   1
Byron        62 72 87  Vanilla      2
Sanders      26 56 79  Chocolate   1
Jaeger       43 66 74           1
Davis        28 69 75  Chocolate   2
Conrad       69 85 94  Vanilla      1
Walters      55 67 72  Chocolate   2
Rossburger   28 78 81  Spice        2
Matthew      42 81 92  Chocolate   2
Becker       36 62 83  Spice        2
Anderson     27 87 85  Chocolate   1
Merritt      62 73 84  Chocolate   1
;

proc means data=cake n mean max min range std fw=8;
    var PresentScore TasteScore;
    title 'Summary of Presentation and Taste Scores';
run;
```

Program Description

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=60;
```

Create the CAKE data set. CAKE contains data from a cake-baking contest: each participant's last name, age, score for presentation, score for taste, cake flavor, and

number of cake layers. The number of cake layers is missing for two observations. The cake flavor is missing for another observation.

```
data cake;
  input LastName $ 1-12 Age 13-14 PresentScore 16-17
         TasteScore 19-20 Flavor $ 23-32 Layers 34 ;
  datalines;
Orlando      27 93 80  Vanilla      1
Ramey        32 84 72  Rum          2
Goldston     46 68 75  Vanilla      1
Roe          38 79 73  Vanilla      2
Larsen       23 77 84  Chocolate   .
Davis        51 86 91  Spice        3
Strickland   19 82 79  Chocolate   1
Nguyen       57 77 84  Vanilla      .
Hildenbrand  33 81 83  Chocolate   1
Byron        62 72 87  Vanilla      2
Sanders      26 56 79  Chocolate   1
Jaeger       43 66 74             1
Davis        28 69 75  Chocolate   2
Conrad       69 85 94  Vanilla      1
Walters      55 67 72  Chocolate   2
Rossburger   28 78 81  Spice        2
Matthew      42 81 92  Chocolate   2
Becker       36 62 83  Spice        2
Anderson     27 87 85  Chocolate   1
Merritt      62 73 84  Chocolate   1
;
```

Specify the statistics and the statistic options. The statistic keywords specify the statistics and their order in the output. FW= uses a field width of eight to display the statistics.

```
proc means data=cake n mean max min range std fw=8;
```

Specify the analysis variables. The VAR statement specifies that PROC MEANS calculate statistics on the PresentScore and TasteScore variables.

```
var PresentScore TasteScore;
```

Specify the title.

```
title 'Summary of Presentation and Taste Scores';
run;
```

Example 2: Computing Descriptive Statistics with Class Variables

Features: PROC MEANS statement option
MAXDEC=
CLASS statement
TYPES statement

Data set: [GRADE](#)

Details

This example does the following:

- analyzes the data for the two-way combination of class variables and across all observations
- limits the number of decimal places for the displayed statistics

Program

```
options nodate pageno=1 linesize=80 pagesize=60;

data grade;
    input Name $ 1-8 Gender $ 11 Status $13 Year $ 15-16
           Section $ 18 Score 20-21 FinalGrade 23-24;
    datalines;
Abbott      F 2 97 A 90 87
Branford    M 1 98 A 92 97
Crandell    M 2 98 B 81 71
Dennison    M 1 97 A 85 72
Edgar       F 1 98 B 89 80
Faust       M 1 97 B 78 73
Greeley     F 2 97 A 82 91
Hart        F 1 98 B 84 80
Isley       M 2 97 A 88 86
Jasper      M 1 97 B 91 93
    ;

proc means data=grade maxdec=3;
    var Score;
    class Status Year;
    types () status*year;
    title 'Final Exam Grades for Student Status and Year of Graduation';
run;
```

Program Description

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=60;
```

Create the GRADE data set. GRADE contains each student's last name, gender, status of either undergraduate (1) or graduate (2), expected year of graduation, class section (A or B), final exam score, and final grade for the course.

```
data grade;
  input Name $ 1-8 Gender $ 11 Status $13 Year $ 15-16
        Section $ 18 Score 20-21 FinalGrade 23-24;
  datalines;
Abbott   F 2 97 A 90 87
Branford M 1 98 A 92 97
Crandell M 2 98 B 81 71
Dennison M 1 97 A 85 72
Edgar    F 1 98 B 89 80
Faust    M 1 97 B 78 73
Greeley  F 2 97 A 82 91
Hart     F 1 98 B 84 80
Isley    M 2 97 A 88 86
Jasper   M 1 97 B 91 93
  ;
```

Generate the default statistics and specify the analysis options. Because no statistics are specified in the PROC MEANS statement, all default statistics (N, MEAN, STD, MIN, MAX) are generated. MAXDEC= limits the displayed statistics to three decimal places.

```
proc means data=grade maxdec=3;
```

Specify the analysis variable. The VAR statement specifies that PROC MEANS calculate statistics on the Score variable.

```
var Score;
```

Specify subgroups for the analysis. The CLASS statement separates the analysis into subgroups. Each combination of unique values for Status and Year represents a subgroup.

```
class Status Year;
```

Specify which subgroups to analyze. The TYPES statement requests that the analysis be performed on all the observations in the GRADE data set as well as the two-way combination of Status and Year, which results in four subgroups (because Status and Year each have two unique values).

```
types () status*year;
```

Specify the title.

```
title 'Final Exam Grades for Student Status and Year of Graduation';
run;
```

Output

PROC MEANS displays the default statistics for all the observations (_TYPE_=0) and the four class levels of the Status and Year combination (Status=1, Year=97; Status=1, Year=98; Status=2, Year=97; Status=2, Year=98).

Output 35.4 Final Exam Grades**Final Exam Grades for Student Status and Year of Graduation****The MEANS Procedure**

Analysis Variable : Score					
N Obs	N	Mean	Std Dev	Minimum	Maximum
10	10	86.000	4.714	78.000	92.000

Analysis Variable : Score							
Status	Year	N Obs	N	Mean	Std Dev	Minimum	Maximum
1	97	3	3	84.667	6.506	78.000	91.000
	98	3	3	88.333	4.041	84.000	92.000
2	97	3	3	86.667	4.163	82.000	90.000
	98	1	1	81.000	.	81.000	81.000

Example 3: Using the BY Statement with Class Variables

Features: PROC MEANS statement option
 statistic keywords
 BY statement
 CLASS statement
 SORT procedure

Data set: [GRADE](#)

Details

This example does the following:

- separates the analysis for the combination of class variables within BY values
- shows the sort order requirement for the BY statement
- calculates the minimum, maximum, and median

Program

```
options nodate pageno=1 linesize=80 pagesize=60;

proc sort data=Grade out=GradeBySection;
    by section;
run;

proc means data=GradeBySection min max median;
    by Section;
    var Score;
    class Status Year;
    title1 'Final Exam Scores for Student Status and Year of
Graduation';
    title2 ' Within Each Section';
run;
```

Program Description

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=60;
```

Sort the GRADE data set. PROC SORT sorts the observations by the variable Section. Sorting is required in order to use Section as a BY variable in the PROC MEANS step.

```
proc sort data=Grade out=GradeBySection;
    by section;
run;
```

Specify the analyses. The statistic keywords specify the statistics and their order in the output.

```
proc means data=GradeBySection min max median;
```

Divide the data set into BY groups. The BY statement produces a separate analysis for each value of Section.

```
    by Section;
```

Specify the analysis variable. The VAR statement specifies that PROC MEANS calculate statistics on the Score variable.

```
    var Score;
```

Specify subgroups for the analysis. The CLASS statement separates the analysis by the values of Status and Year. Because there is no TYPES statement in this program, analyses are performed for each subgroup, within each BY group.

```
    class Status Year;
```

Specify the titles.

```

title1 'Final Exam Scores for Student Status and Year of
Graduation';
title2 ' Within Each Section';
run;

```

Output

Output 35.5 Final Exam Scores

Final Exam Scores for Student Status and Year of Graduation Within Each Section

The MEANS Procedure

Section=A

Analysis Variable : Score					
Status	Year	N Obs	Minimum	Maximum	Median
1	97	1	85.0000000	85.0000000	85.0000000
	98	1	92.0000000	92.0000000	92.0000000
2	97	3	82.0000000	90.0000000	88.0000000

Section=B

Analysis Variable : Score					
Status	Year	N Obs	Minimum	Maximum	Median
1	97	2	78.0000000	91.0000000	84.5000000
	98	2	84.0000000	89.0000000	86.5000000
2	98	1	81.0000000	81.0000000	81.0000000

Example 4: Using a CLASSDATA= Data Set with Class Variables

Features:

PROC MEANS statement options
 CLASSDATA=
 EXCLUSIVE
 FW=
 MAXDEC=
 PRINTALLTYPES

CLASS statement

Data sets: [CAKE](#)
[CAKETYPE](#)

Details

This example does the following:

- specifies the field width and decimal places of the displayed statistics
- uses only the values in CLASSDATA= data set as the levels of the combinations of class variables
- calculates the range, median, minimum, and maximum
- displays all combinations of the class variables in the analysis

Program

```
options nodate pageno=1 linesize=80 pagesize=60;

data caketype;
  input Flavor $ 1-10 Layers 12;
  datalines;
Vanilla    1
Vanilla    2
Vanilla    3
Chocolate  1
Chocolate  2
Chocolate  3
;

proc means data=cake range median min max fw=7 maxdec=0
  classdata=caketype exclusive printalltypes;

  var TasteScore;

  class flavor layers;

  title 'Taste Score For Number of Layers and Cake Flavor';
run;
```

Program Description

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=60;
```

Create the CAKETYPE data set. CAKETYPE contains the cake flavors and number of layers that must occur in the PROC MEANS output.

```
data caketype;
  input Flavor $ 1-10 Layers 12;
  datalines;
Vanilla    1
Vanilla    2
Vanilla    3
Chocolate  1
Chocolate  2
Chocolate  3
;
```

Specify the analyses and the analysis options. The FW= option uses a field width of seven and the MAXDEC= option uses zero decimal places to display the statistics. CLASSDATA= and EXCLUSIVE restrict the class levels to the values that are in the CAKETYPE data set. PRINTALLTYPES displays all combinations of class variables in the output.

```
proc means data=cake range median min max fw=7 maxdec=0
          classdata=caketype exclusive printalltypes;
```

Specify the analysis variable. The VAR statement specifies that PROC MEANS calculate statistics on the TasteScore variable.

```
var TasteScore;
```

Specify subgroups for analysis. The CLASS statement separates the analysis by the values of Flavor and Layers. Note that these variables, and only these variables, must appear in the CAKETYPE data set.

```
class flavor layers;
```

Specify the title.

```
title 'Taste Score For Number of Layers and Cake Flavor';
run;
```

Output

PROC MEANS calculates statistics for the 13 chocolate and vanilla cakes. Because the CLASSDATA= data set contains 3 as the value of Layers, PROC MEANS uses 3 as a class value even though the frequency is zero.

Output 35.6 Taste Score**Taste Score For Number of Layers and Cake Flavor****The MEANS Procedure**

Analysis Variable : TasteScore				
N Obs	Range	Median	Minimum	Maximum
13	22	80	72	94

Analysis Variable : TasteScore					
Layers	N Obs	Range	Median	Minimum	Maximum
1	8	19	82	75	94
2	5	20	75	72	92
3	0	.	.	.	.

Analysis Variable : TasteScore					
Flavor	N Obs	Range	Median	Minimum	Maximum
Chocolate	8	20	81	72	92
Vanilla	5	21	80	73	94

Analysis Variable : TasteScore						
Flavor	Layers	N Obs	Range	Median	Minimum	Maximum
Chocolate	1	5	6	83	79	85
	2	3	20	75	72	92
	3	0	.	.	.	.
Vanilla	1	3	19	80	75	94
	2	2	14	80	73	87
	3	0	.	.	.	.

Example 5: Using Multilabel Value Formats with Class Variables

Features:

PROC MEANS statement options
 statistic keywords
 FW=
 NONOBS

CLASS statement options

MLF

ORDER=

TYPES statement

FORMAT procedure

FORMAT statement

Data set:

CAKE

Details

This example does the following:

- computes the statistics for the specified keywords and displays them in order
- specifies the field width of the statistics
- suppresses the column with the total number of observations
- analyzes the data for the one-way combination of cake flavor and the two-way combination of cake flavor and participant's age
- assigns user-defined formats to the class variables
- uses multilabel formats as the levels of class variables
- orders the levels of the cake flavors by the descending frequency count and orders the levels of age by the ascending formatted values

Program

```
options nodate pageno=1 linesize=80 pagesize=64;

proc format;
  value $flvrfmt
    'Chocolate'='Chocolate'
    'Vanilla'='Vanilla'
    'Rum','Spice'='Other Flavor';
  value agefmt (multilabel)
    15 - 29='below 30 years'
    30 - 50='between 30 and 50'
    51 - high='over 50 years'
    15 - 19='15 to 19'
    20 - 25='20 to 25'
    25 - 39='25 to 39'
    40 - 55='40 to 55'
    56 - high='56 and above';

run;

proc means data=cake fw=6 n min max median nonobs;
  class flavor/order=data;
  class age /mlf order=fmt;
```

```

types flavor flavor*age;

var TasteScore;

format age agefmt. flavor $flvrfmt.;

title 'Taste Score for Cake Flavors and Participant's Age';

run;

```

Program Description

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=64;
```

Create the \$FLVRFMT. and AGEFMT. formats. PROC FORMAT creates user-defined formats to categorize the cake flavors and ages of the participants. MULTILABEL creates a multilabel format for Age. A multilabel format is one in which multiple labels can be assigned to the same value, in this case because of overlapping ranges. Each value is represented in the output for each range in which it occurs.

```

proc format;
  value $flvrfmt
    'Chocolate'='Chocolate'
    'Vanilla'='Vanilla'
    'Rum','Spice'='Other Flavor';
  value agefmt (multilabel)
    15 - 29='below 30 years'
    30 - 50='between 30 and 50'
    51 - high='over 50 years'
    15 - 19='15 to 19'
    20 - 25='20 to 25'
    25 - 39='25 to 39'
    40 - 55='40 to 55'
    56 - high='56 and above';

run;

```

Specify the analyses and the analysis options. FW= uses a field width of six to display the statistics. The statistic keywords specify the statistics and their order in the output. NONOBS suppresses the N Obs column.

```
proc means data=cake fw=6 n min max median nonobs;
```

Specify subgroups for the analysis. The CLASS statements separate the analysis by values of Flavor and Age. ORDER=DATA orders values according to their order in the input data set. ORDER=FMT orders the levels of Age by ascending formatted values. MLF specifies that multilabel value formats be used for Age.

```

class flavor/order=data;
class age /mlf order=fmt;

```

Specify which subgroups to analyze. The TYPES statement requests the analysis for the one-way combination of Flavor and the two-way combination of Flavor and Age.

```
types flavor flavor*age;
```

Specify the analysis variable. The VAR statement specifies that PROC MEANS calculate statistics on the TasteScore variable.

```
var TasteScore;
```

Format the output. The FORMAT statement assigns user-defined formats to the Age and Flavor variables for this analysis.

```
format age agefmt. flavor $flvrfmt.;
```

Specify the title.

```
title 'Taste Score for Cake Flavors and Participant''s Age';  
run;
```

Output

The one-way combination of class variables appears before the two-way combination. A field width of six truncates the statistics to four decimal places. For the two-way combination of Age and Flavor, the total number of observations is greater than the one-way combination of Flavor. This situation arises because of the multilabel format for age, which maps one internal value to more than one formatted value. The order of the levels of Flavor is based on the frequency count for each level. The order of the levels of Age is based on the order of the user-defined formats.

Output 35.7 Taste Score for Cake Flavors**Taste Score for Cake Flavors and Participant's Age****The MEANS Procedure**

Analysis Variable : TasteScore				
Flavor	N	Minimum	Maximum	Median
Vanilla	6	73.00	94.00	82.00
Other Flavor	4	72.00	91.00	82.00
Chocolate	9	72.00	92.00	83.00

Analysis Variable : TasteScore					
Flavor	Age	N	Minimum	Maximum	Median
Vanilla	25 to 39	2	73.00	80.00	76.50
	40 to 55	1	75.00	75.00	75.00
	56 and above	3	84.00	94.00	87.00
	below 30 years	1	80.00	80.00	80.00
	between 30 and 50	2	73.00	75.00	74.00
	over 50 years	3	84.00	94.00	87.00
Other Flavor	25 to 39	3	72.00	83.00	81.00
	40 to 55	1	91.00	91.00	91.00
	below 30 years	1	81.00	81.00	81.00
	between 30 and 50	2	72.00	83.00	77.50
	over 50 years	1	91.00	91.00	91.00
Chocolate	15 to 19	1	79.00	79.00	79.00
	20 to 25	1	84.00	84.00	84.00
	25 to 39	4	75.00	85.00	81.00
	40 to 55	2	72.00	92.00	82.00
	56 and above	1	84.00	84.00	84.00
	below 30 years	5	75.00	85.00	79.00
	between 30 and 50	2	83.00	92.00	87.50
	over 50 years	2	72.00	84.00	78.00

Example 6: Using Preloaded Formats with Class Variables

Features:

- PROC MEANS statement options
 - COMPLETETYPES
 - FW=
 - MISSING
 - NONOBS
- CLASS statement options
 - EXCLUSIVE
 - ORDER=
 - PRELOADFMT
- WAYS statement
- FORMAT procedure
- FORMAT statement

Data set: [CAKE](#)

Details

This example does the following:

- specifies the field width of the statistics
- suppresses the column with the total number of observations
- includes all possible combinations of class variables values in the analysis even if the frequency is zero
- considers missing values as valid class levels
- analyzes the one-way and two-way combinations of class variables
- assigns user-defined formats to the class variables
- uses only the preloaded range of user-defined formats as the levels of class variables
- orders the results by the value of the formatted data

Program

```
options nodate pageno=1 linesize=80 pagesize=64;
proc format;
  value layerfmt 1='single layer'
```



```

                2-3='multi-layer'
                .='unknown';
value $flvrfmt (notsorted)
    'Vanilla'='Vanilla'
    'Orange','Lemon'='Citrus'
    'Spice'='Spice'
    'Rum','Mint','Almond'='Other Flavor';

run;

proc means data=cake fw=7 completetypes missing nonobs;
    class flavor layers/preloadfmt exclusive order=data;
    ways 1 2;
    var TasteScore;
    format layers layerfmt. flavor $flvrfmt.;
    title 'Taste Score For Number of Layers and Cake Flavors';
run;

```

Program Description

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=64;
```

Create the LAYERFMT. and \$FLVRFMT. formats. PROC FORMAT creates user-defined formats to categorize the number of cake layers and the cake flavors. NOTSORTED keeps \$FLVRFMT unsorted to preserve the original order of the format values.

```

proc format;
    value layerfmt 1='single layer'
                2-3='multi-layer'
                .='unknown';
    value $flvrfmt (notsorted)
        'Vanilla'='Vanilla'
        'Orange','Lemon'='Citrus'
        'Spice'='Spice'
        'Rum','Mint','Almond'='Other Flavor';

run;

```

Generate the default statistics and specify the analysis options. FW= uses a field width of seven to display the statistics. COMPLETETYPES includes class levels with a frequency of zero. MISSING considers missing values valid values for all class variables. NONOBS suppresses the N Obs column. Because no specific analyses are requested, all default analyses are performed.

```
proc means data=cake fw=7 completetypes missing nonobs;
```

Specify subgroups for the analysis. The CLASS statement separates the analysis by values of Flavor and Layers. PRELOADFMT and EXCLUSIVE restrict the levels

to the preloaded values of the user-defined formats. ORDER=DATA orders the levels of Flavor and Layer by formatted data values.

```
class flavor layers/preloadfmt exclusive order=data;
```

Specify which subgroups to analyze. The WAYS statement requests one-way and two-way combinations of class variables.

```
ways 1 2;
```

Specify the analysis variable. The VAR statement specifies that PROC MEANS calculate statistics on the TasteScore variable.

```
var TasteScore;
```

Format the output. The FORMAT statement assigns user-defined formats to the Flavor and Layers variables for this analysis.

```
format layers layerfmt. flavor $flvrfmt.;
```

Specify the title.

```
title 'Taste Score For Number of Layers and Cake Flavors';
run;
```

Output

The one-way combination of class variables appears before the two-way combination. PROC MEANS reports only the level values that are listed in the preloaded range of user-defined formats even when the frequency of observations is zero (in this case, citrus). PROC MEANS rejects entire observations based on the exclusion of any single class value in a given observation. Therefore, when the number of layers is unknown, statistics are calculated for only one observation. The other observation is excluded because the flavor chocolate was not included in the preloaded user-defined format for Flavor. The order of the levels is based on the order of the user-defined formats. PROC FORMAT automatically sorted the Layers format and did not sort the Flavor format.

Output 35.8 Taste Score for Number of Layers**Taste Score For Number of Layers and Cake Flavors****The MEANS Procedure**

Analysis Variable : TasteScore					
Layers	N	Mean	Std Dev	Minimum	Maximum
unknown	1	84.000	.	84.000	84.000
single layer	3	83.000	9.849	75.000	94.000
multi-layer	6	81.167	7.548	72.000	91.000

Analysis Variable : TasteScore					
Flavor	N	Mean	Std Dev	Minimum	Maximum
Vanilla	6	82.167	7.834	73.000	94.000
Citrus	0	.	.	.	.
Spice	3	85.000	5.292	81.000	91.000
Other Flavor	1	72.000	.	72.000	72.000

Analysis Variable : TasteScore						
Flavor	Layers	N	Mean	Std Dev	Minimum	Maximum
Vanilla	unknown	1	84.000	.	84.000	84.000
	single layer	3	83.000	9.849	75.000	94.000
	multi-layer	2	80.000	9.899	73.000	87.000
Citrus	unknown	0	.	.	.	.
	single layer	0	.	.	.	.
	multi-layer	0	.	.	.	.
Spice	unknown	0	.	.	.	.
	single layer	0	.	.	.	.
	multi-layer	3	85.000	5.292	81.000	91.000
Other Flavor	unknown	0	.	.	.	.
	single layer	0	.	.	.	.
	multi-layer	1	72.000	.	72.000	72.000

Example 7: Computing a Confidence Limit for the Mean

Features: PROC MEANS statement options
 ALPHA=
 FW=
 MAXDEC=
 CLASS statement

Data set: [Charity](#)

Details

This example does the following:

- specifies the field width and number of decimal places of the statistics
- computes a two-sided 90% confidence limit for the mean values of MoneyRaised and HoursVolunteered for the three years of data

If this data is representative of a larger population of volunteers, then the confidence limits provide ranges of likely values for the true population means.

Program

```
data charity;
  input School $ 1-7 Year 9-12 Name $ 14-20 MoneyRaised 22-26
        HoursVolunteered 28-29;
  datalines;
Monroe  2016 Allison 31.65 19
Monroe  2016 Barry  23.76 16
Monroe  2016 Candace 21.11 5

      . . . more data lines . . .
Kennedy 2018 Sid    27.45 25
Kennedy 2018 Will   28.88 21
Kennedy 2018 Morty  34.44 25
;

proc means data=charity fw=8 maxdec=2 alpha=0.1 clm mean std;
  class Year;
  var MoneyRaised HoursVolunteered;
  title 'Confidence Limits for Fund Raising Statistics';
  title2 '2016-18';
```

```
run;
```

Program Description

Create the CHARITY data set. CHARITY contains information about high-school students' volunteer work for a charity. The variables give the name of the high school, the year of the fund-raiser, the first name of each student, the amount of money each student raised, and the number of hours each student volunteered. A DATA step creates this data set.

```
data charity;
  input School $ 1-7 Year 9-12 Name $ 14-20 MoneyRaised 22-26
        HoursVolunteered 28-29;
  datalines;
Monroe  2016 Allison 31.65 19
Monroe  2016 Barry   23.76 16
Monroe  2016 Candace 21.11  5

      . . . more data lines . . .
Kennedy 2018 Sid     27.45 25
Kennedy 2018 Will    28.88 21
Kennedy 2018 Morty   34.44 25
;
```

Specify the analyses and the analysis options. FW= uses a field width of eight and MAXDEC= uses two decimal places to display the statistics. ALPHA=0.1 specifies a 90% confidence limit, and the CLM keyword requests two-sided confidence limits. MEAN and STD request the mean and the standard deviation, respectively.

```
proc means data=charity fw=8 maxdec=2 alpha=0.1 clm mean std;
```

Specify subgroups for the analysis. The CLASS statement separates the analysis by values of Year.

```
class Year;
```

Specify the analysis variables. The VAR statement specifies that PROC MEANS calculate statistics on the MoneyRaised and HoursVolunteered variables.

```
var MoneyRaised HoursVolunteered;
```

Specify the titles.

```
title 'Confidence Limits for Fund Raising Statistics';
title2 '2016-18';
run;
```

Output

PROC MEANS displays the lower and upper confidence limits for both variables for each year.

Output 35.9 Confidence Limits

Confidence Limits for Fund Raising Statistics 2016-18						
The MEANS Procedure						
Year	N Obs	Variable	Lower 90% CL for Mean	Upper 90% CL for Mean	Mean	Std Dev
2016	34	MoneyRaised	25.69	32.29	28.99	11.37
		HoursVolunteered	17.68	22.73	20.21	8.71
2017	29	MoneyRaised	24.61	31.61	28.11	11.09
		HoursVolunteered	15.67	20.19	17.93	7.17
2018	46	MoneyRaised	26.73	33.78	30.26	14.23
		HoursVolunteered	19.68	22.63	21.15	5.96

Example 8: Computing Output Statistics

Features:

- PROC MEANS statement option
NOPRINT
- CLASS statement
- OUTPUT statement options
statistic keywords
IDGROUP
LEVELS
WAYS
- PRINT procedure

Data set: [GRADE](#)

Details

This example does the following:

- suppresses the display of PROC MEANS output
- stores the average final grade in a new variable
- stores the name of the student with the best final exam scores in a new variable
- stores the number of class variables that are combined in the `_WAY_` variable
- stores the value of the class level in the `_LEVEL_` variable
- displays the output data set

Program

```
options nodate pageno=1 linesize=80 pagesize=60;

proc means data=Grade noprint;

    class Status Year;

    var FinalGrade;

    output out=sumstat mean=AverageGrade
           idgroup (max(score) obs out (name)=BestScore)
           / ways levels;

run;

proc print data=sumstat noobs;
    title1 'Average Undergraduate and Graduate Course Grades';
    title2 'For Two Years';
run;
```

Program Description

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=60;
```

Specify the analysis options. NOPRINT suppresses the display of all PROC MEANS output.

```
proc means data=Grade noprint;
```

Specify subgroups for the analysis. The CLASS statement separates the analysis by values of Status and Year.

```
class Status Year;
```

Specify the analysis variable. The VAR statement specifies that PROC MEANS calculate statistics on the FinalGrade variable.

```
var FinalGrade;
```

Specify the output data set options. The OUTPUT statement creates the SUMSTAT data set and writes the mean value for the final grade to the new variable AverageGrade. IDGROUP writes the name of the student with the top exam score to the variable BestScore and the observation number that contained the top score. WAYS and LEVELS write information about how the class variables are combined.

```
output out=sumstat mean=AverageGrade
       idgroup (max(score) obs out (name)=BestScore)
       / ways levels;

run;
```

Print the output data set WORK.SUMSTAT. The NOOBS option suppresses the observation numbers.

```
proc print data=sumstat noobs;
  title1 'Average Undergraduate and Graduate Course Grades';
  title2 'For Two Years';
run;
```

Output

The first observation contains the average course grade and the name of the student with the highest exam score over the two-year period. The next four observations contain values for each class variable value. The remaining four observations contain values for the Year and Status combination. The variables `_WAY_`, `_TYPE_`, and `_LEVEL_` show how PROC MEANS created the class variable combinations. The variable `_OBS_` contains the observation number in the GRADE data set that contained the highest exam score.

Output 35.10 Average Course Grades

Average Undergraduate and Graduate Course Grades For Two Years								
Status	Year	_WAY_	_TYPE_	_LEVEL_	_FREQ_	AverageGrade	BestScore	_OBS_
		0	0	1	10	83.0000	Branford	2
	97	1	1	1	6	83.6667	Jasper	10
	98	1	1	2	4	82.0000	Branford	2
1		1	2	1	6	82.5000	Branford	2
2		1	2	2	4	83.7500	Abbott	1
1	97	2	3	1	3	79.3333	Jasper	10
1	98	2	3	2	3	85.6667	Branford	2
2	97	2	3	3	3	88.0000	Abbott	1
2	98	2	3	4	1	71.0000	Crandell	3

Example 9: Computing Different Output Statistics for Several Variables

Features:

- PROC MEANS statement options
 - DESCEND
 - NOPRINT
- CLASS statement
- OUTPUT statement options

statistic keywords
 PRINT procedure
 WHERE= data set option

Data set:

GRADE

Details

This example does the following:

- suppresses the display of PROC MEANS output
- stores the statistics for the class level and combinations of class variables that are specified by WHERE= in the output data set
- orders observations in the output data set by descending _TYPE_ value
- stores the mean exam scores and mean final grades without assigning new variables names
- stores the median final grade in a new variable
- displays the output data set

Program

```
options nodate pageno=1 linesize=80 pagesize=60;
proc means data=Grade noprint descend;
    class Status Year;
    var Score FinalGrade;
    output out=Sumdata (where=(status='1' or _type_=0))
        mean= median(finalgrade)=MedianGrade;
run;
proc print data=Sumdata;
    title 'Exam and Course Grades for Undergraduates Only';
    title2 'and for All Students';
run;
```

Program Description

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=60;
```

Specify the analysis options. NOPRINT suppresses the display of all PROC MEANS output. DESCEND orders the observations in the OUT= data set by descending _TYPE_ value.

```
proc means data=Grade noprint descend;
```

Specify subgroups for the analysis. The CLASS statement separates the analysis by values of Status and Year.

```
class Status Year;
```

Specify the analysis variables. The VAR statement specifies that PROC MEANS calculate statistics on the Score and FinalGrade variables.

```
var Score FinalGrade;
```

Specify the output data set options. The OUTPUT statement writes the mean for Score and FinalGrade to variables of the same name. The median final grade is written to the variable MedianGrade. The WHERE= data set option restricts the observations in SUMDATA. One observation contains overall statistics (_TYPE_=0). The remainder must have a status of 1.

```
output out=Sumdata (where=(status='1' or _type_=0))
               mean= median(finalgrade)=MedianGrade;
run;
```

Print the output data set WORK.SUMDATA.

```
proc print data=Sumdata;
  title 'Exam and Course Grades for Undergraduates Only';
  title2 'and for All Students';
run;
```

Output

The first three observations contain statistics for the class variable levels with a status of 1. The last observation contains the statistics for all the observations (no subgroup). Score contains the mean test score and FinalGrade contains the mean final grade.

Output 35.11 Exam and Course Grades

Exam and Course Grades for Undergraduates Only and for All Students

Obs	Status	Year	_TYPE_	_FREQ_	Score	FinalGrade	MedianGrade
1	1	97	3	3	84.6667	79.3333	73
2	1	98	3	3	88.3333	85.6667	80
3	1		2	6	86.5000	82.5000	80
4			0	10	86.0000	83.0000	83

Example 10: Computing Output Statistics with Missing Class Variable Values

Features: PROC MEANS statement options
CHARTYPE
NOPRINT
NWAY
CLASS statement options
ASCENDING
MISSING
ORDER=
OUTPUT statement
PRINT procedure

Data set: [CAKE](#)

Details

This example does the following:

- suppresses the display of PROC MEANS output
- considers missing values as valid level values for only one class variable
- orders observations in the output data set by the ascending frequency for a single class variable
- stores observations for only the highest `_TYPE_` value
- stores `_TYPE_` as binary character values
- stores the maximum taste score in a new variable
- displays the output data set

Program

```
options nodate pageno=1 linesize=80 pagesize=60;
proc means data=cake chartype nway noprint;
    class flavor /order=freq ascending;
    class layers /missing;
    var TasteScore;
    output out=cakestat max=HighScore;
run;
```

```
proc print data=cakestat;
  title 'Maximum Taste Score for Flavor and Cake Layers';
run;
```

Program Description

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=60;
```

Specify the analysis options. NWAY prints observations with the highest _TYPE_ value. NOPRINT suppresses the display of all PROC MEANS output.

```
proc means data=cake chartype nway noprint;
```

Specify subgroups for the analysis. The CLASS statements separate the analysis by Flavor and Layers. ORDER=FREQ and ASCENDING order the levels of Flavor by ascending frequency. MISSING uses missing values of Layers as a valid class level value.

```
class flavor /order=freq ascending;
class layers /missing;
```

Specify the analysis variable. The VAR statement specifies that PROC MEANS calculate statistics on the TasteScore variable.

```
var TasteScore;
```

Specify the output data set options. The OUTPUT statement creates the CAKESTAT data set and generates the maximum value for the taste score to the new variable HighScore.

```
output out=cakestat max=HighScore;
run;
```

Print the output data set WORK.CAKESTAT.

```
proc print data=cakestat;
  title 'Maximum Taste Score for Flavor and Cake Layers';
run;
```

Output

The CAKESTAT output data set contains only observations for the combination of the two class variables, Flavor and Layers. Therefore, the value of _TYPE_ is 11 for all observations. The observations are ordered by ascending frequency of Flavor. The missing value in Layers is a valid value for this class variable. PROC MEANS excludes the observation with the missing flavor because it is an invalid value for Flavor.

Output 35.12 Maximum Taste Score**Maximum Taste Score for Flavor and Cake Layers**

Obs	Flavor	Layers	_TYPE_	_FREQ_	HighScore
1	Rum	2	11	1	72
2	Spice	2	11	2	83
3	Spice	3	11	1	91
4	Vanilla	.	11	1	84
5	Vanilla	1	11	3	94
6	Vanilla	2	11	2	87
7	Chocolate	.	11	1	84
8	Chocolate	1	11	5	85
9	Chocolate	2	11	3	92

Example 11: Identifying an Extreme Value with the Output Statistics

Features: CLASS statement
 OUTPUT statement options
 statistic keyword
 MAXID
 PRINT procedure

Data set: [Charity](#)

Details

This example does the following:

- identifies the observations with maximum values for two variables
- creates new variables for the maximum values
- displays the output data set

Program

```
proc means data=Charity n mean range chartype;
    class School Year;
    var MoneyRaised HoursVolunteered;
    output out=Prize maxid(MoneyRaised(name)
        HoursVolunteered(name))= MostCash MostTime
        max= ;

    title 'Summary of Volunteer Work by School and Year';
run;

proc print data=Prize;
    title 'Best Results: Most Money Raised and Most Hours Worked';
run;
```

Program Description

Specify the analyses. The statistic keywords specify the statistics and their order in the output. CHARTYPE writes the _TYPE_ values as binary characters in the output data set

```
proc means data=Charity n mean range chartype;
```

Specify subgroups for the analysis. The CLASS statement separates the analysis by School and Year.

```
class School Year;
```

Specify the analysis variables. The VAR statement specifies that PROC MEANS calculate statistics on the MoneyRaised and HoursVolunteered variables.

```
var MoneyRaised HoursVolunteered;
```

Specify the output data set options. The OUTPUT statement writes the new variables, MostCash, and MostTime, which contain the names of the students who collected the most money and volunteered the most time, respectively, to the PRIZE data set.

```
output out=Prize maxid(MoneyRaised(name)
    HoursVolunteered(name))= MostCash MostTime
    max= ;
```

Specify the title.

```
title 'Summary of Volunteer Work by School and Year';
run;
```

Print the WORK.PRIZE output data set.

```
proc print data=Prize;
    title 'Best Results: Most Money Raised and Most Hours Worked';
run;
```

Output

The first page of output shows the output from PROC MEANS with the statistics for six class levels: one for Monroe High for the years 1992, 1993, and 1994; and one for Kennedy High for the same three years.

Output 35.13 Summary of Volunteer Work

Summary of Volunteer Work by School and Year						
The MEANS Procedure						
School	Year	N Obs	Variable	N	Mean	Range
Kennedy	2007	18	MoneyRaised	18	29.3811111	39.7500000
			HoursVolunteered	18	21.4444444	30.0000000
	2008	17	MoneyRaised	17	28.1564706	23.5600000
			HoursVolunteered	17	19.4117647	20.0000000
	2009	18	MoneyRaised	18	31.5794444	65.4400000
			HoursVolunteered	18	24.2777778	15.0000000
Monroe	2007	16	MoneyRaised	16	28.5450000	48.2700000
			HoursVolunteered	16	18.8125000	38.0000000
	2008	12	MoneyRaised	12	28.0500000	52.4600000
			HoursVolunteered	12	15.8333333	21.0000000
	2009	28	MoneyRaised	28	29.4100000	73.5300000
			HoursVolunteered	28	19.1428571	26.0000000

The output from PROC PRINT shows the maximum MoneyRaised and HoursVolunteered values and the names of the students who are responsible for them. The first observation contains the overall results, the next three contain the results by year, the next two contain the results by school, and the final six contain the results by School and Year.

Output 35.14 Best Results

Best Results: Most Money Raised and Most Hours Worked								
Obs	School	Year	_TYPE_	_FREQ_	MostCash	MostTime	MoneyRaised	HoursVolunteered
1		.	00	109	Willard	Tonya	78.65	40
2		2007	01	34	Tonya	Tonya	55.16	40
3		2008	01	29	Cameron	Amy	65.44	31
4		2009	01	46	Willard	L.T.	78.65	33
5	Kennedy	.	10	53	Luther	Jay	72.22	35
6	Monroe	.	10	56	Willard	Tonya	78.65	40
7	Kennedy	2007	11	18	Thelma	Jay	52.63	35
8	Kennedy	2008	11	17	Bill	Amy	42.23	31
9	Kennedy	2009	11	18	Luther	Che-Min	72.22	33
10	Monroe	2007	11	16	Tonya	Tonya	55.16	40
11	Monroe	2008	11	12	Cameron	Myrtle	65.44	26
12	Monroe	2009	11	28	Willard	L.T.	78.65	33

Example 12: Identifying the Top Three Extreme Values with the Output Statistics

Features:

- PROC MEANS statement option
NOPRINT
- CLASS statement
- OUTPUT statement options
statistic keywords
AUTOLABEL
AUTONAME
IDGROUP
- TYPES statement
- FORMAT procedure
- FORMAT statement
- PRINT procedure
- RENAME= data set option

Data set: [Charity](#)

Details

This example does the following:

- suppresses the display of PROC MEANS output
- analyzes the data for the one-way combination of the class variables and across all observations
- stores the total and average amount of money raised in new variables
- stores in new variables the top three amounts of money raised, the names of the three students who raised the money, the years when it occurred, and the schools the students attended
- automatically resolves conflicts in the variable names when names are assigned to the new variables in the output data set
- appends the statistic name to the label of the variables in the output data set that contain statistics that were computed for the analysis variable
- assigns a format to the analysis variable so that the statistics that are computed from this variable inherit the attribute in the output data set
- renames the `_FREQ_` variable in the output data set
- displays the output data set and its contents

Program

```
proc format;
  value yrFmt . = " All";
  value $schFmt ' ' = "All    ";
run;

proc means data=Charity noprint;
  class School Year;
  types () school year;
  var MoneyRaised;

  output out=top3list(rename=(_freq_=NumberStudents)) sum= mean=
    idgroup( max(moneyraised) out[3] (moneyraised name
      school year)=)/autolabel autoname;

  label MoneyRaised='Amount Raised';
  format year yrfmt. school $schfmt.
    moneyraised dollar8.2;
run;

proc print data=top3list;
  title1 'School Fund Raising Report';
  title2 'Top Three Students';
run;
```

```
proc datasets library=work nolist;
  contents data=top3list;
  title1 'Contents of the PROC MEANS Output Data Set';
run;
```

Program Description

Create the YRFMT. and \$SCHFMT. formats. PROC FORMAT creates user-defined formats that assign the value of **all** to the missing levels of the class variables.

```
proc format;
  value yrFmt . = " All";
  value $schFmt ' ' = "All    ";
run;
```

Generate the default statistics and specify the analysis options. NOPRINT suppresses the display of all PROC MEANS output.

```
proc means data=Charity noprint;
```

Specify subgroups for the analysis. The CLASS statement separates the analysis by values of School and Year.

```
class School Year;
```

Specify which subgroups to analyze. The TYPES statement requests the analysis across all the observations and for each one-way combination of School and Year.

```
types () school year;
```

Specify the analysis variable. The VAR statement specifies that PROC MEANS calculate statistics on the MoneyRaised variable.

```
var MoneyRaised;
```

Specify the output data set options. The OUTPUT statement creates the TOP3LIST data set. RENAME= renames the _FREQ_ variable that contains the frequency count for each class level. SUM= and MEAN= specify that the sum and mean of the analysis variable (MoneyRaised) are written to the output data set. IDGROUP writes 12 variables that contain the top three amounts of money raised and the three corresponding students, schools, and years. AUTOLABEL appends the analysis variable name to the label for the output variables that contain the sum and mean. AUTONAME resolves naming conflicts for these variables.

```
output out=top3list(rename=(_freq_=NumberStudents))sum= mean=
  idgroup( max(moneyraised) out[3] (moneyraised name
    school year)=)/autolabel autoname;
```

Format the output. The LABEL statement assigns a label to the analysis variable MoneyRaised. The FORMAT statement assigns user-defined formats to the Year and School variables and a SAS dollar format to the MoneyRaised variable.

```
label MoneyRaised='Amount Raised';
format year yrfmt. school $schfmt.
  moneyraised dollar8.2;
run;
```

Print the output data set WORK.TOP3LIST.

```
proc print data=top3list;
  title1 'School Fund Raising Report';
  title2 'Top Three Students';
run;
```

Display information about the TOP3LIST data set. PROC DATASETS displays the contents of the TOP3LIST data set. NOLIST suppresses the directory listing for the WORK data library.

```
proc datasets library=work nolist;
  contents data=top3list;
  title1 'Contents of the PROC MEANS Output Data Set';
run;
```

Output

The output from PROC PRINT shows the top three values of MoneyRaised, the names of the students who raised these amounts, the schools the students attended, and the years when the money was raised. The first observation contains the overall results, the next three contain the results by year, and the final two contain the results by school. The missing class levels for School and Year are replaced with the value ALL. The labels for the variables that contain statistics that were computed from MoneyRaised include the statistic name at the end of the label.

Output 35.15 School Fund Raising

MoneyRaised_Sum	MoneyRaised_Mean	MoneyRaised_1	MoneyRaised_2	MoneyRaised_3	Name_1	Name_2	Name_3	School_1
\$3192.75	\$29.29	\$78.65	\$72.22	\$65.44	Willard	Luther	Cameron	Monroe
\$892.92	\$28.80	\$55.16	\$53.76	\$52.63	Tonya	Edward	Thelma	Monroe
\$907.92	\$28.37	\$65.44	\$47.33	\$42.23	Cameron	Myrtle	Bill	Monroe
\$1391.91	\$30.26	\$78.65	\$72.22	\$56.87	Willard	Luther	L.T.	Monroe
\$1575.95	\$29.73	\$72.22	\$52.63	\$43.89	Luther	Thelma	Jenny	Kennedy
\$1616.80	\$28.87	\$78.65	\$65.44	\$56.87	Willard	Cameron	L.T.	Monroe

See the TEMPLATE procedure in *SAS Output Delivery System: User's Guide* for an example of how to create a custom table template for this output data set.

Output 35.16 PROC MEANS Output Data Set

School Fund Raising Report Top Three Students										
Obs	School	Year	_TYPE_	NumberStudents	MoneyRaised_Sum	MoneyRaised_Mean	MoneyRaised_1	MoneyRaised_2	MoneyRaised_3	Name_1
1	All	All	0	109	\$3192.75	\$29.29	\$78.65	\$72.22	\$65.44	Willard
2	All	2007	1	34	\$985.58	\$28.99	\$55.16	\$53.76	\$52.63	Tonya
3	All	2008	1	29	\$815.26	\$28.11	\$65.44	\$47.33	\$42.23	Cameron
4	All	2009	1	46	\$1391.91	\$30.26	\$78.65	\$72.22	\$56.87	Willard
5	Kenn	All	2	53	\$1575.95	\$29.73	\$72.22	\$52.63	\$43.89	Luther
6	Monr	All	2	56	\$1616.80	\$28.87	\$78.65	\$65.44	\$56.87	Willard

Contents of the PROC MEANS Output Data Set**The DATASETS Procedure**

Data Set Name	WORK.TOP3LIST	Observations	6
Member Type	DATA	Variables	18
Engine	V9	Indexes	0
Created	Tuesday, July 05, 2011 10:24:55 AM	Observation Length	144
Last Modified	Tuesday, July 05, 2011 10:24:55 AM	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	WINDOWS_32		
Encoding	wlatin1 Western (Windows)		

Engine/Host Dependent Information	
Data Set Page Size	12288
Number of Data Set Pages	1
First Data Page	1
Max Obs per Page	85
Obs in First Data Page	6
Number of Data Set Repairs	0
Filename	C:\DOCUME~1\lirezn\LOCALS~1\Temp\SAS Temporary Files_TD3456_d21560_top3list.sas7bdat
Release Created	9.0301M0
Host Created	XP_PRO

Alphabetic List of Variables and Attributes					
#	Variable	Type	Len	Format	Label
7	MoneyRaised_1	Num	8	DOLLAR8.2	Amount Raised
8	MoneyRaised_2	Num	8	DOLLAR8.2	Amount Raised
9	MoneyRaised_3	Num	8	DOLLAR8.2	Amount Raised
6	MoneyRaised_Mean	Num	8	DOLLAR8.2	Amount Raised_Mean
5	MoneyRaised_Sum	Num	8	DOLLAR8.2	Amount Raised_Sum
10	Name_1	Char	7		
11	Name_2	Char	7		
12	Name_3	Char	7		
4	NumberStudents	Num	8		
1	School	Char	7	\$\$CHFMT.	
13	School_1	Char	7	\$\$CHFMT.	
14	School_2	Char	7	\$\$CHFMT.	
15	School_3	Char	7	\$\$CHFMT.	
2	Year	Num	8	YRFMT.	
16	Year_1	Num	8	YRFMT.	
17	Year_2	Num	8	YRFMT.	
18	Year_3	Num	8	YRFMT.	
3	_TYPE_	Num	8		

Example 13: Using the STACKODSOUTPUT Option to Control Data

Features: PROC PRINT
PROC CONTENTS

The first example does not use the STACKODSOUTPUT option. The second example uses the STACKODSOUTPUT option.

Program

```
proc means data=sashelp.class;
class sex;
var weight height;
ods output summary=default;
run;

proc print data=default; run;
proc contents data=default; run;
```

Program Description

This code processes the data without using the STACKODSOUTPUT option.

```
proc means data=sashelp.class;
class sex;
var weight height;
ods output summary=default;
run;
```

Prints the data using PROC PRINT. Print the contents of the procedure using PROC CONTENTS.

```
proc print data=default; run;
proc contents data=default; run;
```

OUTPUT

The following outputs show the difference in processing data using the STACKODSOUTPUT option and then not using the option.

This output is generated without the STACKODSOUTPUT option.

Output 35.17 Output without Using the STACKODSOUTPUT Option

The SAS System							
The MEANS Procedure							
Sex	N Obs	Variable	N	Mean	Std Dev	Minimum	Maximum
F	9	Weight	9	90.1111111	19.3839137	50.5000000	112.5000000
		Height	9	60.5888889	5.0183275	51.3000000	66.5000000
M	10	Weight	10	108.9500000	22.7271864	83.0000000	150.0000000
		Height	10	63.9100000	4.9379370	57.3000000	72.0000000

The SAS System									
Obs	Sex	NObs	VName_Weight	Weight_N	Weight_Mean	Weight_StdDev	Weight_Min	Weight_Max	VName_Height
1	F	9	Weight	9	90.1111111111	19.38391372	50.5	112.5	Height
2	M	10	Weight	10	108.95	22.727186363	83	150	Height

The SAS System

The CONTENTS Procedure

Data Set Name	WORK.DEFAULT	Observations	2
Member Type	DATA	Variables	14
Engine	V9	Indexes	0
Created	Sunday, January 23, 2011 02:48:10 PM	Observation Length	104
Last Modified	Sunday, January 23, 2011 02:48:10 PM	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label	Summary statistics		
Data Representation	WINDOWS_32		
Encoding	wlatin1 Western (Windows)		

Engine/Host Dependent Information	
Data Set Page Size	12288
Number of Data Set Pages	1
First Data Page	1
Max Obs per Page	117
Obs in First Data Page	2
Number of Data Set Repairs	0
Filename	C:\DOCUME~1\lirezn\LOCALS~1\Temp\SAS Temporary Files_TD3828_d21560_\default.sas7bdat
Release Created	9.0301B0
Host Created	XP_PRO

Alphabetic List of Variables and Attributes					
#	Variable	Type	Len	Format	Label
14	Height_Max	Num	8	BEST12.	Maximum
11	Height_Mean	Num	8	BEST12.	Mean
13	Height_Min	Num	8	BEST12.	Minimum
10	Height_N	Num	8	BEST2.	N
12	Height_StdDev	Num	8	BEST12.	Std Dev
2	NObs	Num	8	BEST2.	N Obs
1	Sex	Char	1		
9	VName_Height	Char	6		Variable
3	VName_Weight	Char	6		Variable
8	Weight_Max	Num	8	BEST12.	Maximum
5	Weight_Mean	Num	8	BEST12.	Mean
7	Weight_Min	Num	8	BEST12.	Minimum
4	Weight_N	Num	8	BEST2.	N
6	Weight_StdDev	Num	8	BEST12.	Std Dev

This code processes the data using the STACKODSOUTPUT option.

```
proc means data=sashelp.class STACKODSOUTPUT;
```



```

class sex;
var weight height;
ods output summary=stacked;
run;

```

Print the data using PROC PRINT. Print the contents of the procedure using PROC CONTENTS.

```

proc print data=stacked; run;
proc contents data=stacked; run;

```

This output is generated with the STACKODSOUTPUT option.

Output 35.18 *Output Using the STACKODSOUTPUT Option*

The SAS System

The MEANS Procedure

Sex	N Obs	Variable	N	Mean	Std Dev	Minimum	Maximum
F	9	Weight	9	90.111111	19.383914	50.500000	112.500000
		Height	9	60.588889	5.018328	51.300000	66.500000
M	10	Weight	10	108.950000	22.727186	83.000000	150.000000
		Height	10	63.910000	4.937937	57.300000	72.000000

The SAS System

Obs	Sex	NObs	_control_	Variable	N	Mean	StdDev	Min	Max
1	F	9		Weight	9	90.111111	19.383914	50.500000	112.500000
2	F	9		Height	9	60.588889	5.018328	51.300000	66.500000
3	M	10	1	Weight	10	108.950000	22.727186	83.000000	150.000000
4	M	10		Height	10	63.910000	4.937937	57.300000	72.000000

The SAS System

The CONTENTS Procedure

Data Set Name	WORK.STACKED	Observations	4
Member Type	DATA	Variables	9
Engine	V9	Indexes	0
Created	Sunday, January 23, 2011 03:19:07 PM	Observation Length	56
Last Modified	Sunday, January 23, 2011 03:19:07 PM	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label	Summary statistics		
Data Representation	WINDOWS_32		
Encoding	wlatin1 Western (Windows)		

Engine/Host Dependent Information	
Data Set Page Size	8192
Number of Data Set Pages	1
First Data Page	1
Max Obs per Page	145
Obs in First Data Page	4
Number of Data Set Repairs	0
Filename	C:\DOCUME~1\lirezn\LOCALS~1\Temp\SAS Temporary Files_TD5560_d21560_stacked.sas7bdat
Release Created	9.0301B0
Host Created	XP_PRO

Alphabetic List of Variables and Attributes					
#	Variable	Type	Len	Format	Label
9	Max	Num	8	D12.3	Maximum
6	Mean	Num	8	D12.3	
8	Min	Num	8	D12.3	Minimum
5	N	Num	8	BEST2.	
2	NObs	Num	8	BEST2.	N Obs
1	Sex	Char	1		
7	StdDev	Num	8	D12.3	Std Dev
4	Variable	Char	6		
3	_control_	Char	1		

References

Jain R., and I. Chlamtac. 1985. "The P^2 Algorithm for Dynamic Calculation of Quantiles and Histograms without Sorting Observations." *Communications of the Association of Computing Machinery* 28 (10): 1076–0185.

MIGRATE Procedure

Overview: MIGRATE Procedure	1343
What Does the MIGRATE Procedure Do?	1344
Concepts: MIGRATE Procedure	1344
Data Sets	1344
Views	1345
Catalogs	1346
MDDBs	1347
Item Stores	1347
Not Supported	1347
Syntax: MIGRATE Procedure	1347
PROC MIGRATE Statement	1348
Usage: MIGRATE Procedure	1350
Migrating with the CVP Engine to Avoid Truncation	1350
Using a SAS/CONNECT Server	1351
Migrating a Data Set with Audit Trails, Generations, Indexes, or Integrity Constraints	1352
Migrating a SAS Data Set with NODUPKEY Sort Indicator	1353
Additional Steps for Unsupported Catalogs	1354
Examples: MIGRATE Procedure	1355
Example 1: Migrate with Direct Access	1355
Example 2: Migrate from SAS®9 by Using SAS/CONNECT	1356
Example 3: Migrate with the CVP Engine to Avoid Truncation	1358
Example 4: Migrate When a Catalog Is from a Different Encoding	1359

Overview: MIGRATE Procedure

What Does the MIGRATE Procedure Do?

The procedure migrates members of a library to the current release of SAS software. The source and target libraries must be V9 engine libraries.

The procedure migrates the following library members:

- data sets with alternate collating sequence, audit trails, compression, created and modified datetimes, deleted observations, encryption, extended attributes, generations, indexes, integrity constraints, and passwords
- in many cases, views, catalogs, item stores, and multidimensional databases (MDDBs) (see [Concepts: MIGRATE Procedure on page 1344](#))

The procedure does not support stored compiled DATA step programs or stored compiled macros. (Instead, move the source code to the target, where you can compile and store it.) The procedure does not support SAS program files. The procedure does not support Scalable Performance Data (SPD) engine data sets. (See [SAS Scalable Performance Data Engine: Reference](#).) The procedure does not support the extended observation count attribute.

Concepts: MIGRATE Procedure

Data Sets

PROC MIGRATE retains alternate collating sequence, compression, created and modified datetimes, deleted observations, encryption, extended attributes, indexes, integrity constraints, and passwords. See several important restrictions [at the beginning of the syntax on page 1347](#) and in the syntax for each option.

The audit trail and generations are also migrated. Indexes and integrity constraints are rebuilt on the member in the target library. See [“Migrating a Data Set with Audit Trails, Generations, Indexes, or Integrity Constraints” on page 1352](#).

Migrated data sets take on the data representation and encoding attributes of the target library. When you migrate a data set to an encoding where the characters are represented by more bytes, truncation might occur if the column length does not accommodate the larger character size. For example, a character might be represented in Wlatin1 encoding as one byte but in UTF-8 as two bytes. The best solution is to expand the column length of data sets in the library by using the CVP engine during migration. See [“Migrating with the CVP Engine to Avoid Truncation” on page 1350](#).

For encrypted data sets, created and modified datetime values are not retained. Current datetime values are used. For AES- or AES2-encrypted data sets, you must run SAS interactively in order to provide ENCRYPTKEY= values. PROC MIGRATE cannot automatically access the AES or AES2 encryption key. For SAS Proprietary-encrypted data sets, or password-protected data sets, you can run SAS in batch. PROC MIGRATE can automatically access the password. In all cases, if the migration is successful, the encryption, encryption key, and passwords are retained in the target library.

For linguistically sorted data sets, the International Components for Unicode (ICU) version number might be different from that of the current session. In that case, PROC MIGRATE retains the data set's sort order in the OUT= destination library. However, the data set is no longer marked as sorted, and a message is written to the SAS log. For more information about linguistic sorting, see [Chapter 55, “SORT Procedure,” on page 2039](#).

Views

As with data sets, migrated data views take on the data representation and encoding attributes of the target library.

DATA Step Views

When you create a DATA step view, you can specify the SOURCE= option to store the DATA step code along with the view. PROC MIGRATE supports DATA step views with stored code. The stored code is recompiled the first time the DATA step view is accessed by SAS in the target environment. PROC MIGRATE does not support DATA step views that were created prior to SAS 8 or DATA step views without stored code. For DATA step views without stored code, use the DESCRIBE statement in the source session to recover the DATA step code. Then submit the DATA step code in the target session and recompile it.

PROC SQL Views

Be aware that embedded librefs associated with a view are not updated during migration. The following example illustrates the issue. In this example, Lib1.MyView contains a view of the data set Lib1.MyData:

```
data Lib1.MyData;x=1;
run;
proc sql;
    create view Lib1.MyView as select * from Lib1.MyData;
quit;
```

After you migrate Lib1 to Lib2, you have Lib2.MyView and Lib2.MyData. However, because Lib2.MyView was originally created with an embedded libref

of Lib1, it still references the data set Lib1.MyData, not Lib2.MyData. The following example fails with an error message that Lib1 cannot be found:

```
proc print data=Lib2.MyView;
run;
```

Catalogs

To migrate catalogs, PROC MIGRATE calls PROC CPORT and PROC CIMPORT. You might notice that CPORT and CIMPORT notes are written to the SAS log during migration. PROC CPORT and CIMPORT restrictions apply. Stored compiled macros that are stored in catalogs are not supported. (Instead, move the source code to the target, where you can compile and store it.) Catalog entries might need to be updated after migrating (for example, code that contains hardcoded pathnames).

If catalogs were created under a different encoding or data representation than the target environment, then transcoding errors can occur during migration. A best practice is to know the encoding and data representation of the session where the catalogs were created. Catalogs do not maintain metadata about the encoding or data representation, so you cannot use PROC DATASETS to learn those attributes of a catalog. The following options are available to avoid these transcoding errors:

- If the current session encoding is UTF-8, and your user-defined formats catalogs were created under non-UTF-8 sessions and contain national characters, then you can use PROC MIGRATE with the ENCODING= option. The option enables PROC MIGRATE to transcode catalog entries for user-defined formats. The lengths of the formats are expanded if necessary to avoid truncation. Other library members are migrated, but this option does not apply to them. You must specify the encoding of the session where the catalog was created. See [“ENCODING=encoding-value” on page 1349](#) and [“Example 4: Migrate When a Catalog Is from a Different Encoding” on page 1359](#).
- If the data representation is not compatible with the target session, then SAS/CONNECT is also required. You specify the server in the IN= option or the SLIBREF= option. See [“Using a SAS/CONNECT Server” on page 1351](#). If you do not have access to SAS/CONNECT software, then use the process described in [“Additional Steps for Unsupported Catalogs” on page 1354](#).
- If data sets are present in the library, you can use the CVP engine to expand lengths in the data. The CVP engine does not transcode catalog entries or expand the lengths in formats. See [“Migrating with the CVP Engine to Avoid Truncation” on page 1350](#).

Here are two ways to determine whether a SAS library contains catalogs:

- Use operating system tools to examine the file system. The file extension for SAS catalogs is `.sas7bcat`.
- Submit a DATASETS procedure with MEMTYPE=CAT. Examine the SAS log for a list of catalogs.

MDDBs

PROC MIGRATE supports MDDBs with no known issues.

Item Stores

PROC MIGRATE supports item stores unless you migrate from a 32-bit to a 64-bit environment. In that case, an error message is not written to the SAS log, but item stores might not work correctly in the target library.

Therefore, PROC MIGRATE is usually not a best practice. For more details, see [“SAS Files That Can and Cannot Be Migrated” in SAS V9 LIBNAME Engine: Reference](#).

Not Supported

PROC MIGRATE does not support stored compiled DATA step programs or stored compiled macros. (Instead, move the source code to the target, where you can compile and store it.) PROC MIGRATE does not support SAS program files. PROC MIGRATE does not support SPD Engine data sets. (See [SAS Scalable Performance Data Engine: Reference](#).) See also restrictions above for each member type.

Syntax: MIGRATE Procedure

Restrictions:

SAS data set options are not supported with PROC MIGRATE.

When PROC MIGRATE creates a member in the target library, the member does not retain its permissions from the source library. It has the permissions that are associated with the user ID of the person who ran PROC MIGRATE.

PROC MIGRATE IN=*libref-1* OUT=*libref-2* <*options*>;

Statement	Task	Example
PROC MIGRATE	Migrate members in a SAS library to the current SAS version	Ex. 1 , Ex. 2 , Ex. 3 , Ex. 4

PROC MIGRATE Statement

Migrates a SAS library forward to the current release of SAS software.

Syntax

```
PROC MIGRATE IN=libref-1 OUT=libref-2
< BUFSIZE=KEEPSIZE | n | nK | nM | nG >
< ENCODING=encoding-value >
< MOVE >
< SLIBREF=libref >
< KEEPNOUPKEY >;
```

Required Arguments

IN=*libref-1*

names the source SAS library from which to migrate members. To determine whether you must specify a SAS/CONNECT libref in the IN= option or in the SLIBREF= option, see [“Using a SAS/CONNECT Server” on page 1351](#).

Restriction Concatenated libraries are not supported. See SAS Note 24539 at <https://support.sas.com/kb/24/539.html>.

Requirement Assign the IN= source library to a different physical location than the OUT= target library.

OUT=*libref-2*

names the target SAS library to contain the migrated members.

Restriction Do not assign the OUT= target library to a SAS/CONNECT server. The REMOTE engine is not supported for the target library. The following example causes an error:

```
libname Lib1 'source-library-pathname';
libname Lib2 server=server-id;
proc migrate in=Lib1 out=Lib2;
run;
```

Requirement Assign the OUT= target library to a different physical location than the IN= source library.

Interaction PROC MIGRATE can use the LIBNAME option OUTREP= for DATA, VIEW, ACCESS, MDDB, and DMDB member types. If you specify the OUTREP= option, you might also want to specify the EXTENDOBSCOUNTER= option. These options are appropriate in the LIBNAME statement for the OUT= library. See [SAS DATA Step Statements: Reference](#).

Tip Assign the target library to an empty location. If a member already exists in the target library that has the same name and member type as a member in the source library, the member is not migrated. An error message is written to the SAS log, and PROC MIGRATE continues with the next member.

Optional Arguments

BUFSIZE=KEEPSIZE | *n* | *nK* | *nM* | *nG*

specifies the buffer page size of the members that are written to the target library. For example, a value of 10000 specifies a page size of 10,000 bytes, and a value of 4k specifies a page size of 4096 bytes. A value of 0 results in the default. Setting the page size can help optimize SAS performance.

For more details about the BUFSIZE= data set option or system option, see the documentation for your operating environment.

KEEPSIZE

retains (clones) the page size of members from the source library.

n

specifies the number of bytes.

nK

specifies the number of kilobytes.

nM

specifies the number of megabytes.

nG

specifies the number of gigabytes.

Default the buffer page size of the current session

ENCODING=*encoding-value*

specifies the session encoding that was used when formats catalogs were created.

Specify this option for user-defined formats catalogs that were created under a different encoding than the target session. The option enables PROC MIGRATE to transcode catalog entries for user-defined formats. The lengths of the formats are expanded if necessary to avoid truncation. Other library members are migrated, but this option does not apply to them. See [“Example 4: Migrate When a Catalog Is from a Different Encoding” on page 1359](#).

Restrictions DBCS characters in format names or function names are not supported.

SAS Version 6 user-defined formats are not supported.

Requirement The encoding of the current session must be UTF-8.

Interaction If the data representation is not compatible with the current session, then SAS/CONNECT is also required. See [“Using a SAS/CONNECT Server” on page 1351](#).

MOVE

deletes the original members from the source library. If a member already exists in the target library, the member is not deleted from the source library and a message is sent to the SAS log. If a catalog already exists in the target library, then no catalogs are deleted from the source library and a message is sent to the log. If a data set has referential integrity constraints, the data set is not deleted from the source library and an error is sent to the log.

Interaction The CVP engine is read-only, so it does not support the MOVE option.

Tip Use the MOVE option only if your system is space-constrained. It is preferable to verify the migration of the member before it is deleted.

SLIBREF=libref

specifies a libref that is assigned through a SAS/CONNECT server.

To determine whether you must specify a SAS/CONNECT libref in the IN= option or in the SLIBREF= option, see [“Using a SAS/CONNECT Server” on page 1351](#).

Restriction If the catalogs were created in SAS 6 or SAS 8, SLIBREF= must be assigned through a SAS 8 server.

Requirement The server must be running on the same type of operating environment as the source library. For example, if the source session is running under Linux, then the server must be running under Linux.

Interaction If CATALOG is the only member type in the library and you are using the SLIBREF= option, then omit the IN= option.

KEEPNODUPKEY

specifies to retain the NODUPKEY sort order. See [“Migrating a SAS Data Set with NODUPKEY Sort Indicator” on page 1353](#).

Usage: MIGRATE Procedure

Migrating with the CVP Engine to Avoid Truncation

When you process a file in an encoding that uses more bytes to represent the characters, truncation could occur if the variable length does not accommodate the larger character size. For example, a character might be represented in wlatin1 encoding as one byte but in UTF-8 as two bytes. Truncation can also cause “garbage” characters or incorrect characters in output. In addition, some Microsoft

Word characters such as smart quotation marks require more than one byte in UTF-8. When these characters transcode from a single-byte encoding, truncation might occur if the data buffer is not large enough.

PROC MIGRATE supports using the CVP engine to expand the lengths of character variables in SAS data sets. Specify the CVP engine in the LIBNAME statement for the IN= library.

By default, the CVP engine automatically chooses a multiplier value. The automatic value is usually sufficient to avoid truncation. You can also use the CVPMULTIPLIER= option to specify it yourself. For more information, read about avoiding character data truncation in [SAS National Language Support \(NLS\): Reference Guide](#).

Here are the restrictions for using the CVP engine with PROC MIGRATE:

- SAS catalogs are migrated but the CVP engine does not transcode the catalog entries or expand the defined lengths in formats. For information about the methods of migrating catalogs, see [“Catalogs” on page 1346](#).
- The engine does not support data that was created under the V6 or TAPE engines.
- The engine is read-only, so the MOVE option is not supported.
- The engine cannot be specified in a library assignment that uses SAS/CONNECT. For details about using CVP and SAS/CONNECT together in PROC MIGRATE, see [“When to Specify the Server in the IN= Option or in the SLIBREF= Option” on page 1352](#).

For more information about the CVP engine, see [“Avoiding Character Data Truncation By Using the CVP Engine” in SAS National Language Support \(NLS\): Reference Guide](#).

Using a SAS/CONNECT Server

When to Use a SAS/CONNECT Server

Here are two reasons to use a SAS/CONNECT server:

- You can use a server to migrate across computers when you do not have direct access.
- You are required to use a server if the following two conditions are both met:
 - The source library contains catalogs.
 - The source library was created under a data representation that is not compatible with the target session. The SAS Viya platform on Linux for x64 has a SAS data representation of LINUX_X86_64. The following three data representations are compatible for SAS catalogs: Linux for x64

(LINUX_X86_64), Linux on the Power Architecture (LINUX_POWER_64), and Solaris for x64 (SOLARIS_X86_64).

For information about the methods of migrating catalogs, see [“Catalogs” on page 1346](#).

When to Specify the Server in the IN= Option or in the SLIBREF= Option

- If you are using the CVP engine, specify the SAS/CONNECT server libref in the SLIBREF= option. Specify the CVP libref in the IN= option. You cannot specify CVP in a SAS/CONNECT library assignment, so the two options are both needed. See [“Example 3: Migrate with the CVP Engine to Avoid Truncation” on page 1358](#).
- If you are not using the CVP engine, specify the SAS/CONNECT server libref in the IN= option. This syntax is supported when the source library contains catalogs that were created in a SAS Viya offering, or in SAS 9.1.3 or later. See [“Example 2: Migrate from SAS®9 by Using SAS/CONNECT” on page 1356](#).
- Specify the SAS/CONNECT server libref in the IN= option when the source library does not contain catalogs. Some customers choose to use SAS/CONNECT instead of using direct NFS access.
- Specify the SAS/CONNECT server libref in the SLIBREF= option when the source library contains catalogs that were created prior to SAS 9.1.3.

Requirements for the SAS/CONNECT Server

The SAS/CONNECT server must be running on the same type of operating environment and encoding as the source library. For example, if the source session is running under Linux with Latin1 encoding, then the server must be running under Linux with Latin1 encoding.

The server that you specify in the IN= option must be SAS 9.1.3 or later.

If you cannot meet these requirements, use the alternate method described in [“Additional Steps for Unsupported Catalogs” on page 1354](#).

Migrating a Data Set with Audit Trails, Generations, Indexes, or Integrity Constraints

In all cases, the data set migrates first, and then the audit trails, generations, index, or integrity constraints are applied. Errors are handled in the following ways:

- If an error occurs while an index is created for a migrated data set, the data set might migrate without the index, or processing might stop. A message is written to the SAS log. If an index fails to migrate, resolve the error and re-create the index in the target library.

If an index is missing from the source library, then depending on the environment and system options, SAS might try to repair the data set during migration by re-creating the index. If the data set is incompatible with the session encoding or data representation, re-creating the index produces an error. To resolve the error, re-create the index in the source library using the original operating system or move the index from its original location, and submit the PROC MIGRATE again. The error can occur when a customer has moved a data set using the operating system and failed to include an index in the move.

- If an error occurs while integrity constraints are applied to a migrated data set, or while an audit trail or generations are migrated, the data set is removed from the target library. A note is written to the SAS log. If the MOVE option is specified, it does not delete the data set from the source library.
- For a data set with referential integrity constraints, the MOVE option does not delete any members in the source library, even when the migration is successful. You must remove referential integrity constraints before the member can be deleted. An error message is written to the SAS log.

Migrating a SAS Data Set with NODUPKEY Sort Indicator

When you migrate a SAS data set that was sorted with the NODUPKEY option, you can either use the default behavior or specify the KEEPNOUDUPKEY option.

Under the default behavior (without the KEEPNOUDUPKEY option), the SAS data set retains its sort indicator in the target library. However, the NODUPKEY attribute is removed, and a warning message is written to the SAS log. This is the default behavior because SAS data sets that were sorted with the NODUPKEY option in previous releases might retain observations with duplicate keys. You can re-sort the migrated SAS data set by the key variables in PROC SORT so that observations with duplicate keys are eliminated and the correct attributes are recorded.

If you specify the KEEPNOUDUPKEY option, you must examine your migrated data to determine whether observations with duplicate keys exist. If so, you must re-sort the SAS data set to have the data and NODUPKEY sort attribute match.

Additional Steps for Unsupported Catalogs

When to Use Additional Steps

PROC MIGRATE is not supported for migrating catalogs under the following circumstances:

- To migrate catalogs from an incompatible data representation, you must use SAS/CONNECT software with PROC MIGRATE. (For more information, see [“Using a SAS/CONNECT Server” on page 1351](#).) If you do not have access to SAS/CONNECT software, then use the instructions at [“Using a Transport File for Catalogs from an Incompatible Data Representation” on page 1354](#).
- PROC MIGRATE does not migrate stored compiled macros, which are stored in a catalog. Instead, move the source code to the target, where you can compile and store it. As a best practice, always specify the SOURCE option in the %MACRO statement to save the source code for macro definitions. You could also save the source code in a separate file (for example, a text file).

You can use additional steps for just the catalogs, if unsupported catalogs are the only issue. By using PROC MIGRATE for other members of the library, you can retain those members' attributes.

Using a Transport File for Catalogs from an Incompatible Data Representation

- 1 In the source session, create a transport file with PROC CPORT. Specify MEMTYPE=CATALOG if you want to include catalogs only. (See [Chapter 14, “CPORT Procedure,” on page 341](#).)
- 2 Move the transport file to the target environment. Do not use RLS (a feature of SAS/CONNECT software) to move catalogs, or you will encounter errors. You must use binary FTP, the DOWNLOAD procedure, Network File System (NFS), or another method of directly accessing files.
- 3 In the target session, use PROC CIMPORT to import the transport file. (See [Chapter 10, “CIMPORT Procedure,” on page 179](#).)

Examples: MIGRATE Procedure

Example 1: Migrate with Direct Access

Features: PROC MIGRATE statement options
IN=
OUT=

Details

In this example, the following is demonstrated:

- The target environment has direct access (for example, by using NFS) to the source library.
- If catalogs are present, then additional options might be required. Those options are not used in this example.

Program

```
libname source 'source-library-pathname';  
libname target 'target-library-pathname';  
  
proc migrate in=source out=target options;  
run;
```

Program Description

Assign the source and target libraries. Substitute your library paths. The locations must be accessible by the SAS Compute Server, which is the programming environment for the SAS Viya platform.

```
libname source 'source-library-pathname';  
libname target 'target-library-pathname';
```

Submit PROC MIGRATE, and include any options you want.

```
proc migrate in=source out=target options;
run;
```

Log Messages

The following SAS log messages indicate that CEDA is invoked. Normally CEDA does not support indexes. To avoid that restriction, PROC MIGRATE redefines the indexes in the target session.

```
NOTE: Migrating SOURCE.INVENTORY to TARGET.INVENTORY (memtype=DATA).
NOTE: Data file SOURCE.INVENTORY.DATA is in a format that is native to another host,
or the
      file encoding does not match the session encoding. Cross Environment Data
Access will be
      used, which might require additional CPU resources and might reduce
performance.
NOTE: Simple index PRODTYPE has been defined.
NOTE: Simple index QUARTER has been defined.
NOTE: There were 23040 observations read from the data set SOURCE.INVENTORY.
NOTE: The data set TARGET.INVENTORY has 23040 observations and 11 variables.
```

The following log messages do not include a CEDA message. The data set in the source library has the same data representation and encoding as the target session.

```
NOTE: Migrating SOURCE.MYDATASETUTF8 to TARGET.MYDATASETUTF8 (memtype=DATA).
NOTE: There were 2 observations read from the data set SOURCE.MYDATASETUTF8.
NOTE: The data set TARGET.MYDATASETUTF8 has 2 observations and 4 variables.
```

The following log messages indicate that a catalog was not migrated, and processing stopped. However, the other members of the library are migrated.

```
ERROR: File SOURCE.FORMATS.CATALOG was created for a different operating system.
WARNING: No data is available.
NOTE: The SAS System stopped processing this step because of errors.
```

Example 2: Migrate from SAS®9 by Using SAS/CONNECT

Features:

PROC MIGRATE statement options

```
IN=
OUT=
```

Details

In this example, a spawner starts a session on the SAS/CONNECT server to access remote SAS data. For information about the reasons to use a server, see [“Using a SAS/CONNECT Server” on page 1351](#).

Program

```
options comamid=tcp;
%let myserver=host.name.com;
signon myserver.__1234 user=userid password='mypw';

libname source 'source-library-pathname' server=myserver.__1234;
libname target 'target-library-pathname';

proc migrate in=source out=target <options>;
run;
signoff myserver.__1234;
```

Program Description

Submit the following code to invoke a SAS/CONNECT spawner. The COMAMID= system option specifies TCP/IP as the communications access method. The myserver macro variable is assigned to the host name of the remote SAS/CONNECT server. The SIGNON statement references the myserver macro variable followed by the port number that the SAS/CONNECT spawner is listening on. If your port number or service name is defined in the macro variable, then omit it from the SIGNON.

```
options comamid=tcp;
%let myserver=host.name.com;
signon myserver.__1234 user=userid password='mypw';
```

Assign the source and target libraries. Substitute your library paths, and specify the myserver macro variable for SERVER= in the source library. Include the port number if it is specified in the SIGNON statement. The target library location must be accessible by the Compute Server.

```
libname source 'source-library-pathname' server=myserver.__1234;
libname target 'target-library-pathname';
```

Submit PROC MIGRATE, and include any options you want.

```
proc migrate in=source out=target <options>;
run;
signoff myserver.__1234;
```

Log Messages

The following SAS log messages indicate that a data set and a formats catalog are migrated. PROC MIGRATE calls the CPORT and CIMPORT procedures to migrate catalogs. Normally CEDA does not support catalogs, but using a SAS/CONNECT server avoids CEDA restrictions. The server must have the same data representation and encoding as the data in the source library.

```
NOTE: Migrating SOURCE.TEST to TARGET.TEST (memtype=DATA).  
NOTE: There were 3 observations read from the data set SOURCE.TEST.  
NOTE: The data set TARGET.TEST has 3 observations and 2 variables.  
NOTE: PROC CPORT begins to transport catalog SOURCE.FORMATS  
NOTE: Entry MYFMT.FORMAT has been transported.  
NOTE: Entry MYABC.FORMATC has been transported.
```

Example 3: Migrate with the CVP Engine to Avoid Truncation

Features: CVP engine
 PROC MIGRATE statement options
 IN=
 OUT=
 SLIBREF=

Note: See also [“Migrating with the CVP Engine to Avoid Truncation” on page 1350](#).

Details

The CVP engine prevents truncation by increasing the lengths of variables during migration. When you migrate to an encoding that uses more bytes to represent the characters, truncation might occur if a variable's length does not accommodate the larger character size.

In this example, the following is demonstrated:

- The SLIBREF= argument accesses the catalogs in the source library. The SLIBREF= argument specifies a SAS/CONNECT spawner that is compatible with the data representation of the catalogs. For information about the reasons to use a server, see [“Using a SAS/CONNECT Server” on page 1351](#).
- Although a formats catalog is migrated in this example, you must ensure that the defined lengths in formats are sufficient. The CVP engine does not transcode the catalog entries or expand the defined lengths in formats. To transcode and expand user-defined formats in a catalog, see [“Example 4: Migrate When a Catalog Is from a Different Encoding” on page 1359](#).

- The IN= argument accesses the rest of the supported file types in the source library. The IN= argument specifies a libref that is assigned to the CVP engine and accesses the files directly via a Network File System (NFS). The library assignment does not use the SAS/CONNECT spawner. The CVP engine cannot be specified in a library assignment that uses SAS/CONNECT. NFS is a standard protocol of UNIX operating environments. See the documentation for NFS and for your operating environment.
- For more information, see [“Migrating with the CVP Engine to Avoid Truncation” on page 1350](#).

Submit the following code to invoke a SAS/CONNECT spawner. The COMAMID= system option specifies TCP/IP as the communications access method. The myserver macro variable is assigned to the host name of the remote SAS/CONNECT server. The SIGNON statement references the myserver macro variable followed by the port number that the SAS/CONNECT spawner is listening on. If your port number or service name is defined in the macro variable, then omit it from the SIGNON.

```
options comamid=tcp;
%let myserver=host.name.com;
signon myserver.__1234 user=userid password='mypw';
```

Assign the svrlib, source, and target libraries. Substitute your library paths, and specify the myserver macro variable for SERVER= in the svrlib library. Include the port number if it is specified in the SIGNON statement. The source library uses the CVP engine and accesses the same location as the svrlib library. The target library location must be accessible by the Compute Server.

```
libname svrlib 'source-library-pathname' server=myserver.__1234;
libname source cvp 'source-library-pathname';
libname target 'target-library-pathname';
```

Submit PROC MIGRATE, and include any options you want.

```
proc migrate in=source slibref=svrlib out=target <options>;
run;
signoff myserver.__1234;
```

Example 4: Migrate When a Catalog Is from a Different Encoding

Features:	CVP engine
	PROC MIGRATE statement options
	IN=
	OUT=
	ENCODING=

Details

The `ENCODING=` option migrates user-defined formats catalogs that were created under a different encoding than the target session. Use the `ENCODING=` option to specify the encoding of the session where the catalogs were created. For more information, see [“ENCODING=encoding-value” on page 1349](#) and [“Catalogs” on page 1346](#).

Note that this example code does not demonstrate the use of `SAS/CONNECT`. In this example, the catalogs must have been created under a data representation is compatible with the current session.

Create a sample data set and user-defined format. When the format is later used in the `PRINT` procedure, it causes the word `café` to print as `café` and the word `latté` to print as `espresso`. If the format is not properly migrated, the `é` character does not print. Submit this code in a `latin1` session.

```
/* Create an example library. Submit this code in a latin1 session. */
libname mylib 'library-pathname';
data mylib.test;
    input drink $;
    datalines;
cafe
latté
;
run;

proc format library=mylib;
    value $translate
        'cafe' = 'café'
        'latté' = 'espresso'
    ;
run;
```

Migrate the library. The `ENCODING=` option enables SAS to transcode the user-defined formats catalog from `latin1` to `utf8`. Submit this code in the target session, which has an encoding of `UTF-8`. The `CVP` engine is not necessary for this example. It is shown here for users who need to expand data values in their data sets.

```
/* Migrate the library. Submit this code in a UTF-8 session. */
libname mytest.cvp 'source-library-pathname';
libname outlib 'target-library-pathname';

proc migrate in=mytest out=outlib encoding='latin1';
run;
```

Demonstrate a successful migration. Submit the `FMTSEARCH=` option to specify the migrated library. Use the `FORMAT` statement in `PROC PRINT` to apply the format.

```
options fmtsearch=(outlib);
```

```
proc print data=outlib.test;  
    format drink $translate.;  
run;
```

Outlib.Test Data Set Showing the Formatted Data

Obs	drink
1	café
2	espresso

OPTIONS Procedure

Overview: OPTIONS Procedure	1363
What Does the OPTIONS Procedure Do?	1363
Syntax: OPTIONS Procedure	1364
PROC OPTIONS Statement	1364
Usage: OPTIONS Procedure	1370
Display a List of System Options	1370
Display Information about One or More Options	1371
Display Information about System Option Groups	1373
Display Restricted Options	1376
Display Options That Can Be Saved	1377
Identify Where an Option Was Set	1378
Results: OPTIONS Procedure	1378
View PROC OPTIONS Output in the SAS Log	1378
Examples: OPTIONS Procedure	1379
Example 1: Producing the Short Form of the Options Listing	1379
Example 2: Displaying the Setting of a Single Option	1380
Example 3: Displaying Expanded Path Environment Variables	1381
Example 4: List the Options That Can Be Specified by the INSERT and APPEND Options	1382

Overview: OPTIONS Procedure

What Does the OPTIONS Procedure Do?

The OPTIONS procedure lists the current settings of SAS system options in the SAS log.

SAS system options control how SAS formats output, handles files, processes data sets, interacts with the operating environment, and does other tasks that are not specific to a single SAS program or data set. You use the OPTIONS procedure to obtain information about an option or a group of options. Here is some of the information that the OPTIONS procedure provides:

- the current value of an option and how it was set
- a description of an option
- valid syntax for the option, valid option values, and the range of values
- where you can set the system option
- if the option can be restricted by your site administrator
- if the option has been restricted
- system options that belong to a system option group
- system options that are specific for an operating environment
- if an option value has been modified by the INSERT or APPEND system options
- system options that can be saved by the OPTSAVE procedure.

For additional information about SAS system options, see [SAS System Options: Reference](#).

Syntax: OPTIONS Procedure

Note: SAS system options are documented in several publications. You can access all system options from “[SAS System Options Documented in Other SAS Publications](#)” in [SAS System Options: Reference](#).

PROC OPTIONS <options>;

Statement	Task	Example
PROC OPTIONS	List the current system option settings to the SAS Log	Ex. 1 , Ex. 2 , Ex. 3 , Ex. 4

PROC OPTIONS Statement

Lists the current settings of SAS system options in the SAS log.

Examples: “[Example 1: Producing the Short Form of the Options Listing](#)” on page 1379
 “[Example 2: Displaying the Setting of a Single Option](#)” on page 1380
 “[Example 3: Displaying Expanded Path Environment Variables](#)” on page 1381

“Example 4: List the Options That Can Be Specified by the INSERT and APPEND Options” on page 1382

Syntax

PROC OPTIONS <*options*>;

Summary of Optional Arguments

LISTGROUPS

lists the system option groups as well as a description of each group.

NOEXPAND

Returns the path for the environment value.

Choose the format of the listing

DEFINE

displays the short description of the option, the option group, and the option type.

EXPAND

Returns the value of the specified environment variable.

HEXVALUE

displays system option character values as hexadecimal values.

LOGNUMBERFORMAT

includes or omits punctuation in the display of numeric system options values

LONG

lists each system option on a separate line with a description.

NOLOGNUMBERFORMAT

omits punctuation in the display of numeric system options values

SHORT

specifies to display a compressed listing of options without descriptions.

VALUE

displays the option's value and scope, as well as how the value was set.

Restrict the number of options displayed

GROUP=*group-name* | **GROUP=**(*group-name-1 ... group-name-n*)

displays the options in one or more groups specified by *group-name*.

HOST

displays only host options.

LISTINSERTAPPEND

lists the system options whose value can be modified by the INSERT and APPEND system options.

LISTOPTSAVE

lists the system options that can be saved with PROC OPTSAVE .

LISTRESTRICT

lists the system options that can be restricted by your site administrator.

NOHOST

displays only portable options.

OPTION=*option-name* | OPTION=(*option-name-1...option-name-n*)

displays information about one or more system options.

RESTRICT

displays system options that the site administrator has restricted from being updated.

Optional Arguments

DEFINE

displays the short description of the option, the option group, and the option type. SAS displays information about when the option can be set, whether an option can be restricted, the valid values for the option, and whether the OPTSAVE procedure will save the option.

Interaction This option is ignored when SHORT is specified.

Example [“Example 2: Displaying the Setting of a Single Option” on page 1380](#)

EXPAND

Returns the value of the specified environment variable. When displaying a character option, EXPAND replaces an environment variable in the option value with the value of the environment variable. EXPAND is ignored if the option is a Boolean option, such as CENTER or NOCENTER, or if the value of the option is numeric.

When displaying a path, NOEXPAND displays the path using environment variable(s) and not the value of the environment variable(s).

Default NOEXPAND

Restriction Variable expansion is valid only in the Windows and UNIX operating environments.

Tip By default, some option values are displayed with expanded variables. Other options require the EXPAND option in the PROC OPTIONS statement. Use the DEFINE option in the PROC OPTIONS statement to determine whether an option value expands variables by default or if the EXPAND option is required. If the output from PROC OPTIONS DEFINE shows the following information, you must use the EXPAND option to expand variable values:

Expansion: Environment variables, within the option value,
are not expanded

See [“NOEXPAND” on page 1368](#).

Example [“Example 3: Displaying Expanded Path Environment Variables” on page 1381](#)

GROUP=*group-name* | GROUP=(*group-name-1 ... group-name-n*)

displays the options in one or more groups specified by *group-name*.

Requirement When you specify more than one group, enclose the group names in parenthesis and separate the group names by a space.

See [“Display Information about System Option Groups” on page 1373](#)

HEXVALUE

displays system option character values as hexadecimal values.

HOST

displays only host options. HOST displays only host options.

NOHOST displays only portable options.

See [“NOHOST” on page 1369](#).

LISTINSERTAPPEND

lists the system options whose value can be modified by the INSERT and APPEND system options. The INSERT option specifies a value that is inserted as the first value of a system option value list. The APPEND option specifies a value that is appended as the last value of a system option value list. Use the LISTINERTAPPEND option to display which system options can have values inserted at the beginning or appended at the end of their value lists.

See [“INSERT= System Option” in SAS System Options: Reference](#)

[“APPEND= System Option” in SAS System Options: Reference](#)

Example [“Example 4: List the Options That Can Be Specified by the INSERT and APPEND Options” on page 1382](#)

LISTGROUPS

lists the system option groups as well as a description of each group.

See [“Display Information about System Option Groups” on page 1373](#)

LISTOPTSAVE

lists the system options that can be saved with PROC OPTSAVE .

LISTRESTRICT

lists the system options that can be restricted by your site administrator.

See [“RESTRICT” on page 1369](#) option to list options that have been restricted by the site administrator

LOGNUMBERFORMAT

includes or omits punctuation in the display of numeric system options values. LOGNUMBERFORMAT displays numeric system option values using locale-specific punctuation.

NOLOGNUMBERFORMAT displays numeric system option values without using punctuation, such as a comma or a period.

Default NOLOGNUMBERFORMAT

See [“NOLOGNUMBERFORMAT” on page 1369](#).

Example [“Example 2: Displaying the Setting of a Single Option” on page 1380](#)

LONG

lists each system option on a separate line with a description. This is the default. Alternatively, you can create a compressed listing without descriptions.

See [“SHORT” on page 1369](#) option to produce a compressed listing without descriptions

Example [“Example 1: Producing the Short Form of the Options Listing” on page 1379](#)

OPTION=option-name | OPTION=(option-name-1...option-name-n)

displays a short description and the value (if any) of the option specified by *option-name*. DEFINE and VALUE options provide additional information about the option.

option-name

specifies the option to use as input to the procedure.

Requirement If a SAS system option uses an equal sign, such as PAGESIZE=, do not include the equal sign when specifying the option to OPTION=.

Example [“Example 2: Displaying the Setting of a Single Option” on page 1380](#)

NOEXPAND

Returns the path for the environment value. When displaying a character option, EXPAND replaces an environment variable in the option value with the value of the environment variable. EXPAND is ignored if the option is a Boolean option, such as CENTER or NOCENTER, or if the value of the option is numeric.

When displaying a path, NOEXPAND displays the path using environment variable(s) and not the value of the environment variable(s).

Default NOEXPAND

Restriction Variable expansion is valid only in the Windows and UNIX operating environments.

Tip By default, some option values are displayed with expanded variables. Other options require the EXPAND option in the PROC OPTIONS statement. Use the DEFINE option in the PROC OPTIONS statement to determine whether an option value expands variables by default or if the EXPAND option is required. If the output from PROC OPTIONS DEFINE shows the following information, you must use the EXPAND option to expand variable values:

Expansion: Environment variables, within the option value,
are not expanded

See [“EXPAND” on page 1366](#)

Example [“Example 3: Displaying Expanded Path Environment Variables” on page 1381](#)

NOHOST

displays only portable options. HOST displays only host options.

NOHOST displays only portable options.

Aliases PORTABLE

PORT

See [“HOST” on page 1367](#)

NOLOGNUMBERFORMAT

omits punctuation in the display of numeric system options values
LOGNUMBERFORMAT displays numeric system option values using locale-specific punctuation.

NOLOGNUMBERFORMAT displays numeric system option values without using punctuation, such as a comma or a period.

Default NOLOGNUMBERFORMAT

See [“LOGNUMBERFORMAT” on page 1367](#)

Example [“Example 2: Displaying the Setting of a Single Option” on page 1380](#)

RESTRICT

displays the system options that have been set by your site administrator in a restricted options configuration file. These options cannot be changed by the user. For each option that is restricted, the RESTRICT option displays the option's value, scope, and how it was set.

If your site administrator has not restricted any options, then the following message appears in the SAS log:

```
Your Site Administrator has not restricted any SAS options.
```

See [“LISTRESTRICT” on page 1367](#) option to list options that can be restricted by the site administrator

SHORT

specifies to display a compressed listing of options without descriptions.

See [“LONG” on page 1368](#) option to create a listing with descriptions of the options.

VALUE

displays the option's value and scope, as well as how the value was set. If the value was set using a configuration file, the SAS log displays the name of the configuration file. If the option was set using the INSERT or APPEND system options, the SAS log displays the value that was inserted or appended.

Interactions This option has no effect when SHORT is specified.

When the option is in the Threaded Kernel (TK) system options group, the value of **How option value set** is displayed as **Internal**

Note SAS options that are passwords, such as EMAILPW, return the value xxxxxxxx and not the actual password.

Example [“Example 2: Displaying the Setting of a Single Option” on page 1380](#)

Usage: OPTIONS Procedure

Display a List of System Options

The log that results from running PROC OPTIONS can show the system options for the options that are available for all operating environment and those that are specific to a single operating environment. Options that are available for all operating environments are referred to as portable options. Options that are specific to a single operating environment are referred to as host options.

The following example shows a partial log that displays the settings of portable options.

```
proc options;  
run;
```


Example Code 37.1 The SAS Log Showing a Partial Listing of SAS System Options

Portable Options:

```

NOACCESSIBLECHECK Do not detect and log ODS output that is not accessible.
NOACCESSIBLEGRAPH Do not create accessible ODS graphics by default.
NOACCESSIBLEPDF Do not create accessible PDF files by default.
NOACCESSIBLETABLE Do not create accessible tables for enabled procedures, by default.

ANIMATION=STOP Specifies whether to start or stop animation.
ANIMDURATION=MIN Specifies the number of seconds that each animation frame displays.
ANIMLOOP=YES Specifies the number of iterations that animated images repeat.
ANIMOVERLAY Specifies that animation frames are overlaid in order to view all frames.
APPEND= Specifies an option=value pair to insert the value at the end of the existing
option value.
APPLETLOC=site-specific-path Specifies the location of Java applets, which is typically a URL.
ARMAGENT= Specifies an ARM agent (which is an executable module or keyword, such as
LOG4SAS) that contains a specific implementation of the ARM API.
ARMLOC=ARMLOG.LOG Specifies the location of the ARM log.
ARMSUBSYS=(ARM_NONE) Specifies the SAS ARM subsystems to enable or disable.
AUTOCORRECT Automatically corrects misspelled procedure names and keywords, and global
statement names.

```

The log displays both portable and host options when you submit `proc options;`

To view only host options, use this version of the OPTIONS procedure:

```

proc options host;
run;

```

Example Code 37.2 The SAS Log Showing a Partial List of Host Options

Host Options:

```

ALIGNSASIOFILES Aligns SAS files on a page boundary for improved performance.
ALTLOG= Specifies the location for a copy of the SAS log when SAS is running in batch
mode.
ALTPRINT= Specifies the location for a copy of the SAS procedure output when SAS is
running in batch mode.
AUTHPROVIDERDOMAIN= Specifies the authentication provider that is associated with a domain.
AUTHSERVER= Specifies the domain server that finds and authenticates secure server logins.

```

Display Information about One or More Options

To view the setting of one or more particular options, you can use the `OPTION=` and `DEFINE` options in the `PROC OPTIONS` statement. The following example shows a log that `PROC OPTIONS` produces for a single SAS system option.

```

proc options option=errorcheck define;
run;

```

Example Code 37.3 *The Setting of a Single SAS System Option*

```

ERRORCHECK=NORMAL
Option Definition Information for SAS Option ERRORCHECK
  Group= ERRORHANDLING
  Group Description: Error messages and error conditions settings
  Description: Specifies whether SAS enters syntax-check mode when errors are found in the
                  LIBNAME, FILENAME, %INCLUDE, and LOCK statements.
  Type: The option value is of type CHARACTER
        Maximum Number of Characters: 10
        Casing: The option value is retained uppercased
        Quotes: If present during "set", start and end quotes are removed
        Parentheses: The option value does not require enclosure within parentheses. If
                     present, the parentheses are retained.
        Expansion: Environment variables, within the option value, are not expanded
        Number of valid values: 2
                  Valid value: NORMAL
                  Valid value: STRICT
  When Can Set: Startup or anytime during the SAS Session
  Restricted: Your Site Administrator can restrict modification of this option
  Optsave: PROC Optsave or command Dmoptsave will save this option

```

To view the settings for more than one option, enclose the options in parentheses and separate the options with a space:

```

proc options option=(pdfsecurity pdfpassword) define;
run;

```

Example Code 37.4 *The Settings of Two SAS System Options*

```

PDFSECURITY=NONE
Option Definition Information for SAS Option PDFSECURITY
  Group= PDF
  Group Description: PDF settings
  Group= SECURITY
  Group Description: Security settings
  Description: Specifies the level of encryption to use for PDF documents.
  Type: The option value is of type CHARACTER
        Maximum Number of Characters: 4
        Casing: The option value is retained uppercased
        Quotes: If present during "set", start and end quotes are removed
        Parentheses: The option value does not require enclosure within parentheses. If
                     present, the parentheses are retained.
        Expansion: Environment variables, within the option value, are not expanded
        Number of valid values: 3
                  Valid value: HIGH
                  Valid value: LOW
                  Valid value: NONE
  When Can Set: Startup or anytime during the SAS Session
  Restricted: Your Site Administrator can restrict modification of this option
  Optsave: PROC Optsave or command Dmoptsave will save this option

```

```

PDFPASSWORD=xxxxxxx
Option Definition Information for SAS Option PDFPASSWORD
  Group= PDF
  Group Description: PDF settings
  Group= SECURITY
  Group Description: Security settings
  Description: Specifies the password to use to open a PDF document and the password used by a
                PDF document owner.
  Type: The option value is of type CHARACTER
        Maximum Number of Characters: 2048
        Casing: The option value is retained with original casing
        Quotes: If present during "set", start and end quotes are removed
        Parentheses: The option value must be enclosed within parentheses. The parentheses are
                    retained.
        Expansion: Environment variables, within the option value, are not expanded
        Number of valid values: 2
          Valid value: OPEN
          Valid value: OWNER
  When Can Set: Startup or anytime during the SAS Session
  Restricted: Your Site Administrator cannot restrict modification of this option
  Optsave: PROC Optsave or command Dmoptsave will not save this option

```

Display Information about System Option Groups

Each SAS system option belongs to one or more groups, which are based on functionality, such as error handling or sorting. You can display a list of system-option groups and the system options that belong to one or more of the groups.

Use the LISTGROUPS option to display a list of system-option groups.

```

proc options listgroups;
run;

```

Example Code 37.5 List of SAS System Option Groups

Option Groups	
GROUP=ANIMATION	Animation
GROUP=CAS	CAS Options
GROUP=CODEGEN	Code generation
GROUP=COMMUNICATIONS	Networking and encryption
GROUP=DATACOM	Datacom
GROUP=DATAQUALITY	Data Quality
GROUP=DB2	DB2
GROUP=EMAIL	E-mail
GROUP=ENVDISPLAY	Display
GROUP=ENVFILES	Files
GROUP=ERRORHANDLING	Error handling
GROUP=EXECMODES	Initialization and operation
GROUP=EXTFILES	External files
GROUP=GRAPHICS	Driver settings
GROUP=HELP	Help
GROUP=IDMS	IDMS
GROUP=IMS	IMS
GROUP=INPUTCONTROL	Data Processing
GROUP=INSTALL	Installation
GROUP=ISPF	ISPF
GROUP=LANGUAGECONTROL	Language control
GROUP=LISTCONTROL	Procedure output
GROUP=LOGCONTROL	SAS log
GROUP=LOG_LISTCONTROL	SAS log and procedure output
GROUP=MACRO	SAS macro
GROUP=MEMORY	Memory
GROUP=META	Metadata
GROUP=ODSPRINT	ODS Printing
GROUP=PDF	PDF
GROUP=PERFORMANCE	Performance
GROUP=REXX	REXX
GROUP=SASFILES	SAS Files
GROUP=SECURITY	Security
GROUP=SMF	SMF
GROUP=SORT	Procedure options
GROUP=SQL	SQL
GROUP=SVG	SVG
GROUP=TK	TK

Use the GROUP= option to display system options that belong to a particular group. You can specify one or more groups.

```
proc options group=(svg graphics);
run;
```

Example Code 37.6 Sample Output Using the GROUP= Option

```

Group=SVG
  ANIMATION=STOP      Specifies whether to start or stop animation.
  ANIMDURATION=MIN    Specifies the number of seconds that each animation frame displays.
  ANIMLOOP=YES        Specifies the number of iterations that animated images repeat.
  ANIMOVERLAY         Specifies that animation frames are overlaid in order to view all frames.
  SVGAUTOPLAY         Starts animation when the page is loaded in the browser.
  NOSVGCONTROLBUTTONS
                        Does not display the paging control buttons and an index in a multipage SVG
                        document.
  SVGFADEIN=0         Specifies the number of seconds for the fade-in effect for a graph.
  SVGFADEMODE=OVERLAP
                        Specifies whether to use sequential frames or to overlap frames for the
                        fade-in effect of a graph.
  SVGFADEOUT=0        Specifies the number of seconds for a graph to fade out of view.
  SVGHEIGHT=          Specifies the height of the viewport. Specifies the value of the height
                        attribute of the outermost SVG element.
  NOSVGMAGNIFYBUTTON
                        Disables the SVG magnifier tool.
  SVGPRESERVEASPECTRATIO=
                        Specifies whether to force uniform scaling of SVG output. Specifies the
                        preserveAspectRatio attribute on the outermost SVG element.
  SVGTITLE=           Specifies the text in the title bar of the SVG output. Specifies the value of
                        the TITLE element in the SVG file.
  SVGVIEWBOX=         Specifies the coordinates, width, and height that are used to set the viewBox
                        attribute on the outermost SVG element.
  SVGWIDTH=           Specifies the width of the viewport. Specifies the value of the width
                        attribute of the outermost SVG element.
  SVGX=               Specifies the x-axis coordinate of one corner of the rectangular region for
                        an embedded SVG element. Specifies the x attribute in the outermost SVG
                        element.
  SVGY=               Specifies the y-axis coordinate of one corner of the rectangular region for
                        an embedded SVG element. Specifies the y attribute in the outermost SVG
                        element.

Group=GRAPHICS
  DEVICE=             Specifies the device driver to which SAS/GRAPH sends procedure output.
  GSTYLE              Uses ODS styles to generate graphs that are stored as GRSEG catalog entries.
  GWINDOW             Displays SAS/GRAPH output in the GRAPH window.
  MAPS= ("!sasroot/path-to-maps")
                        Specifies the location of SAS/GRAPH map data sets.
  MAPSGFK= (          "!sasroot/path-to-maps" )
                        Specifies the location of GfK maps.
  MAPSSAS= (          "!sasroot/path-to-maps" )
                        Specifies the location of SAS map data sets.

```

You can use the following group names as values for the GROUP= option to list the system options in a group:

ANIMATION	GRAPHICS	META
CAS	HELP	ODSPRINT
COMMUNICATIONS	INPUTCONTROL	PDF
DATAQUALITY	INSTALL	PERFORMANCE
EMAIL	LANGUAGECONTROL	SASFILES
ENVDISPLAY	LISTCONTROL	SECURITY
ENVFILES	LOGCONTROL	SORT
ERRORHANDLING	LOG_LISTCONTROL	SQL
EXECMODES	MACRO	SVG

EXTFILES	MEMORY	TK
----------	--------	----

You can use the following groups to list operating environment–specific values that might be available when you use the GROUP= option with PROC OPTIONS.

ADABAS	IDMS	REXX
CODEGEN	IMS	SMF
DATAACOM	ISPF	
DB2	ORACLE	

Operating Environment Information: Refer to the SAS documentation for your operating environment for more information about these host-specific options.

Display Restricted Options

Your site administrator can restrict some system options so that your SAS session conforms to options that are set for your site. Restricted options can be modified only by your site administrator. The OPTIONS procedure provides two options that display information about restricted options. The RESTRICT option lists the system options that your site administrator has restricted. The LISTRESTRICT option lists the options that can be restricted by your site administrator. For a listing of options that cannot be restricted, see [“Restricted Options” in SAS System Options: Reference](#).

The following SAS logs shows the output when the RESTRICT option is specified and partial output when the LISTRESTRICT option is specified.

```
proc options restrict;
run;
```

Example Code 37.7 A List of Options That Have Been Restricted by the Site Administrator

```
Option Value Information For SAS Option CMPOPT
Option Value: (NOEXTRAMATH NOMISSCHECK NOPRECISE NOGUARDCHECK
NOGENSYMNAMES NOFUNCDIFFERENCING)
Option Scope: SAS Session
How option value set: Site Administrator Restricted
```

```
proc options listrestrict;
run;
```

Example Code 37.8 A Partial Log That Lists Options That Can Be Restricted

Your Site Administrator can restrict the ability to modify the following Portable Options:

ACCESSIBLECHECK	Detect and log ODS output that is not accessible.
ACCESSIBLEGRAPH	Create accessible ODS graphics by default.
ACCESSIBLEPDF	Create accessible PDF files by default.
ACCESSIBLETABLE	Create accessible tables for enabled procedures, by default.
ANIMATION	Specifies whether to start or stop animation.
ANIMDURATION	Specifies the number of seconds that each animation frame displays.
ANIMLOOP	Specifies the number of iterations that animated images repeat.
ANIMOVERLAY	Specifies that animation frames are overlaid in order to view all frames.
APPLETLOC	Specifies the location of Java applets, which is typically a URL.
ARMAGENT	Specifies an ARM agent (which is an executable module or keyword, such as LOG4SAS) that contains a specific implementation of the ARM API.
ARMLLOC	Specifies the location of the ARM log.
ARMSUBSYS	Specifies the SAS ARM subsystems to enable or disable.
AUTOCORRECT	Automatically corrects misspelled procedure names and keywords, and global statement names.
AUTOSAVELOC	Specifies the location of the Program Editor auto-saved file.

Display Options That Can Be Saved

Many system options can be saved by using PROC OPTSAVE. The options can later be restored by using PROC OPTLOAD. You can list the system options that can be saved and later restored by using the LISTOPTSAVE option on PROC OPTIONS.

Note: PROC OPTSAVE does not save these types of system options:

- SAS invocation options
- SAS environment variable options
- SAS system options that include passwords
- SAS system options that are read-only and cannot be set

The following SAS log shows a partial list of the options that can be saved by using PROC OPTSAVE:

```
proc options listoptsave;
run;
```

Example Code 37.9 A Partial List of System Options That Can Be Saved

Core options that can be saved with OPTSAVE

ACCESSIBLECHECK	Detect and log ODS output that is not accessible.
ACCESSIBLEGRAPH	Create accessible ODS graphics by default.
ACCESSIBLEPDF	Create accessible PDF files by default.
ACCESSIBLETABLE	Create accessible tables for enabled procedures, by default.
ANIMATION	Specifies whether to start or stop animation.
ANIMDURATION	Specifies the number of seconds that each animation frame displays.
ANIMLOOP	Specifies the number of iterations that animated images repeat.
ANIMOVERLAY	Specifies that animation frames are overlaid in order to view all frames.
APPLETLOC	Specifies the location of Java applets, which is typically a URL.
AUTOCORRECT	Automatically corrects misspelled procedure names and keywords, and global statement names.
AUTOSAVELOC	Specifies the location of the Program Editor auto-saved file.
AUTOSIGNON	Enables a SAS/CONNECT client to automatically submit the SIGNON command remotely with the RSUBMIT command.
BINDING	Specifies the binding edge type of duplexed printed output.
BOMFILE	Writes the byte order mark (BOM) prefix when a Unicode-encoded file is written to an external file.
BOTTOMMARGIN	Specifies the size of the margin at the bottom of a printed page.
BUFNO	Specifies the number of buffers for processing SAS data sets.
BUFSIZE	Specifies the size of a buffer page for output SAS data sets.

Identify Where an Option Was Set

You can specify the VALUE option in a PROC OPTIONS statement to identify the option's scope and the method that was used to set the option. If an option's value is set using multiple methods (for example, in an OPTIONS statement or in multiple configuration files), the VALUE option identifies which method determines the effective value. If the method is a configuration file, the VALUE option identifies the file path.

Use the following PROC OPTIONS statement to specify the VALUE option:

```
proc options option=option-name value;
```

For more information, see [Chapter 37, "OPTIONS Procedure,"](#) on page 1363.

Results: OPTIONS Procedure

View PROC OPTIONS Output in the SAS Log

SAS writes the options list to the SAS log.

SAS system options of the form *option* | *NOption* are listed as either *option* or *NOption*, depending on the current setting. They are always sorted by the positive form. For example, NOCAPS would be listed under the Cs.

The OPTIONS procedure displays passwords in the SAS log as eight Xs, regardless of the actual password length.

Operating Environment Information: PROC OPTIONS produces additional information that is specific to the environment under which you are running SAS. For more information about this and for descriptions of host-specific options, refer to the SAS documentation for your operating environment.

Examples: OPTIONS Procedure

Example 1: Producing the Short Form of the Options Listing

Features: PROC OPTIONS statement option
SHORT

Details

This example shows how to generate the short form of the listing of SAS system option settings. Compare this short form with the long form that is shown in [“Display a List of System Options” on page 1370](#).

Program

List all options and their settings. SHORT lists the SAS system options and their settings without any descriptions.

```
proc options short;  
run;
```

Log

Example Code 37.10 Partial Listing of the SHORT Option

Portable Options:

```
NOACCESSIBLECHECK NOACCESSIBLEGRAPH NOACCESSIBLEPDF NOACCESSIBLETABLE ANIMATION=STOP
ANIMDURATION=MIN ANIMLOOP=YES ANIMOVERLAY APPEND=
APPLETLOC=//bb03smb01/sasgen/dev/mva-v940m6/avdobj/jar/wx6no ARMAGENT= ARMLOC=ARMLOG.LOG
ARMSUBSYS=(ARM_NONE) AUTOCORRECT AUTOEXEC= AUTOSAVELOC= NOAUTOSIGNON BINDING=DEFAULT BOMFILE
BOTTOMMARGIN=0.000 IN BUFNO=1 BUFSIZE=0 BYERR BYLINE BYSORTED NOCAPS NOCARDIMAGE CASAUTHINFO=
CASDATALIMIT=100M CASHOST= CASLIB= CASNCHARMULTIPLIER=1 CASNWORKERS=ALL CASPORT=0 CASSESSOPTS=
CASTIMEOUT=60 CASUSER= CATCACHE=0 CBUFNO=0 CENTER CGOPTIMIZE=3 NOCHARCODE NOCHKPTCLEAN CLEANUP
NOCMDMAC CMPLIB= CMPMODEL=BOTH CMPOPT=(NOEXTRAMATH NOMISSCHECK NOPRECISE NOGUARDCHECK
NOGENSYMNAMES NOFUNCDIFFERENCING SHORTCIRCUIT NOPROFILE) NOCOLLATE COLOPHON= COLORPRINTING
```

Example 2: Displaying the Setting of a Single Option

Features:	PROC OPTIONS statement option
	OPTION=
	DEFINE
	LOGNUMBERFORMAT
	VALUE

Details

This example shows how to display the setting of a single SAS system option. The log shows the current setting of the SAS system option MAXMEMQUERY. The DEFINE and VALUE options display additional information. The LOGNUMBERFORMAT displays the value using commas.

Program

Specify the MAXMEMQUERY SAS system option. OPTION=MAXMEMQUERY displays option value information. DEFINE and VALUE display additional information. LOGNUMBERFORMAT specifies to format the value using commas.

```
proc options option=maxmemquery define value lognumberformat;
run;
```

Log

Example Code 37.11 Log Output from Specifying the MAXMEMQUERY Option

```
Option Value Information For SAS Option MAXMEMQUERY
  Value: 268,435,456
  Scope: SAS Session
  How option value set: Shipped Default
Option Definition Information for SAS Option MAXMEMQUERY
  Group= MEMORY
  Group Description: Memory settings
  Description: For certain procedures, specifies the maximum amount of memory that can be allocated
per request.
  Type: The option value is of type LONG
    Range of Values: The minimum is 0 and the maximum is 9007199254740992
    Valid Syntax(any casing): MIN|MAX|n|nK|nM|nG|nT|hexadecimal
  Numeric Format: Usage of LOGNUMBERFORMAT impacts the value format
  When Can Set: Startup or anytime during the SAS Session
  Restricted: Your Site Administrator can restrict modification of this option
  Optsave: PROC Optsave or command Dmoptsave will save this option
```

Example 3: Displaying Expanded Path Environment Variables

Features: PROC OPTIONS statement options

- OPTION=
- EXPAND
- NOEXPAND
- HOST

Details

This example shows the value of an environment variable when the path is displayed.

Program

Show the value of the environment variables: The EXPAND option causes the values of environment variables to display in place of the environment variable. The NOEXPAND option causes the environment variable to display. In this example, the environment variable is !sasroot

```
proc options option=msg expand;
```

```
run;
proc options option=msg noexpand;
run;
```

Log

Example Code 37.12 *Displaying an Expanded and Nonexpanded Pathname Using the OPTIONS Procedure*

```
82  proc options option=msg expand;
83  run;
    SAS (r) Proprietary Software Release
    MSG=/opt/sas/viya/home/SASFoundation/sasmsg
        Specifies the path to the library that contains SAS messages.
NOTE: PROCEDURE OPTIONS used (Total process time):
      real time          0.00 seconds
      cpu time           0.00 seconds

84  proc options option=msg noexpand;
85  run;
    SAS (r) Proprietary Software Release
    MSG=!SASROOT/sasmsg
        Specifies the path to the library that contains SAS messages.
```

Example 4: List the Options That Can Be Specified by the INSERT and APPEND Options

Features: PROC OPTIONS statement option
LISTINSERTAPPEND

Details

This example shows how to display the options that can be specified by the INSERT and APPEND system options.

Program

List all options that can be specified by the INSERT and APPEND options in the SAS Viya platform. The LISTINSERTAPPEND option provides a list and a description of these options.

```
proc options listinsertappend;
```

```
run;
```

Log

Example Code 37.13 *Displaying the Options That Can Be Specified by the INSERT and APPEND Options*

```
Core options that can utilize INSERT and APPEND
  AUTOEXEC          Specifies the location of the SAS AUTOEXEC files.
  CMPLIB            Specifies one or more SAS data sets that contain compiler subroutines to
include during compilation.
  FMTSEARCH          Specifies the order in which format catalogs are searched.
  MAPSGFK            Specifies the location of GfK maps.
  SASAUTOS           Specifies the location of one or more autocall libraries.
  SASHELP            Specifies the location of the Sashelp library.
  SASSCRIPT          Specifies one or more locations of SAS/CONNECT server sign-on script files.
Host options that can utilize INSERT and APPEND
  MSG                Specifies the path to the library that contains SAS messages.
  SET                Defines an environment variable.
```


OPTLOAD Procedure

Overview: OPTLOAD Procedure	1385
What Does the OPTLOAD Procedure Do?	1385
Syntax: OPTLOAD Procedure	1385
PROC OPTLOAD Statement	1386
Example: Load a Data Set of Saved System Options	1387

Overview: OPTLOAD Procedure

What Does the OPTLOAD Procedure Do?

The OPTLOAD procedure reads SAS system option settings that are stored in the SAS registry or a SAS data set and puts them into effect.

You can load SAS system option settings from a SAS data set or registry key by using the PROC OPTLOAD statement.

When an option is restricted by the site administrator, and the option value that is being set by PROC OPTLOAD differs from the option value that was established by the site administrator, SAS issues a warning message to the log.

Syntax: OPTLOAD Procedure

PROC OPTLOAD <options>;

Statement	Task	Example
PROC OPTLOAD	Use SAS system option settings that are stored in the SAS registry or in a SAS data set	Ex. 1

PROC OPTLOAD Statement

Loads saved setting of SAS system options that are stored in the SAS registry or in a SAS data set.

Syntax

```
PROC OPTLOAD <options>;
```

Summary of Optional Arguments

- DATA=libref.dataset**
Load SAS system option settings from an existing data set.
- KEY="SAS registry key"**
Load SAS system option settings from an existing registry key.

Optional Arguments

- DATA=libref.dataset**
specifies the library and data set name from where SAS system option settings are loaded. The SAS variable OPTNAME contains the character value of the SAS system option name, and the SAS variable OPTVALUE contains the character value of the SAS system option setting.
 - Default** If you omit the DATA= option and the KEY= option, the procedure will use the default SAS library and data set. The default library is where the current user profile resides. Unless you specify a library, the default library is SASUSER. If SASUSER is being used by another active SAS session, then the temporary WORK library is the default location from which the data set is loaded. The default data set name is MYOPTS.
 - Requirement** The SAS library and data set must exist.
- KEY="SAS registry key"**
specifies the location in the SAS registry of stored SAS system option settings. The registry is retained in SASUSER. If SASUSER is not available, then the temporary WORK library is used. For example, KEY="OPTIONS" loads system options from the OPTIONS registry key.

Requirements "SAS registry key" must be an existing SAS registry key.

You must use quotation marks around the "SAS registry key" name. Separate the names in a sequence of key names with a backslash (\). For example, KEY="CORE\OPTIONS" loads system options from the CORE\OPTIONS registry key.

Example: Load a Data Set of Saved System Options

Features: PROC OPTLOAD statement option
DATA=

Details

This example saves the current system option settings using the OPTSAVE procedure, modifies the YEARCUTOFF system option, and then loads the original set of system options.

Program

```
libname mysas "/your-file-path";  
proc options option=yearcutoff;  
run;  
proc optsave out=mysas.options;  
run;  
options yearcutoff=2000;  
proc options option=yearcutoff;  
run;  
proc optload data=mysas.options;  
run;  
  
proc options option=yearcutoff;  
run;
```

Program Description

These statements and procedures were submitted one at a time and not run as a SAS program to allow the display of the YEARCUTOFF option.

Assign the libref.

```
libname mysas "/your-file-path";
```

Display the value of the YEARCUTOFF= system option.

```
proc options option=yearcutoff;  
run;
```

Save the current system option settings in mysas.options.

```
proc optsave out=mysas.options;  
run;
```

Use the OPTIONS statement to set the YEARCUTOFF= system option to the value 2000.

```
options yearcutoff=2000;
```

Display the value of the YEARCUTOFF= system option.

```
proc options option=yearcutoff;  
run;
```

Load the saved system option settings.

```
proc optload data=mysas.options;  
run;
```

Display the value of the YEARCUTOFF= system option. After loading the saved system option settings, the value of the YEARCUTOFF= option has been restored to the original value.

```
proc options option=yearcutoff;  
run;
```

Log

Example Code 38.1 The SAS Log Shows the YEARCUTOFF= Value After Loading Options Using PROC OPTLOAD

```

1  libname mysas "/your-file-path";
NOTE: Libref MYSAS was successfully assigned as follows:
      Engine:          V9
      Physical Name:  /your-file-path

2  proc options option=yearcutoff;
3  run;

      SAS (r) Proprietary Software Release

      YEARCUTOFF=1940   Specifies the first year of a 100-year span that is used by date informats
                        and functions to read a two-digit year.

NOTE: PROCEDURE OPTIONS used (Total process time):
      real time          0.00 seconds
      cpu time           0.00 seconds

4  proc optsave out=mysas.options;
5  run;

NOTE: The data set MYSAS.OPTIONS has 259 observations and 2 variables.
NOTE: PROCEDURE OPTSAVE used (Total process time):
      real time          0.03 seconds
      cpu time           0.03 seconds

6  options yearcutoff=2000;

7  proc options option=yearcutoff;
8  run;

      SAS (r) Proprietary Software Release

      YEARCUTOFF=2000   Specifies the first year of a 100-year span that is used by date informats
                        and functions to read a two-digit year.

NOTE: PROCEDURE OPTIONS used (Total process time):
      real time          0.00 seconds
      cpu time           0.00 seconds

9  proc optload data=mysas.options;
10 run;

NOTE: PROCEDURE OPTLOAD used (Total process time):
      real time          0.06 seconds
      cpu time           0.01 seconds

```

```
11  proc options option=yearcutoff;  
12  run;
```

SAS (r) Proprietary Software Release

YEARCUTOFF=1940 Specifies the first year of a 100-year span that is used by date informats
 and functions to read a two-digit year.

NOTE: PROCEDURE OPTIONS used (Total process time):

real time	0.00 seconds
cpu time	0.00 seconds

OPTSAVE Procedure

Overview: OPTSAVE Procedure	1391
What Does the OPTSAVE Procedure Do?	1391
Syntax: OPTSAVE Procedure	1392
PROC OPTSAVE Statement	1392
Usage: OPTSAVE Procedure	1393
Determine If a Single Option Can Be Saved	1393
Create a List of Options That Can Be Saved	1394
Example: Saving System Options in a Data Set	1395

Overview: OPTSAVE Procedure

What Does the OPTSAVE Procedure Do?

PROC OPTSAVE saves the current SAS system option settings in the SAS registry or in a SAS data set.

SAS system options can be saved across SAS sessions. You can save the settings of the SAS system options in a SAS data set or registry key by using the PROC OPTSAVE statement.

Syntax: OPTSAVE Procedure

PROC OPTSAVE <options>;

Statement	Task	Example
PROC OPTSAVE	Save the current SAS system option settings to the SAS registry or to a SAS data set	Ex. 1

PROC OPTSAVE Statement

Saves the current SAS system option settings in the SAS registry or in a SAS data set.

Syntax

PROC OPTSAVE <options>;

Summary of Optional Arguments

- KEY="SAS registry key"**
Save SAS system option settings to a registry key.
- OUT=libref.dataset**
Save SAS system option settings to a SAS data set.

Optional Arguments

- KEY="SAS registry key"**
specifies the location in the SAS registry of stored SAS system option settings. The registry is retained in SASUSER. If SASUSER is not available, then the temporary WORK library is used. For example, KEY="OPTIONS" saves the system options in the OPTIONS registry key.
- Restriction "SAS registry key" names cannot span multiple lines.
- Requirements Separate the names in a sequence of key names with a backslash (\). Individual key names can contain any character except a backslash.

The length of a key name cannot exceed 255 characters (including the backslashes).

You must use quotation marks around the “SAS registry key” name.

Tip To specify a subkey, enter multiple key names starting with the root key.

CAUTION **If the key already exists, it will be overwritten.** If the specified key does not already exist in the current SAS registry, then the key is automatically created when option settings are saved in the SAS registry.

OUT=libref.dataset

specifies the names of the library and data set where SAS system option settings are saved. The SAS variable OPTNAME contains the character value of the SAS system option name. The SAS variable OPTVALUE contains the character value of the SAS system option setting.

Default If you omit the OUT= and the KEY= options, the procedure will use the default SAS library and data set. The default SAS library is where the current user profile resides. Unless you specify a SAS library, the default library is SASUSER. If SASUSER is in use by another active SAS session, then the temporary WORK library is the default location where the data set is saved. The default data set name is MYOPTS.

CAUTION **If the data set already exists, it will be overwritten.**

Usage: OPTSAVE Procedure

Determine If a Single Option Can Be Saved

You can specify DEFINE in the OPTIONS procedure to determine whether an option can be saved. In the log output, the line beginning with **Optsave:** indicates whether the option can be saved.

```
proc options option=pageno define;
run;
```

Example Code 39.1 The SAS Log Displaying Output for the Option Procedure DEFINE Option

```

Option Definition Information for SAS Option PAGENO
Group= LISTCONTROL
Group Description: Procedure output and display settings
Description: Resets the SAS output page number.
Type: The option value is of type LONG
      Range of Values: The minimum is 1 and the maximum is 2147483647
      Valid Syntax(any casing): MIN|MAX|n|nK|nM|nG|nT|hexadecimal
Numeric Format: Usage of LOGNUMBERFORMAT impacts the value format
When Can Set: Startup or anytime during the SAS Session
Restricted: Your Site Administrator can restrict modification of this option
Optsave: PROC Optsave or command Dmoptsave will save this option

```

Create a List of Options That Can Be Saved

Some system options cannot be saved.

Note: PROC OPTSAVE does not save these types of system options:

- SAS invocation options
 - SAS environment variable options
 - SAS system options that include passwords
 - SAS system options that are read-only and cannot be set
-

To create a list of options that can be saved, submit this SAS code:

```

proc options listoptsave;
run;

```

Here is a partial listing of options that can be saved:

Example Code 39.2 A Partial Listing of Options That Can Be Saved

Core options that can be saved with OPTSAVE

ACCESSIBLECHECK	Detect and log ODS output that is not accessible.
ACCESSIBLEGRAPH	Create accessible ODS graphics by default.
ACCESSIBLEPDF	Create accessible PDF files by default.
ACCESSIBLETABLE	Create accessible tables for enabled procedures, by default.
ANIMATION	Specifies whether to start or stop animation.
ANIMDURATION	Specifies the number of seconds that each animation frame displays.
ANIMLOOP	Specifies the number of iterations that animated images repeat.
ANIMOVERLAY	Specifies that animation frames are overlaid in order to view all frames.
APPLETLOC	Specifies the location of Java applets, which is typically a URL.
AUTOCORRECT	Automatically corrects misspelled procedure names and keywords, and global statement names.
AUTOSAVELOC	Specifies the location of the Program Editor auto-saved file.
AUTOSIGNON	Enables a SAS/CONNECT client to automatically submit the SIGNON command remotely with the RSUBMIT command.
BINDING	Specifies the binding edge type of duplexed printed output.
BOMFILE	Writes the byte order mark (BOM) prefix when a Unicode-encoded file is written to an external file.
BOTTOMMARGIN	Specifies the size of the margin at the bottom of a printed page.
BUFNO	Specifies the number of buffers for processing SAS data sets.
BUFSIZE	Specifies the size of a buffer page for output SAS data sets.
BYERR	SAS issues an error message and stops processing if the SORT procedure attempts to sort a <code>_NULL_</code> data set.
BYLINE	Prints the BY line above each BY group.
BYSORTED	Requires observations in one or more data sets to be sorted in alphabetic or numeric order.
CAPS	Converts certain types of input, and all data lines, into uppercase characters.
CARDIMAGE	Processes SAS source code and data lines as 80-byte records.
CBUFNO	Specifies the number of extra page buffers to allocate for each open SAS catalog.
CENTER	Center SAS procedure output.

Example: Saving System Options in a Data Set

Features: PROC OPTSAVE statement option
OUT=

Details

This example saves the current system option settings using the OPTSAVE procedure.

Program

```
libname mysas "/your-file-path";

proc optsave out=mysas.options;
run;
```

Program Description

Create a libref.

```
libname mysas "/your-file-path";
```

Save the current system option settings.

```
proc optsave out=mysas.options;
run;
```

Log

Example Code 39.3 The SAS Log Shows Processing of PROC OPTSAVE

```
1  libname mysas "/your-file-path";
NOTE: Libref MYSAS was successfully assigned as follows:
      Engine:          V9
      Physical Name:   /your-file-path

2  proc optsave out=mysas.options;
3  run;

NOTE: The data set MYSAS.OPTIONS has 289 observations and 2 variables.
NOTE: PROCEDURE OPTSAVE used (Total process time):
      real time          0.03 seconds
      cpu time           0.03 seconds
```

PRINT Procedure

Overview: PRINT Procedure	1397
What Does the PRINT Procedure Do?	1398
A Simple Report	1398
Customized Report	1399
Syntax: PRINT Procedure	1400
PROC PRINT Statement	1402
ATTRIB Statement	1418
BY Statement	1419
FORMAT Statement	1425
ID Statement	1426
LABEL Statement	1428
PAGEBY Statement	1429
SUM Statement	1430
SUMBY Statement	1432
VAR Statement	1432
WHERE Statement	1434
Usage: PRINT Procedure	1436
Use ODS Styles with PROC PRINT	1436
Error Processing in the PRINT Procedure Output	1447
Results: PRINT Procedure	1447
About PROC PRINT Output	1447
Page Layout for HTML, the Default ODS Destination	1448
Page Layout for Limited Page Sizes	1448
Examples: PRINT Procedure	1451
Example 1: Print a CAS Table	1451
Example 2: Selecting Variables to Print	1453
Example 3: Customizing Text in Column Headings	1456
Example 4: Creating Separate Sections of a Report for Groups of Observations	1460
Example 5: Summing Numeric Variables with One BY Group	1468
Example 6: Summing Numeric Variables with Multiple BY Variables	1473
Example 7: Limiting the Number of Sums in a Report	1478
Example 8: Controlling the Layout of a Report with Many Variables	1483
Example 9: Creating a Customized Layout with BY Groups and ID Variables	1485
Example 10: Printing All the Data Sets in a SAS Library	1490

Overview: PRINT Procedure

What Does the PRINT Procedure Do?

The PRINT procedure prints the observations in a SAS data set or rows from a table using all or some of the variables. You can create a variety of reports ranging from a simple table to a highly customized report that groups the data and calculates totals and subtotals for numeric variables.

A Simple Report

The following output illustrates the simplest type of report that you can produce. The statements that produce the output follow. [“Example 2: Selecting Variables to Print” on page 1453](#) creates the data set EXPREV.

```
options obs=10;
proc print data=exprev;
run;
```

TIP The OBS= system option is valid for all steps during your current SAS session or until you change the setting. To set the number of observations for a single PROC step, use the OBS= data set option:

```
proc print data=exprev(obs=10);
```

For more information, see [“OBS= Data Set Option” in SAS Data Set Options: Reference](#) and [“OBS= System Option” in SAS System Options: Reference](#)

Obs	Country	Emp_ID	Order_Date	Ship_Date	Sale_Type	Quantity	Price	Cost
1	Antarctica	99999999	1/1/18	1/7/18	Internet	2	92.6	20.70
2	Puerto Rico	99999999	1/1/18	1/5/18	Catalog	14	51.2	12.10
3	Virgin Islands (U.S.)	99999999	1/1/18	1/4/18	In Store	25	31.1	15.65
4	Aruba	99999999	1/1/18	1/4/18	Catalog	30	123.7	59.00
5	Bahamas	99999999	1/1/18	1/4/18	Catalog	8	113.4	28.45
6	Bermuda	99999999	1/1/18	1/4/18	Catalog	7	41.0	9.25
7	Belize	120458	1/2/18	1/2/18	In Store	2	146.4	36.70
8	British Virgin Islands	99999999	1/2/18	1/5/18	Catalog	11	40.2	20.20
9	Canada	99999999	1/2/18	1/5/18	Catalog	100	11.8	5.00
10	Cayman Islands	120454	1/2/18	1/2/18	In Store	20	71.0	32.30

This next example creates the CAS table Mycas.Cars as a subset of the Sashelp.cars data set:

```
options cashost="cloud.example.com" casport=5555;
cas casauto;
libname mycas cas;

proc casutil outcaslib="casuser";
load data=sashelp.cars replace;
run;

data mycas.cars;
  set mycas.cars(where=(weight>6000));
  keep make model type;
run;

proc print data=mycas.cars;
  title "Cars Greater Than 6000 Pounds";
run;
```

Cars Greater Than 6000 Pounds

Obs	Make	Model	Type
1	Ford	Excursion 6.8 XLT	SUV
2	Hummer	H2	SUV
3	GMC	Yukon XL 2500 SLT	SUV

Customized Report

The following HTML report is a customized report that is produced by PROC PRINT using ODS. The statements that create this report do the following:

- customize the title and the column headings

- customize the appearance of the report
- place dollar signs and commas in numeric output
- selectively include and control the order of variables in the report
- group the data by JobCode
- sum the values for Salary for each job code and for all job codes
- add a label for the summary line and the grand total line.

For an explanation of the program that produces this report, see [“Program: Creating an HTML5 Report with the STYLE Option”](#) on page 1488.

Figure 40.1 Customized Report Produced by PROC PRINT Using ODS

Expenses Incurred for Salaries for Flight Attendants and Mechanics

JobCode	Gender	Salary
FA1	F	\$23,177.00
	F	\$22,454.00
	M	\$22,268.00
Total		\$67,899.00

JobCode	Gender	Salary
FA2	F	\$28,888.00
	F	\$27,787.00
	M	\$28,572.00
Total		\$85,247.00

JobCode	Gender	Salary
FA3	F	\$32,886.00
	F	\$33,419.00
	M	\$32,217.00
Total		\$98,522.00

JobCode	Gender	Salary
ME1	M	\$29,769.00
Grand Total		\$281,437.00

Syntax: PRINT Procedure

Interactions:

A common practice is to sort a data set using PROC SORT before you use the PROC PRINT BY statement.

If you sort a CAS table with VARCHAR variables using PROC SORT, VARCHAR variables are converted to CHAR variables.

Note: PROC PRINT supports the VARCHAR data type for CAS tables.

Tips: Each password and encryption key option must be coded on a separate line to ensure that they are properly blotted in the log.

Supports the Output Delivery System. For details, see [“Output Delivery System: Basic Concepts” in SAS Output Delivery System: User’s Guide](#).

You can use the ATTRIB, FORMAT, LABEL, TITLE, and WHERE statements.

SAS includes checks to verify that the PROC PRINT output is accessible for the visually impaired. When you set the [ACCESSIBLECHECK system option](#), SAS verifies whether the output is accessible. For best practices about creating accessible output, see [Creating Accessible SAS Viya Platform Output Using ODS and ODS Graphics](#).

PROC PRINT *<options>;*

ATTRIB *variable-list(s) attribute-list(s);*

BY *<DESCENDING> variable-1 <<DESCENDING> variable-2 ...> <NOTSORTED>;*

PAGEBY *BY-variable;*

SUMBY *BY-variable;*

FORMAT *variable-1 <...variable-n> <format> <DEFAULT=default-format>;*

ID *variable(s)*

</ STYLE <(location(s))>=<style-override>;

LABEL *variable-1=label-1<...variable-n=label-n>;*

SUM *variable(s)*

</ STYLE <(location(s))>=<style-override>;

VAR *variable(s)*

</ STYLE <(location(s))>=<style-override> >;

WHERE *where-expression-1*

< logical-operator where-expression-n>;

Statement	Task	Example
PROC PRINT	Print observations in a data set	Ex. 1, Ex. 2, Ex. 3, Ex. 4, Ex. 6, Ex. 9
BY	Produce a separate section of the report for each BY group	Ex. 4, Ex. 5, Ex. 6, Ex. 7, Ex. 9
ID	Identify observations by the formatted values of the variables that you list instead of by observation numbers	Ex. 8
PAGEBY	Control page ejects that occur before a page is full	Ex. 4
SUMBY	Limit the number of sums that appear in the report	Ex. 5, Ex. 6, Ex. 7, Ex. 9

Statement	Task	Example
SUM	Total values of numeric variables	Ex. 7
VAR	Select variables that appear in the report and determine their order	Ex. 2, Ex. 3, Ex. 9

PROC PRINT Statement

Prints observations in a SAS data set using some or all of the variables.

Syntax

PROC PRINT *<options>*;

Summary of Optional Arguments

CONTENTS=*link-text* **<#BYLINE>** **<#BYVAL>** **<#BYVAR>**
specifies text for the links in the table of contents.

DATA=*SAS-data-set*
specifies the SAS data set to print.

Control column format

GRANDTOTAL_LABEL=*'label'*
displays a label on the grand total line.

HEADING=HORIZONTAL | VERTICAL
controls the orientation of the column headings.

LABEL
specifies to use the variables' labels as column headings.

NO4KLEAPRULE
specifies to ignore the 4,000-year correction rule for leap years.

SPLIT=*'split-character'*
specifies the split character, which controls line breaks in column headings.

STYLE **<(location(s))>=****<style-override(s)>**
specify one or more ODS style overrides to modify the default style element and attributes in a specific area of a report.

SUMLABEL

NOSUMLABEL

SUMLABEL=*'label'*
specifies whether to display a label on the summary line for a BY group.

Control general format**BLANKLINE**=*n***BLANKLINE**=(**COUNT**=*n* <**STYLE**=[*style-attribute-specification(s)*]>)writes a blank line after *n* observations.**DOUBLE**

writes a blank line between observations.

N<="*string-1*"> <="*string-2*">

prints the number of observations in the data set, in BY groups, or both and specifies explanatory text to print with the number.

NOOBS

suppresses the column in the output that identifies each observation by number.

OBS="*column-header*"

specifies a column heading for the column that identifies each observation by number.

ROUND

rounds unformatted numeric values to two decimal places.

Control page format**ROWS**=*page-format*

formats the rows on a page.

UNIFORM

specifies to use each variable's formatted width as its column width on all pages.

WIDTH=**FULL** | **MINIMUM** | **UNIFORM** | **UNIFORMBY**

determines the column width for each variable.

Optional Arguments

BLANKLINE=*n***BLANKLINE**=(**COUNT**=*n* <**STYLE**=[*style-attribute-specification(s)*]>)specifies to insert a blank line after every *n* observations. The observation count is reset to 0 at the beginning of each BY group for all ODS destinations.*n***COUNT**=*n*

specifies the observation number after which SAS inserts a blank line.

STYLE=[*style-attribute-specification(s)*]

specifies the style attribute to use for the blank line.

Default DATA

Tip You can use the BACKGROUND_COLOR style attribute to make a visual distinction between observations using color.

See [The STYLE= option on page 1410](#) for valid style attributes.Example ["Example 2: Selecting Variables to Print" on page 1453](#)Alias **BLANK**=

Tip SAS includes checks to verify that the PROC PRINT output is accessible for the visually impaired. When you specify the `BLANKLINE=` option, the output that PROC PRINT creates includes one or more lines that are not data. Screen readers and users might interpret these lines incorrectly. When you set the [ACCESSIBLECHECK system option](#), SAS checks whether the `BLANKLINE` option has been used to add blank lines to the output. If blank lines are in the output, SAS writes a warning message to the SAS log. For best practices about creating accessible output, see [Creating Accessible SAS Viya Platform Output Using ODS and ODS Graphics](#).

CONTENTS=*link-text* <#BYLINE> <#BYVAL> <#BYVAR>

specifies the text for the links in the table contents file to the output produced by the PROC PRINT statement.

link-text

specifies text to use in the table of contents.

#BYLINE

substitutes the entire BY line without leading or trailing blanks for #BYLINE in the text string. The BY line uses the format *variable-name=value*.

#BYVAL_n

#BYVAL(*BY-variable-name*)

substitutes the current value of the specified BY variable for #BYVAL in the text string. Specify the variable with one of these values:

n

specifies a variable by its position in the BY statement. For example, #BYVAL2 specifies the second variable in the BY statement.

BY-variable-name

specifies a variable from the BY statement by its name. For example, #BYVAL(YEAR) specifies the BY variable, YEAR. *variable-name* is not case sensitive.

#BYVAR_n

#BYVAR(*BY-variable-name*)

substitutes the name of the BY variable or the label associated with the variable (whatever the BY line would normally display) for #BYVAR in the text string. Specify the variable with one of these values:

n

specifies a variable by its position in the BY statement. For example, #BYVAR2 specifies the second variable in the BY statement.

BY-variable-name

specifies a variable from the BY statement by its name. For example, #BYVAR(SITES) specifies the BY variable, SITES. *variable-name* is not case sensitive.

Restrictions CONTENTS= does not affect the HTML body file. It affects only the HTML contents file.

CONTENTS= is not valid for the ODS LISTING destination.

See For information about HTML output, see [Files Produced by the HTML Destination](#) and “[ODS HTML5 Statement](#)” in *SAS Output Delivery System: User's Guide*.

DATA=SAS-*data-set*

specifies the SAS data set to print.

See “[Input Data Sets](#)” on page 24

DOUBLE

writes a blank line between observations.

Alias D

Restriction DOUBLE is valid only for the ODS LISTING destination.

Example “[Example 2: Selecting Variables to Print](#)” on page 1453

GRANDTOTAL_LABEL='label'

displays a label on the grand total line. You can include the #BYVAR and #BYVAL variables in 'label'.

Aliases GRAND_LABEL=
GRANDTOT_LABEL=
GTOT_LABEL=
GTOTAL_LABEL=

Restriction The #BYVAR and #BYVAL variables are not supported for the LISTING destination.

Tip SAS includes checks to verify that the PROC PRINT output is accessible for the visually impaired. When you set the [ACCESSIBLECHECK system option](#), SAS verifies whether a label is available for both the SUMLABEL and the GRANDTOTAL_LABEL options. If SAS detects that the output does not have a label for the summary and grand total values, SAS writes a message to the log. For best practices about creating accessible output, see [Creating Accessible SAS Viya Platform Output Using ODS and ODS Graphics](#).

Example “[Example 6: Summing Numeric Variables with Multiple BY Variables](#)” on page 1473

HEADING=HORIZONTAL | VERTICAL

controls the orientation of the column headings.

HORIZONTAL

prints all column headings horizontally.

Alias H

VERTICAL

prints all column headings vertically.

Alias V

Restriction For LISTING output, if the column heading is too long for the page, the variable name is used in place of a label.

Default Headings are either all horizontal or all vertical. If you omit HEADING=, PROC PRINT determines the direction of the column headings as follows:

If you do not use LABEL, spacing specifies whether column headings are vertical or horizontal.

If you use LABEL and at least one variable has a label, all headings are horizontal.

LABEL

specifies to use the variables' labels as column headings.

Alias L

Default PROC PRINT uses the name of the variable as the column heading in the following two circumstances:

1. if you omit the LABEL option in the PROC PRINT statement, even if the PROC PRINT step contains a LABEL statement
2. if a variable does not have a label

Interactions By default, if you specify LABEL and at least one variable has a label, PROC PRINT prints all column headings horizontally. Therefore, using LABEL might increase the number of pages of output. (Use HEADING=VERTICAL in the PROC PRINT statement to print vertical column headings.)

PROC PRINT sometimes conserves space by splitting labels across multiple lines. Use SPLIT= in the PROC PRINT statement to control where these splits occur. You do not need to use LABEL if you use SPLIT=.

Note The SAS system option LABEL must be in effect in order for any procedure to use labels. For more information see [“LABEL System Option” in SAS System Options: Reference](#).

Tip To create a blank column heading for a variable, use this LABEL statement in your PROC PRINT step:

```
label variable-name='00'x;
```

See For information about using the LABEL statement to create temporary labels in procedures, see [“LABEL Statement” in SAS DATA Step Statements: Reference](#).

For information about using the LABEL statement in a DATA step to create permanent labels, see [“LABEL Statement” in SAS DATA Step Statements: Reference](#).

Example [“Example 4: Creating Separate Sections of a Report for Groups of Observations” on page 1460](#)

N=<“string-1”> <“string-2”>

prints the number of observations in the data set, in BY groups, or both and specifies explanatory text to print with the number.

N Option Use	PROC PRINT Action
With neither a BY nor a SUM statement	Prints the number of observations in the data set at the end of the report and labels the number with the value of <i>string-1</i> .
With a BY statement	Prints the number of observations in the BY group at the end of each BY group and labels the number with the value of <i>string-1</i> .
With a BY statement and a SUM statement	Prints the number of observations in the BY group at the end of each BY group and prints the number of observations in the data set at the end of the report. The numbers for BY groups are labeled with <i>string-1</i> ; the number for the entire data set is labeled with <i>string-2</i> .

Tip SAS includes checks to verify that the PROC PRINT output is accessible for the visually impaired. When you specify the N= option, the output that PROC PRINT creates includes a text line that is not data. Screen readers might interpret this line of text as data. When you set the [ACCESSIBLECHECK system option](#), SAS verifies whether the output is accessible. If the output contains the text that is not data, SAS writes a warning message to the SAS log. For best practices about creating accessible output, see [Creating Accessible SAS Viya Platform Output Using ODS and ODS Graphics](#).

Examples [“Example 3: Customizing Text in Column Headings” on page 1456](#)
[“Example 4: Creating Separate Sections of a Report for Groups of Observations” on page 1460](#)
[“Example 5: Summing Numeric Variables with One BY Group” on page 1468](#)

NO4KLEAPRULE

specifies to ignore the 4,000-year correction rule for leap years. SAS follows the Gregorian calendar rules for leap years. These rules are a common standard in computing. However, compared to most data sources, SAS provides greater precision with the following additional correction rule: In SAS, every year that can be evenly divided by 4,000 is a common year and is not leap year.

Use this argument only if your data source does not follow the 4,000-year correction rule for leap years. The rule affects the PROC PRINT output of dates or datetimes greater than February 28, 4000. Do not use this argument for data created by SAS.

Note This option was added in [2026.09](#).

NOOBS

suppresses the column in the output that identifies each observation by number.

Example [“Example 4: Creating Separate Sections of a Report for Groups of Observations” on page 1460](#)

OBS=“column-header”

specifies a column heading for the column that identifies each observation by number.

Tip OBS= honors the split character. (See the discussion of the [SPLIT= option on page 1410](#).)

Example [“Example 3: Customizing Text in Column Headings” on page 1456](#)

ROUND

rounds unformatted numeric values to two decimal places. (Formatted values are already rounded by the format to the specified number of decimal places.) For both formatted and unformatted variables, PROC PRINT uses these rounded values to calculate any sums in the report.

If you omit ROUND, PROC PRINT adds the actual values of the rows to obtain the sum *even though it displays the formatted (rounded) values*. Any sums are also rounded by the format, but they include only one rounding error, that of rounding the sum of the actual values. The ROUND option, on the other hand, rounds values before summing them, so there might be multiple rounding errors. The results without ROUND are more accurate, but ROUND is useful for published reports where it is important for the total to be the sum of the printed (rounded) values.

Be aware that the results from PROC PRINT with the ROUND option might differ from the results of summing the same data with other methods such as PROC MEANS or the DATA step. Consider a simple case in which the following is true:

- The data set contains three values for X: .003, .004, and .009.
- X has a format of 5.2.

Depending on how you calculate the sum, you can get three different answers: 0.02, 0.01, and 0.016. The following figure shows the results of calculating the sum with PROC PRINT (without and with the ROUND option) and PROC MEANS.

Figure 40.2 Three Methods of Summing Variables

Actual	Values	PROC PRINT the ROUND	without option	PROC PRINT the ROUND	with option	PROC MEANS
		OBS	X	OBS	X	Analysis Variable : X
.003		1	0.00	1	0.00	Sum
.004		2	0.00	2	0.00	
.009		3	0.01	3	0.01	0.0160000
=====			=====		=====	
.016			0.02		0.01	

Notice that the sum produced without the ROUND option (.02) is closer to the actual result (0.16) than the sum produced with ROUND (0.01). However, the sum produced with ROUND reflects the numbers that are displayed in the report.

Alias R

CAUTION Do not use ROUND with PICTURE formats. ROUND is for use with numeric values. SAS procedures treat variables that have picture formats as character variables. Using ROUND with such variables might lead to unexpected results.

ROWS=page-format

formats the rows on a page. Currently, PAGE is the only value that you can use for *page-format*:

PAGE

prints only one row of variables for each observation per page. When you use ROWS=PAGE, PROC PRINT does not divide the page into sections; it prints as many observations as possible on each page. If the observations do not fill the last page of the output, PROC PRINT divides the last page into sections and prints all the variables for the last few observations.

- Restriction ROWS= is valid only for the ODS LISTING destination. Therefore, HTML output from PROC PRINT appears the same if you use ROWS=.
- Tip The PAGE value can reduce the number of pages in the output if the data set contains large numbers of variables and observations. However, if the data set contains a large number of variables but few observations, the PAGE value can increase the number of pages in the output.
- See [“Page Layout for Limited Page Sizes” on page 1448](#) for discussion of the default layout.
- Example [“Example 8: Controlling the Layout of a Report with Many Variables” on page 1483](#)

SPLIT='split-character'

specifies the split character, which controls line breaks in column headings. It also uses labels as column headings. PROC PRINT breaks a column heading when it reaches the split character and continues the header on the next line. The split character is not part of the column heading although each occurrence of the split character counts toward the 256-character maximum for a label.

Alias S=

Interactions You do not need to use both LABEL and SPLIT= because SPLIT= implies the use of labels.

The OBS= option honors the split character. (See the discussion of "OBS="column-header"" on page 1408.)

Note PROC PRINT does not split labels of BY variables in the heading preceding each BY group, a summary label, or a grand total level, even if you specify SPLIT=. Instead, PROC PRINT replaces the split character with a blank.

Example "Example 3: Customizing Text in Column Headings" on page 1456

STYLE <(location(s))>=<style-override(s)>

specify one or more ODS style overrides to modify the default style element and attributes in a specific area of a report.

You can specify a style override in two ways:

- Specify a style element. A style element is a collection of style attributes that apply to a particular part of the output for a SAS program.
- Specify a style attribute. A style attribute is a name-value pair that describes a single behavioral or visual aspect of a piece of output. This is the most specific method of changing the appearance of your output.

style-override has the following form:

```
style-element-name | [style-attribute-name-1=style-attribute-value-1
    <style-attribute-name-2=style-attribute-value-2 ...>]
```

location

identifies the part of the report that the STYLE option affects. If *location(s)* is not specified, PROC PRINT determines the location to where the style override is applied based on the statement, the specified style element, and the style attribute.

The following table shows the available locations and the other statements in which you can specify them.

Table 40.1 Specifying Locations in the STYLE Option

Location	Location Alias	Affected Report Part	Can Also Be Used in These Statements
BYLABEL	BYSUMLABEL	Label for the BY variable on the	None

Location	Location Alias	Affected Report Part	Can Also Be Used in These Statements
	BYLBL BYSUMLBL	line containing the SUM totals	
DATA	COLUMN COL	All data except for data in the OBS column or the ID columns Or Data in the ID columns when the DATA location is specified in the STYLE= option of the ID statement	VAR ID SUM
GRANDTOTAL	GRANDTOT GRAND GTOTAL GTOT	SUM line containing the grand totals for the whole report	SUM
HEADER	HEAD HDR	All column headings except for the OBS column or the ID columns ¹ Or All column headings of the ID columns when the HEADER location is specified in the STYLE= option of the ID statement	VAR ID SUM
N	None	N= table and contents	None
OBS	OBSDATA OBSCOLUMN OBSCOL	Data in the OBS column or the ID columns unless the DATA location is specified in the STYLE= option of the ID statement	None

Location	Location Alias	Affected Report Part	Can Also Be Used in These Statements
OBSHEADER	OBSHEAD OBSHDR	Header of the OBS column or the ID columns unless the HEADER location is specified in the STYLE= option of the ID statement ¹	None
TABLE	REPORT	Structural part of the report - that is, the underlying table used to set things like the width of the border and the space between cells	None
TOTAL	TOT BYSUMLINE BYLINE BYSUM	SUM line containing totals for each BY group	SUM

¹ Use the OBSHEADER location in the STYLE= option of the PROC PRINT statement to format the OBS column and the ID columns. The PROC PRINT statement STYLE= option in your existing programs might need to include the OBSHEADER location as well as the HEADER location.

Figure 40.3 PROC PRINT Areas and Corresponding Statements

table		
obsheader	header	header
obs	data	data
obs	data	data
obs	data	data
obs	data	data
obs	data	data
bylabel	total	total
grandtotal	grandtotal	grandtotal
n		

If the same style attributes appear for the OBSHEADER location in the PROC PRINT statement and the HEADER location in the ID statement, the HEADER location attributes override the OBSHEADER attributes. All other style attributes for the ID columns in both the PROC PRINT statement and the ID statement are merged to create the style for the ID columns. For example, in the PROC PRINT statement, the attributes for the OBSHEADER location are {fontsize=5 fontweight=bold}. In the ID statement, the attributes for the HEADER location are [fontsize=6 fontstyle=italic]. The resulting style for the ID column is [fontsize=6 fontweight=bold fontstyle=italic].

```
proc print data=expres style(obsheader)={fontsize=5 fontweight=bold};
  id country / style(header)=[fontsize=6 fontstyle=italic];
run;
```

Country	Emp_ID	Order_Date	Ship_Date	Sale_Type	Quantity	Price	Cost
Bermuda	99999999	1/1/18	1/4/18	Catalog	7	41.0	9.25

If the same style attributes appear for the OBS location in the PROC PRINT statement and the DATA location in the ID statement, the DATA location attributes override the OBS attributes. All other style attributes for the ID columns in both the PROC PRINT statement and the ID statement are merged to create the style for the ID columns. For example, in the PROC PRINT statement, the attributes for the OBS location are {backgroundcolor=light gray color=blue}. In the ID statement, the attributes for the DATA location are [color=white fontstyle=italic]. The

resulting style for the ID column is [backgroundcolor=light gray color=white fontstyle=italic].

```
proc print data=exprev style(obs)={backgroundcolor=light gray color=blue};
  id country / style(data)=[color=white fontstyle=italic];
run;
```

Country	Emp_ID	Order_Date	Ship_Date	Sale_Type	Quantity	Price	Cost
Puerto Rico	99999999	1/1/18	1/5/18	Catalog	14	51.2	12.10
Aruba	99999999	1/1/18	1/4/18	Catalog	30	123.7	59.00
Bahamas	99999999	1/1/18	1/4/18	Catalog	8	113.4	28.45
Bermuda	99999999	1/1/18	1/4/18	Catalog	7	41.0	9.25
British Virgin Islands	99999999	1/2/18	1/5/18	Catalog	11	40.2	20.20
Canada	99999999	1/2/18	1/5/18	Catalog	100	11.8	5.00
Virgin Islands (U.S.)	99999999	1/1/18	1/4/18	In Store	25	31.1	15.65
Belize	120458	1/2/18	1/2/18	In Store	2	146.4	36.70
Cayman Islands	120454	1/2/18	1/2/18	In Store	20	71.0	32.30
Antarctica	99999999	1/1/18	1/7/18	Internet	2	92.6	20.70

style-element-name

is the name of a style element in a style template that is registered with the Output Delivery System. SAS provides some [style templates](#). Users can create their own style templates with the TEMPLATE procedure. See [SAS Output Delivery System: Procedures Guide](#).

When style elements are processed, more specific style elements override less specific style elements. For a table of default style elements and style attributes for each PROC PRINT location, see [Table 40.84 on page 1445](#).

Tip You can use compound names and formats for style element names. An example of using a compound style element name is `style(obsheader)=data.italic.red;`. An example of using a format element name is `style=$cities`. For more information about using formats, see the [SAS Output Delivery System: Procedures Guide](#).

style-attribute-specification

describes the style attribute to change. Each *style-attribute-specification* has this general form:

style-attribute-name=style-attribute-value

You can set these style attributes in the TABLE location:

BACKGROUNDColor=	Fontwidth= 1
BackgroundImage=	Color= 1
BorderColor=	Frame=
BorderColorDark=	HTMLClass=
BorderColorLight=	TextAlign=

BORDERWIDTH=	OUTPUTWIDTH=
CELLPADDING=	POSTHTML=
CELLSPACING=	POSTIMAGE=
FONT= ¹	POSTTEXT=
FONTFAMILY= ¹	PREHTML=
FONTSIZE= ¹	PREIMAGE=
FONTSTYLE= ¹	PRETEXT=
FONTWEIGHT= ¹	RULES=

¹ When you use these attributes, they affect only the text that is specified with the PRETEXT=, POSTTEXT=, PREHTML=, and POSTHTML= attributes. To alter the foreground color or the font for the text that appears in the table, you must set the corresponding attribute in a location that affects the cells rather than the table.

You can set these style attributes in all locations other than TABLE:

ASIS=	FONTWIDTH=
BACKGROUNDCOLOR=	HREFTARGET=
BACKGROUNDIMAGE=	CLASS=
BORDERCOLOR=	TEXTALIGN=
BORDERCOLORDARK=	NOBREAKSPACE=
BORDERCOLORLIGHT=	POSTHTML=
BORDERWIDTH=	POSTIMAGE=
HEIGHT=	POSTTEXT=
CELLWIDTH=	PREHTML=
FLYOVER=	PREIMAGE=
FONT=	PRETEXT=
FONTFAMILY=	PROTECTSPECIALCHARACTERS=
FONTSIZE=	TAGATTR=

FONTSTYLE=

URL=

FONTWEIGHT=

VERTICALALIGN=

Restriction STYLE= is not valid for the ODS LISTING or ODS OUTPUT destinations.

See For a table of style attributes that can be used with PROC TABULATE, PROC REPORT, and PROC PRINT, see [Table 40.83 on page 1441](#).

For a table of default style elements and style attributes for each PROC PRINT location, see [Table 40.84 on page 1445](#).

For more information about using styles with PROC PRINT, see [“Use ODS Styles with PROC PRINT” on page 1436](#).

For information about style attributes and PROC TEMPLATE, see [DEFINE Style Statement](#) in *SAS Output Delivery System: Procedures Guide*.

SUMLABEL**NOSUMLABEL****SUMLABEL='label'**

specifies whether to display a label on the summary line for a BY group.

SUMLABEL

specifies to use the variable label, if it exists, as the label on the summary line in place of the variable name.

NOSUMLABEL

specifies to leave the label on the summary line blank. Alternatively, you can use SUMLABEL="" (two single or double quotation marks with no space between them) to indicate a blank on the summary line.

SUMLABEL='label'

specifies the text to use as a label on the summary line of a BY group. You can include the #BYVAR and #BYVAL variables in 'label'.

Restriction The #BYVAR and #BYVAL variables are not supported for the LISTING destination.

Default If you omit SUMLABEL, PROC PRINT uses the BY variable names in the summary line.

Tip SAS includes checks to verify that the PROC PRINT output is accessible for the visually impaired. When you set the [ACCESSIBLECHECK system option](#), SAS verifies whether a label is available for both the SUMLABEL and the GRANDTOTAL_LABEL options. If SAS detects that the output does not have a label for the summary and grand total values, SAS writes a message to the log. For

best practices about creating accessible output, see [Creating Accessible SAS Viya Platform Output Using ODS and ODS Graphics](#).

Examples “Example 5: Summing Numeric Variables with One BY Group” on page 1468

“Example 6: Summing Numeric Variables with Multiple BY Variables” on page 1473

UNIFORM

See [WIDTH=UNIFORM](#) on page 1417.

WIDTH=FULL | MINIMUM | UNIFORM | UNIFORMBY

determines the column width for each variable.

FULL

uses a variable's formatted width as the column width. If the variable does not have a format that explicitly specifies a field width, PROC PRINT uses the default width. For a character variable, the default width is the length of the variable. For a numeric variable, the default width is 12. When you use WIDTH=FULL, the column widths do not vary from page to page.

Tip Using WIDTH=FULL can reduce execution time.

MINIMUM

uses for each variable the minimum column width that accommodates all values of the variable.

Alias MIN

UNIFORM

uses each variable's formatted width as its column width on all pages. If the variable does not have a format that explicitly specifies a field width, PROC PRINT uses the widest data value as the column width. When you specify WIDTH=UNIFORM, PROC PRINT normally needs to read the data set twice. However, if all the variables in the data set have formats that explicitly specify a field width (for example, BEST12. but not BEST.), PROC PRINT reads the data set only once.

Alias

U

Restriction When not all variables have formats that explicitly specify a width, you cannot use WIDTH=UNIFORM with an engine that supports concurrent access if another user is updating the data set at the same time.

Tips If the data set is large and you want a uniform report, you can save computer resources by using formats that explicitly specify a field width so that PROC PRINT reads the data only once.

WIDTH=UNIFORM is the same as UNIFORM.

UNIFORMBY

formats all columns uniformly within a BY group, using each variable's formatted width as its column width. If the variable does not have a format

that explicitly specifies a field width, PROC PRINT uses the widest data value as the column width.

Alias UBY

Restriction You cannot use UNIFORMBY with a sequential data set.

Default If you omit WIDTH= and do not specify the UNIFORM option, PROC PRINT individually constructs each page of output. The procedure analyzes the data for a page and decides how best to display them. Therefore, column widths might differ from one page to another.

Restriction WIDTH= is valid only for the LISTING destination.

Tip Column width is affected not only by variable width but also by the length of column headings. Long column headings might lessen the usefulness of WIDTH=.

See For a discussion of default column widths, see [“Column Width” on page 1450](#).

ATTRIB Statement

Associates a format, informat, label, and length with one or more variables.

Syntax

ATTRIB *variable-list(s) attribute-list(s);*

Arguments

variable-list(s)

names the variables that you want to associate with the attributes.

TIP List the variables in any form that SAS allows.

attribute-list(s)

specifies one or more attributes to assign to *variable-list*. Specify one or more of these attributes in the ATTRIB statement:

FORMAT=*format*

associates a format with variables in *variable-list*.

Tip The format can be either a standard SAS format or a format that is defined with the FORMAT procedure.

INFORMAT=informat

associates an informat with variables in *variable-list*.

Tip The informat can be either a standard SAS informat or an informat that is defined with the FORMAT procedure.

LABEL='label'

associates a label with variables in *variable-list*.

See Also

[“ATTRIB Statement” in SAS DATA Step Statements: Reference.](#)

BY Statement

Orders the output according to the BY groups.

Restriction: Do not specify a group variable in the BY statement. The BY variable cannot be the same as the group variable. A *group variable* is a variable in the input data set that is used to categorize chart variable values into groups. For more information, see [“Group Variables” on page 1760](#).

Examples: [“Example 4: Creating Separate Sections of a Report for Groups of Observations” on page 1460](#)
[“Example 5: Summing Numeric Variables with One BY Group” on page 1468](#)
[“Example 6: Summing Numeric Variables with Multiple BY Variables” on page 1473](#)
[“Example 7: Limiting the Number of Sums in a Report” on page 1478](#)
[“Example 9: Creating a Customized Layout with BY Groups and ID Variables” on page 1485](#)

Syntax

```
BY <DESCENDING> variable-1
      <... <DESCENDING> variable-n>
      <NOTSORTED>;
```

Required Argument

variable

specifies the variable that the procedure uses to form BY groups. You can specify more than one variable. If you do not use the NOTSORTED option in the BY statement, then either the observations in the data set must be sorted by all the variables that you specify, or they must be indexed appropriately. Variables in a BY statement are called *BY variables*.

Optional Arguments

DESCENDING

specifies that the observations are sorted in descending order by the variable that immediately follows the word DESCENDING in the BY statement.

NOTSORTED

specifies that observations are not necessarily sorted in alphabetic or numeric order. The observations are grouped in another way (for example, chronological order).

The requirement for ordering or indexing observations according to the values of BY variables is suspended for BY-group processing when you use the NOTSORTED option. In fact, the procedure does not use an index if you specify NOTSORTED. The procedure defines a BY group as a set of contiguous observations that have the same values for all BY variables. If observations with the same values for the BY variables are not contiguous, then the procedure treats each contiguous set as a separate BY group.

You cannot use the NOTSORTED option in a PROC SORT step.

Note: You cannot use the GROUPFORMAT option, which is available in the BY statement in a DATA step, in a BY statement in any PROC step.

Details

BY-Group Processing

Procedures create output for each BY group. For example, the elementary statistics procedures and the scoring procedures perform separate analyses for each BY group. The reporting and ODS procedures produce a report for each BY group.

Note: All Base SAS procedures except PROC PRINT process BY groups independently. PROC PRINT can report the number of observations in each BY group as well as the number of observations in all BY groups. Similarly, PROC PRINT can sum numeric variables in each BY group and across all BY groups.

You can use only one BY statement in each PROC step. When you use a BY statement, the procedure expects an input data set that is sorted by the order of the BY variables or one that has an appropriate index. If your input data set does not meet these criteria, then an error occurs. Either sort it with the SORT procedure or create an appropriate index on the BY variables.

Depending on the order of your data, you might need to use the NOTSORTED or DESCENDING option in the BY statement in the PROC step.

- For more information about the BY statement, see [SAS DATA Step Statements: Reference](#).
- For more information about PROC SORT, see [Chapter 55, "SORT Procedure," on page 2039](#).

- For more information about BY-group processing in CAS, see [“Grouping and Ordering” in SAS Cloud Analytic Services: DATA Step Programming](#).
- For more information about creating indexes, see [“INDEX CREATE Statement” on page 440](#).

Formatting BY-Variable Values

When a procedure is submitted with a BY statement, the following actions are taken with respect to processing of BY groups:

- 1 The procedure determines whether the data is sorted by the internal (unformatted) values of the BY variable(s).
- 2 The procedure determines whether a format has been applied to the BY variable(s). If the BY variable is numeric and has no user-applied format, then the BEST12. format is applied for the purpose of BY-group processing.
- 3 The procedure continues adding observations to the current BY group until both the internal and formatted values of the BY variable or variables change.

This process can have unexpected results if, for example, nonconsecutive internal BY values share the same formatted value. In this case, the formatted value is represented in different BY groups. Alternatively, if different consecutive internal BY values share the same formatted value, then these observations are grouped into the same BY group.

BY Variables That Have Different Lengths in Two Data Sets

When a procedure has a BY statement and two input data sets, one of the input data sets is called the primary data set and the other is called the secondary data set. The primary data set is usually, but not always, the DATA= data set. A BY statement always applies to the primary data set. The variables in the BY statement must appear in the primary data set.

Each procedure determines whether a BY statement applies to a secondary data set, and performs one of the following actions:

- The procedure might always apply the BY statement to the secondary data set. In this case, one or more variables in the BY statement must appear in the secondary data set.
- The procedure might never apply the BY statement to the secondary data set. In this case, the variables in the BY statement are not required in the secondary data set.
- The procedure might check whether the BY variables are in the secondary data set. If none of the BY variables are in the secondary data set, then BY processing does not occur for the secondary data set. If one or more of the BY variables are in the secondary data set, and they match the BY variables in the primary data set, then BY processing is done for the secondary data set. If some but not all BY variables are in the secondary data set, then the procedure might issue an error message and quit. Or, it might take some other action described in the documentation for that particular procedure.

If the BY statement is applied to the secondary data set, then each BY variable that exists on both the data sets must have the same type, character or numeric, in both data sets. The BY variables are required to have either the same formatted value or the same unformatted value. Formatted values match only if both the formatted lengths and the formatted values are the same. Unformatted values are not required to have the same length in order to match. The unformatted character values match if the unformatted values are the same after stripping the trailing blanks. The unformatted doubles match if they have the same value.

A secondary data set does not need to have all of the BY variables that are in the primary data set. A procedure can define a subset of the BY variables for the secondary data set. For example, if the primary data set has the BY variables A,B,C,D, then the procedure can define the following BY variables on the secondary data set:

- A
- A,B
- A,B,C
- A,B,C,D

If both the primary and secondary data sets have the same number of BY variables, and all the BY variables have the same byte lengths and format lengths, then either the unformatted values or the formatted values in the BY buffer (for all of the BY variables) have to match. If they do not match, then each variable is compared. The formatted values of each variable are compared first. The formatted lengths have to match, and the formatted values have to match. If the formatted lengths and values do not match, then the unformatted values are compared even if the byte lengths are different.

If corresponding character variable lengths differ, then the longer character variable can contain only trailing blanks for the extra characters. If the lengths of the character variables are different, then the values match as long as they are the same after stripping the trailing blanks. For example, 'ABCD' in the primary data set matches 'ABCD ' in the secondary data set. If the secondary data set contained 'ABCDEF', then they would not match.

Use the BY Statement with an ID Statement

PROC PRINT uses a special layout if all BY variables appear in the same order at the beginning of the ID statement. (See [“Example 9: Creating a Customized Layout with BY Groups and ID Variables”](#) on page 1485.)

Use the BY Statement with the NOBYLINE Option

If you use the BY statement with the SAS system option NOBYLINE, which suppresses the BY line that normally appears in output produced with BY-group processing, PROC PRINT always starts a new page for each BY group. This behavior ensures that if you create customized BY lines by putting BY-group information in the title and suppressing the default BY lines with NOBYLINE, the information in the titles matches the report on the pages.

Use a BY Variable When You Print Unsorted Data

If you specify a BY variable whose values are not sorted, SAS stops printing the data set when it processes the first unsorted group. A message is written to the SAS log.

Base SAS Procedures That Support the BY Statement

COMPARE	SORT (required)
CORR	STANDARD
FREQ	SUMMARY
MEANS	TABULATE
PRINT	TRANPOSE
RANK	UNIVARIATE
REPORT (nonwindowing environment only)	ODSTEXT
ODSLIST	ODSTABLE

Note: In the SORT procedure, the BY statement specifies how to sort the data. In the other procedures, the BY statement specifies how the data is currently sorted.

See Also

[“BY Statement” in SAS DATA Step Statements: Reference](#)

Example

Temporarily Dissociating a Format from a Variable

This example uses a BY statement in a PROC PRINT step. There is output for each value of the BY variable Year.

```
proc format;
  value $yrfmt  '1'='Freshman'
                '2'='Sophomore'
                '3'='Junior'
```

```

                                '4'='Senior';
run;
data debate;
    input Name $ Gender $   Year $   GPA   @@;
    format year $yrfmt.;
    datalines;
Capiccio m 1 3.598 Tucker    m 1 3.901
Bagwell   f 2 3.722 Berry    m 2 3.198
Metcalf   m 2 3.342 Gold      f 3 3.609
Gray      f 3 3.177 Syme      f 3 3.883
Baglione  f 4 4.000 Carr      m 4 3.750
Hall      m 4 3.574 Lewis     m 4 3.421
;

options nodate pageno=1 linesize=64
        pagesize=40;
proc print data=debate noobs;
    by year;
    title 'Printing of Team Members';
    title2 'by Year';
run;

```

Output

Printing of Team Members by Year

Year=Freshman

Name	Gender	GPA
Capiccio	m	3.598
Tucker	m	3.901

Year=Sophomore

Name	Gender	GPA
Bagwell	f	3.722
Berry	m	3.198
Metcalf	m	3.342

Year=Junior

Name	Gender	GPA
Gold	f	3.609
Gray	f	3.177
Syme	f	3.883

Year=Senior

Name	Gender	GPA
Baglione	f	4.000
Carr	m	3.750
Hall	m	3.574
Lewis	m	3.421

FORMAT Statement

Associates formats with variables.

Syntax

FORMAT *variable-1* <...*variable-n*> *format variable-1* <...*variable-n*> *format*;

Arguments

variable

names one or more variables for SAS to associate with a format. You must specify at least one *variable*.

Tip To disassociate a format from a variable, use the variable in a FORMAT statement without specifying a format in a DATA step or in PROC DATASETS. In a DATA step, place this FORMAT statement after the SET statement. See [“Removing a Format” in SAS DATA Step Statements: Reference](#). You can also use PROC DATASETS.

format

specifies the format that is listed for writing the values of the variables.

Tip Formats that are associated with variables by using a FORMAT statement behave like formats that are used with a colon modifier in a subsequent PUT statement. For more information about using a colon modifier, see [“PUT Statement: List” in SAS DATA Step Statements: Reference](#).

See [SAS Formats and Informats: Reference](#)

ID Statement

Identifies observations by using the formatted values of the variables that you list instead of by using observation numbers.

Examples: [“Example 8: Controlling the Layout of a Report with Many Variables” on page 1483](#)
[“Example 9: Creating a Customized Layout with BY Groups and ID Variables” on page 1485](#)

Syntax

ID *variable(s)*

< / **STYLE** <(location(s))>=<style-override(s)>>;

Required Argument

variable(s)

specifies one or more variables to print instead of the observation number at the beginning of each row of the report.

- Restriction** If the ID variables occupy so much space that no room remains on the line for at least one other variable, PROC PRINT writes a warning to the SAS log and does not treat all ID variables as ID variables.
- Interaction** If a variable in the ID statement also appears in the VAR statement, the output contains two columns for that variable.

Optional Argument

STYLE <(location(s))>=<style-override(s)>

specifies one or more style overrides to use for ID columns created with the ID statement.

You can specify a style override in two ways:

- Specify a style element. A style element is a collection of style attributes that apply to a particular part of the output for a SAS program.
- Specify a style attribute. A style attribute is a name-value pair that describes a single behavioral or visual aspect of a piece of output. This is the most specific method of changing the appearance of your output.

style-override has the following form:

style-element-name | [*style-attribute-name-1*=*style-attribute-value-1*
<*style-attribute-name-2*=*style-attribute-value-2* ...>]

- Restriction** Style specifications for the OBSHEADER location is not valid in the ID statement.
- Interaction** If the STYLE(HEADER)= option is specified in the ID statement and the STYLE(OBSHEADER)= is specified in the PROC PRINT statement, the style attributes that are specified for the ID statement take precedence over the style elements that are specified in the PROC PRINT statement. Then, the style attributes in the PROC PRINT statement STYLE(OBSHEADER)= option are merged with the style attributes in the ID statement STYLE(HEADER)= option to render the output for the ID column heading.
- Tip** To specify different style overrides for different ID columns, use a separate ID statement for each variable and add a different STYLE option to each ID statement.
- See** For information about the arguments of this option and how it is used, see the [STYLE= on page 1410](#) option in the PROC PRINT statement.

Details

Use the BY Statement with an ID Statement

PROC PRINT uses a special layout if all BY variables appear in the same order at the beginning of the ID statement. (See [“Example 9: Creating a Customized Layout with BY Groups and ID Variables”](#) on page 1485.)

LABEL Statement

Assigns descriptive labels to variables.

Syntax

LABEL *variable-1* = '*text-string*' <...*variable-n* = '*text-string*'>;

LABEL *variable-1* = '' <...*variable-n*=''>; | *variable-1* = <...*variable-n*=>;

Arguments

variable

specifies the variable that you want to label. You can specify labels for multiple variables in a single LABEL statement:

```
data test;
  L = 5;
  W = 10;
  label L = 'Length' W = 'Width';
run;
```

text-string

specifies a label of up to 256 bytes.

```
data test;
  L = 5;
  label L = 'Length';
run;
```

Restrictions If the label includes a semicolon (;) or an equal sign (=), you must enclose the label in either single or double quotation marks.

```
label N = 'Number = ';
```

If the label includes single quotation marks ('), you must enclose the label in double quotation marks.

```
label Name = "Owner's Last Name";
```

There is partial support for variable name substitution with the JAVAIMG device. When the label text for the variable exceeds

the space allowed, the variable name is used instead by procedures that are labeling an axis or a legend title. With the exception of the SAS/GRAPH GPLOT procedure, the variable name is not used when the JAVAIMG device is specified.

Notes

Quotation marks are not required unless the label includes a semicolon, quotation marks, or an equal sign.

The LABEL statement expects an equal sign (=) after the first variable. If any other character is found, an error occurs. The LABEL statement considers everything after the equal sign, following the first variable, as part of the label until the statement comes to another variable followed by an equal sign, regardless of quotation marks. In this example, the label for x is **this is x y 4 = this is y** because the next variable that is followed by an equal sign is z.

```
data test;
  x = 1;
  y = 1;
  z = 1;
  label x = 'this is x' y 4 = 'this is y' z = 'This is z';
run;
```

In this LABEL statement, the x variable has the same label:

```
label x = this is x y 4 = this is y z = This is z;
```

See

[“Character Constants and Character Variables” in SAS Programmer’s Guide: Essentials.](#)

”

removes a label from a variable. Enclose a single blank space in quotation marks to remove an existing label. You can remove labels from multiple variables in a single LABEL statement:

```
data test;
  L=5; W=10;
  label L = ' ' W = ' ';
run;
```

Alternatively, you can remove a label by specifying nothing after the equal sign:

```
data test;
  L=5; W=10;
  label L = W = ;
run;
```

PAGEBY Statement

Controls page ejects that occur before a page is full.

Requirement: BY statement

Example: [“Example 4: Creating Separate Sections of a Report for Groups of Observations” on page 1460](#)

Syntax

PAGEBY *BY-variable*;

Required Argument

BY-variable

identifies a variable appearing in the BY statement in the PROC PRINT step. If the value of the BY variable changes, or if the value of any BY variable that precedes it in the BY statement changes, PROC PRINT begins printing a new page.

Interaction If you use the BY statement with the SAS system option NOBYLINE, which suppresses the BY line that normally appears in output produced with BY-group processing, PROC PRINT always starts a new page for each BY group. This behavior ensures that if you create customized BY lines by putting BY-group information in the title and suppressing the default BY lines with NOBYLINE, the information in the titles matches the report on the pages. (See [“Creating Titles That Contain BY-Group Information” on page 55.](#))

SUM Statement

Totals values of numeric variables.

Tip: SAS includes checks to verify that the PROC PRINT output is accessible for the visually impaired. When you use the SUM statement and set the [ACCESSIBLECHECK system option](#), SAS verifies whether a label has been specified for both the SUMLABEL and the GRANDTOTAL_LABEL options in the PROC PRINT statement. If the output does not have labels for the summary and grand total values, SAS writes a message to the SAS log. For best practices about creating accessible output, see [Creating Accessible SAS Viya Platform Output Using ODS and ODS Graphics](#).

Examples: [“Example 5: Summing Numeric Variables with One BY Group” on page 1468](#)
[“Example 6: Summing Numeric Variables with Multiple BY Variables” on page 1473](#)
[“Example 7: Limiting the Number of Sums in a Report” on page 1478](#)
[“Example 9: Creating a Customized Layout with BY Groups and ID Variables” on page 1485](#)

Syntax

```
SUM variable(s)
</ STYLE <(location(s))>=<style-override(s)> >;
```

Required Argument

variable(s)

identifies the numeric variables to total in the report.

Optional Argument

STYLE <(location(s))>=<style-override(s)>

specifies one or more style overrides to use for cells containing sums that are created with the SUM statement.

You can specify a style override in two ways:

- Specify a style element. A style element is a collection of style attributes that apply to a particular part of the output for a SAS program.
- Specify a style attribute. A style attribute is a name-value pair that describes a single behavioral or visual aspect of a piece of output. This is the most specific method of changing the appearance of your output.

style-override has the following form:

```
style-element-name | [style-attribute-name-1=style-attribute-value-1
<style-attribute-name-2=style-attribute-value-2 ...>]
```

Tips To specify different style overrides for different cells reporting sums, use a separate SUM statement for each variable and add a different STYLE option to each SUM statement.

If the STYLE option is used in multiple SUM statements that affect the same location, the STYLE option in the last SUM statement is used.

See For information about the arguments of this option and how it is used, see the option [STYLE=](#) on page 1403 in the PROC PRINT statement.

Details

Use the SUM and BY Statements Together

When you use a SUM statement and a BY statement with one BY variable, PROC PRINT sums the SUM variables for each BY group that contains more than one observation and totals them over all BY groups. (See [“Example 5: Summing Numeric Variables with One BY Group”](#) on page 1468.)

When you use a SUM statement and a BY statement with multiple BY variables, PROC PRINT sums the SUM variables for each BY group that contains more than one observation, just as it does if you use only one BY variable. However, it

provides sums only for those BY variables whose values change when the BY group changes. (See [“Example 6: Summing Numeric Variables with Multiple BY Variables”](#) on page 1473.)

Note: When the value of a BY variable changes, the SAS System considers that the values of all variables listed after it in the BY statement also change.

SUMBY Statement

Limits the number of sums that appear in the report.

Requirement: BY statement

Example: [“Example 7: Limiting the Number of Sums in a Report”](#) on page 1478

Syntax

SUMBY *BY-variable*;

Required Argument

BY-variable

identifies a variable that appears in the BY statement in the PROC PRINT step. If the value of the BY variable changes, or if the value of any BY variable that precedes it in the BY statement changes, PROC PRINT prints the sums of all variables listed in the SUM statement.

Details

What Variables Are Summed?

If you use a SUM statement, PROC PRINT subtotals only the SUM variables. Otherwise, PROC PRINT subtotals all the numeric variables in the data set except for the variables listed in the ID and BY statements.

VAR Statement

Selects variables that appear in the report and determines their order.

Alias: VARIABLES

Tip: If you omit the VAR statement, PROC PRINT prints all variables in the data set.

Examples:

[“Example 2: Selecting Variables to Print” on page 1453](#)

[“Example 9: Creating a Customized Layout with BY Groups and ID Variables” on page 1485](#)

Syntax

VAR *variable(s)*

< / **STYLE** <(location(s))>=<style-override(s)> >;

Required Argument

variable(s)

identifies the variables to print. PROC PRINT prints the variables in the order in which you list them.

Interaction In the PROC PRINT output, variables that are listed in the ID statement precede variables that are listed in the VAR statement. If a variable in the ID statement also appears in the VAR statement, the output contains two columns for that variable.

Optional Argument

STYLE <(location(s))>=<style-override(s)>

specifies one or more style overrides to use for all columns that are created by a VAR statement.

You can specify a style override in two ways:

- Specify a style element. A style element is a collection of style attributes that apply to a particular part of the output for a SAS program.
- Specify a style attribute. A style attribute is a name-value pair that describes a single behavioral or visual aspect of a piece of output. This is the most specific method of changing the appearance of your output.

style-override has the following form:

```
style-element-name | [style-attribute-name-1=style-attribute-value-1
  <style-attribute-name-2=style-attribute-value-2 ...>]
```

Tip To specify different style overrides for different columns, use a separate VAR statement to create a column for each variable and add a different STYLE option to each VAR statement.

See For information about the arguments of this option and how it is used, see the option [STYLE= on page 1410](#) in the PROC PRINT statement.

WHERE Statement

Subsets the input data set by specifying certain conditions that each observation must meet before it is available for processing.

Syntax

WHERE *where-expression-1*
< *logical-operator where-expression-n*>;

Arguments

where-expression

is an arithmetic or logical expression that generally consists of a sequence of operands and operators.

Tips The operands and operators described in the next several sections are also valid for the WHERE= data set option.

You can specify multiple where-expressions.

logical-operator

can be AND, AND NOT, OR, or OR NOT.

Example

Procedures That Support the WHERE Statement

You can use the WHERE statement with any of the following Base SAS procedures that read a SAS data set:

COMPARE	REPORT
CORR	SORT
DATASETS (APPEND statement)	SQL
FREQ	STANDARD

MEANS or SUMMARY	TABULATE
PRINT	TRANSPOSE
RANK	UNIVARIATE

Details

- The COMPARE procedure and the APPEND statement in PROC DATASETS accept more than one input data set. For more information, see the documentation for the specific procedure.
- To subset the output data set, use the WHERE= data set option:

```
proc report data=debate nowd
              out=onlyfr(where=(year='1')) ;
run;
```

For more information about WHERE=, see [SAS DATA Step Statements: Reference](#).

Temporarily Dissociating a Format from a Variable

In this example, PROC PRINT prints only those observations that meet the condition of the WHERE expression.

```
options nodate pageno=1 linesize=64
              pagesize=40;

proc print data=debate noobs;
  where gpa>3.5;
  title 'Team Members with a GPA';
  title2 'Greater than 3.5';
run;
```

Output

**Team Members with a GPA
Greater than 3.5**

Name	Gender	Year	GPA
Capiccio	m	Freshman	3.598
Tucker	m	Freshman	3.901
Bagwell	f	Sophomore	3.722
Gold	f	Junior	3.609
Syme	f	Junior	3.883
Baglione	f	Senior	4.000
Carr	m	Senior	3.750
Hall	m	Senior	3.574

Usage: PRINT Procedure

Use ODS Styles with PROC PRINT

Using Styles with Base SAS Procedures

Most Base SAS procedures that support ODS use one or more table templates to produce output objects. These table templates include templates for table elements: columns, headers, and footers. Each table element can specify the use of one or more style elements for various parts of the output. These style elements cannot be specified within the syntax of the procedure, but you can use customized styles for the ODS destinations that you use. For more information about customizing tables and styles, see [“TEMPLATE Procedure: Creating a Style Template” in SAS Output Delivery System: Procedures Guide](#).

The Base SAS reporting procedures, PROC PRINT, PROC REPORT, and PROC TABULATE, enable you to quickly analyze your data and organize it into easy-to-read tables. You can use the STYLE= option with these procedure statements to modify the appearance of your report. The STYLE= option enables you to make

changes in sections of output without changing the default style for all of the output. You can customize specific sections of procedure output by specifying the `STYLE=` option in specific statements within the procedure.

The following program uses the `STYLE=` option to create the colors in the PROC PRINT output below. For the complete input data set, see [“EXPREV” on page 2514](#).

```
ods _all_ close;
ods html5 file="file-path/customStyle.html";
proc print data=exprev noobs sumlabel='Total' GRANDTOTAL_LABEL="Grand
Total"
    style(table)=[frame=box rules=groups]
    style(bysumline)=[background=red foreground=linen]
    style(grandtotal)=[foreground=green]
    style(header)=[font_style=italic background=orange];
  by sale_type order_date;
  sum price quantity;
  sumby sale_type;
  label sale_type='Sale Type' order_date='Sale Date';
  format price dollar10.2 cost dollar10.2;
  var Country / style(data)=[font_face=arial font_weight=bold
background=linen];
  var Price / style(data)=[font_style=italic background=yellow];
  var Cost / style(data)=[foreground=hgt. background=lightgreen];
  title 'Retail and Quantity Totals for Each Sale Type';
run;
ods _all_ close;
```

Output 40.1 Enhanced PROC PRINT Output**Retail and Quantity Totals for Each Sale Type****Sale Type=Catalog Sale Date=1/1/12**

<i>Country</i>	<i>Price</i>	<i>Cost</i>	<i>Quantity</i>
Puerto Rico	\$51.20	\$12.10	14
Aruba	\$123.70	\$59.00	30
Bahamas	\$113.40	\$28.45	8
Bermuda	\$41.00	\$9.25	7

Sale Type=Catalog Sale Date=1/2/12

<i>Country</i>	<i>Price</i>	<i>Cost</i>	<i>Quantity</i>
British Virgin Islands	\$40.20	\$20.20	11
Canada	\$11.80	\$5.00	100
Total	\$381.30		170

Sale Type=In Store Sale Date=1/1/12

<i>Country</i>	<i>Price</i>	<i>Cost</i>	<i>Quantity</i>
Virgin Islands (U.S.)	\$31.10	\$15.65	25

Sale Type=In Store Sale Date=1/2/12

<i>Country</i>	<i>Price</i>	<i>Cost</i>	<i>Quantity</i>
Belize	\$146.40	\$36.70	2
Cayman Islands	\$71.00	\$32.30	20
Total	\$248.50		47

Sale Type=Internet Sale Date=1/1/12

<i>Country</i>	<i>Price</i>	<i>Cost</i>	<i>Quantity</i>
Antarctica	\$92.60	\$20.70	2
Grand Total	\$722.40		219

Styles, Style Elements, and Style Attributes

Understanding Styles, Style Elements, and Style Attributes

The appearance of SAS output is controlled by ODS style templates (ODS styles). ODS styles are produced from compiled STYLE templates written in PROC TEMPLATE style syntax. An ODS style template is a collection of style elements that provides specific visual attributes for your SAS output.

- A style element is a named collection of style attributes that apply to a particular part of the output. Each area of ODS output has a style element name that is associated with it. The style element name specifies where the style attributes are applied. For example, a style element might contain instructions for the presentation of column headings or for the presentation of the data inside the cells. Style elements might also specify default colors and fonts for output that uses the style.
- A style attribute is a visual property, such as color, font properties, and line characteristics, that is defined in ODS with a reserved name and value. Style attributes are collectively referenced by a style element within a style template. Each *style attribute* specifies a value for one aspect of the presentation. For example, the BACKGROUND_COLOR= attribute specifies the color for the background of an HTML table or for a colored table in printed output. The FONTSTYLE= attribute specifies whether to use a Roman font or an italic font.

Note: Because styles control the presentation of the data, they have no effect on output objects that go to the DOCUMENT or OUTPUT destination.

Available styles are in the SASHELP.TMPLMST item store. You can display the list of available style templates by using the [LIST](#) statement in PROC TEMPLATE:

```
proc template;
  list styles / store=sashelp.tmplmst;
run;
```

For complete information about viewing ODS styles, see [“Viewing ODS Styles Supplied by SAS” in SAS Output Delivery System: Procedures Guide](#).

HTML5 output uses the HTML5Encore style template. To help you become familiar with styles, style elements, and style attributes, look at the relationship between them.

You can use the [SOURCE](#) statement in PROC TEMPLATE to display the structure of a style template. The following code prints the structure of the HTMLBlue style template to the SAS log:

```
proc template;
  source styles.HTMLBlue;
run;
```

The following figure illustrates the structure of a style. The figure shows the relationship between the style, the style elements, and the style attributes.

Figure 40.4 Diagram of the HtmlBlue Style

```

proc template;
  define style Styles.HtmlBlue; ← 1
    parent = styles.statistical;
    class GraphColors /
      'gblockheader' = cxcfd5de
      'gcphasebox' = cx989EA1
      'gphasebox' = cxDBE6F2
      'gczonec' = cxBECEE0
      'gzonec' = cxCCDCEE
      'gczoneb' = cxCCDCEE
      'gzoneb' = cxD7E5F3
      'gzonea' = cxE3EDF7
      'gconramp3cend' = cx9C1C00
      'gconramp3cneutral' = cx222222
      'gconramp3cstart' = cx0E36AC
      'gramp3cend' = cxD05B5B
      'gramp3cneutral' = cxFAFBFE
      'gramp3cstart' = cx667FA2
      'gcontrollim' = cxE6F2FF
      'gccontrollim' = cxBFC7D9
      'gruntest' = cxCAE3FF
      'gcruntest' = cxBF4D4D
      'gclipping' = cxFFFC6
      'gcclipping' = cxC1C100

...more style elements and style attributes...

class Header / ← 2
  bordercolor = cxB0B7BB ← 3
  backgroundcolor = cxEDF2F9 ← 3
  color = cx112277; ← 3
class Footer / ← 2
  bordercolor = cxB0B7BB ← 3
  backgroundcolor = cxEDF2F9 ← 3
  color = cx112277; ← 3
class RowHeader /
  bordercolor = cxB0B7BB
  backgroundcolor = cxEDF2F9
  color = cx112277;
class RowFooter /
  bordercolor = cxB0B7BB
  backgroundcolor = cxEDF2F9
  color = cx112277;
class Table /
  cellpadding = 5;
class Graph /
  attrpriority = "Color";
class GraphFit2 /
  linestyle = 1;
class GraphClipping /
  markersymbol = "circlefilled";
end;
run;
*** END OF TEXT *** ←

```

The following list corresponds to the numbered items in the preceding figure:

- 1 Styles.HtmlBlue is the *style*. Styles describe how to display presentation aspects (color, font, font size, and so on) of the SAS output. A style determines the overall appearance of the ODS documents that use it. The default style for HTML output is HtmlBlue. Each style consists of style elements.

You can create new styles with the “[DEFINE STYLE Statement](#)” in *SAS Output Delivery System: Procedures Guide*. New styles can be created independently or from an existing style. You can use “[PARENT= Statement](#)” in *SAS Output Delivery System: Procedures Guide* to create a new style from an existing style. For complete documentation about ODS styles, see “[Style Templates](#)” in *SAS Output Delivery System: Procedures Guide*.

- 2 Header and Footer are examples of *style elements*. A style element is a collection of style attributes that apply to a particular part of the output for a SAS program. For example, a style element might contain instructions for the presentation of column headings or for the presentation of the data inside table cells. Style elements might also specify default colors and fonts for output that uses the style. Style elements exist inside styles and consist of one or more style attributes. Style elements can be user-defined or supplied by SAS. User-defined style elements can be created by the “[STYLE Statement](#)” in *SAS Output Delivery System: Procedures Guide*.

Note: For a list of the default style elements used for HTML and markup languages and their inheritance, see “[Style Elements](#)” in *SAS Output Delivery System: Procedures Guide*.

- 3 BORDERCOLOR=, BACKGROUNDColor=, and COLOR= are examples of *style attributes*. Style attributes specify a value for one aspect of the area of the output that its style element applies to. For example, the COLOR= attribute specifies the value `cx112277` for the font color. For a list of style attributes supplied by SAS, see “[Style Attributes](#)” in *SAS Output Delivery System: Procedures Guide*.

Style attributes can be referenced with style references. See “[style-reference](#)” in *SAS Output Delivery System: Procedures Guide* for more information about style references.

The following table shows commonly used style attributes that you can set with the STYLE= option in PROC PRINT, PROC TABULATE, and PROC REPORT. Most of these attributes apply to parts of the table other than cells (for example, table borders and the lines between columns and rows). Note that not all attributes are valid in all destinations. For more information about these style attributes, their valid values, and their applicable destinations, see “[Style Attributes Tables](#)” in *SAS Output Delivery System: Procedures Guide*.

Table 40.2 Style Attributes for PROC REPORT, PROC TABULATE, and PROC PRINT

Attribute	PROC REPORT STATEMENT REPORT Area	PROC REPORT Areas: CALLDEF, COLUMN, HEADER, LINES, SUMMARY	PROC TABULATE STATEMENT TABLE	PROC TABULATE STATEMENTS VAR, CLASS, BOX, CLASSLEV, KEYWORD	PROC PRINT TABLE location	PROC PRINT: all locations other than TABLE
ASIS=	X	X		X		X

Attribute	PROC REPORT STATEMENT REPORT Area	PROC REPORT Areas: CALLDEF, COLUMN, HEADER, LINES, SUMMARY	PROC TABULATE STATEMENT TABLE	PROC TABULATE STATEMENTS VAR, CLASS, BOX, CLASSLEV, KEYWORD	PROC PRINT TABLE location	PROC PRINT: all locations other than TABLE
BACKGROUNDCOLOR=	X	X	X	X	X	X
BACKGROUNDIMAGE=	X	X	X	X	X	X
BORDERBOTTOMCOLOR=	X	X		X		
BORDERBOTTOMSTYLE=	X	X	X	X		
BORDERBOTTOMWIDTH=	X	X	X	X		
BORDERLEFTCOLOR=	X	X		X		
BORDERLEFTSTYLE=	X	X	X	X		
BORDERLEFTWIDTH=	X	X	X	X		
BORDERCOLOR=	X	X		X	X	X
BORDERCOLORDARK=	X	X	X	X	X	X
BORDERCOLORLIGHT=	X	X	X	X	X	X
BODERRIGHTCOLOR=	X	X		X		
BODERRIGHTSTYLE=	X	X	X	X		
BODERRIGHTWIDTH=	X	X	X	X		
BODERTOPCOLOR=	X	X		X		

Attribute	PROC REPORT STATEMENT REPORT Area	PROC REPORT Areas: CALLDEF, COLUMN, HEADER, LINES, SUMMARY	PROC TABULATE STATEMENT TABLE	PROC TABULATE STATEMENTS VAR, CLASS, BOX, CLASSLEV, KEYWORD	PROC PRINT TABLE location	PROC PRINT: all locations other than TABLE
BORDERTOPSTYLE=	X	X	X	X		
BORDERTOPWIDTH =	X	X	X	X		
BORDERWIDTH=	X	X	X	X	X	X
CELLPADDING=	X		X		X	
CELLSPACING=	X		X		X	
CELLWIDTH=	X	X	X	X		X
CLASS=	X	X	X	X	X	X
COLOR=	X	X	X			
FLYOVER=	X	X		X		X
FONT=	X	X	X	X	X	X
FONTFAMILY=	X	X	X	X	X	X
FONTSIZE=	X	X	X	X	X	X
FONTSTYLE=	X	X	X	X	X	X
FONTWEIGHT=	X	X	X	X	X	X
FONTWIDTH=	X	X	X	X		X
FRAME=	X		X		X	
HEIGHT=	X	X		X	X	X
HREFTARGET=		X		X		X
HTMLSTYLE=	X	X	X	X	X	
NOBREAKSPACE=2	X	X		X		X
OUTPUTWIDTH=	X	X	X	X	X	
POSTHTML=1	X	X	X	X	X	X

Attribute	PROC REPORT STATEMENT REPORT Area	PROC REPORT Areas: CALLDEF, COLUMN, HEADER, LINES, SUMMARY	PROC TABULATE STATEMENT TABLE	PROC TABULATE STATEMENTS VAR, CLASS, BOX, CLASSLEV, KEYWORD	PROC PRINT TABLE location	PROC PRINT: all locations other than TABLE
POSTIMAGE=	X	X	X	X	X	X
POSTTEXT= ¹	X	X	X	X	X	X
PREHTML= ¹	X	X	X	X	X	X
PREIMAGE=	X	X	X	X	X	X
PRETEXT= ¹	X	X	X	X	X	X
PROTECTSPECIALC HARS=		X		X	X	X
RULES=	X		X		X	
TAGATTR=	X	X	X	X	X	X
TEXTALIGN=	X	X	X	X	X	X
URL=		X		X		X
VERTICALALIGN=		X		X		X
WIDTH=	X	X	X	X	X	

- ¹ When you use these attributes in this location, they affect only the text that is specified with the PRETEXT=, POSTTEXT=, PREHTML=, and POSTHTML= attributes. To alter the foreground color or the font for the text that appears in the table, you must set the corresponding attribute in a location that affects the cells rather than the table. For complete documentation about style attributes and their values, see [“Style Attributes” in SAS Output Delivery System: Procedures Guide](#).
- ² To help prevent unexpected wrapping of long text strings when using PROC REPORT with the ODS RTF destination, set NOBREAKSPACE=OFF in a location that affects the LINE statement. The NOBREAKSPACE=OFF attribute must be set in the PROC REPORT code either on the LINE statement or on the PROC REPORT statement where style(lines) is specified.

Default Style Elements and Style Attributes for Table Regions

The following table lists the default style elements and style attributes for various locations of PROC PRINT output. The locations in this table correspond to the locations in [Table 40.82 on page 1410](#).

The table lists defaults for the most commonly used ODS destinations: HTML5, PDF, and RTF. Each destination has a default style template that is applied to all

output that is written to the destination. For complete documentation about the ODS destinations and their default styles, see [“Style Templates” in SAS Output Delivery System: Procedures Guide](#).

Note: The default style for ODS output in SAS Data and AI Studio is set by the theme that the session is actively using. Depending on the theme, the default is either Illuminate, Ignite, or HighContrast. For more information, see [“Customizing SAS Data and AI Studio” in SAS Data and AI Studio: User’s Guide](#).

- The default style for HTML5 output is HTMLEncore.
- The default style for PDF output is Pearl.
- The default style for RTF output is RTF.

Table 40.3 Default Style Elements and Style Attributes for Report Regions

Location	Style Element	HTML5 Style Attributes	PDF Style Attributes	RTF Style Attributes
BYLABEL	Byline	FONTFAMILY = "'Avenir Next for SAS', Arial, 'Albany AMT', Helvetica, Helv" FONTSIZE = 2 FONTWEIGHT = bold FONTSTYLE = roman COLOR = cx112277 BACKGROUNDCOLO R = cxd2f2f9	FONTFAMILY = "Arial, 'Albany AMT'" FONTSIZE = 8pt FONTWEIGHT = bold FONTSTYLE = roman COLOR = cx000000 BACKGROUNDCOLO R = cxffffff	FONTFAMILY = "'Times New Roman', 'Times Roman'" FONTSIZE = 11pt FONTWEIGHT = bold FONTSTYLE = roman COLOR = cx000000 BORDERWIDTH = NaN
GRANDTOT AL OBSHEADE R HEADER TOTAL	Header	FONTFAMILY = "'Avenir Next for SAS', Arial, 'Albany AMT', Helvetica, Helv" FONTSIZE = 2 FONTWEIGHT = bold FONTSTYLE = roman COLOR = cx112277 BACKGROUNDCOLO R = cxd2f2f9	FONTFAMILY = "Arial, 'Albany AMT'" FONTSIZE = 8pt FONTWEIGHT = bold FONTSTYLE = roman COLOR = cx000000 BACKGROUNDCOLO R = cxffffff BORDERWIDTH = NaN	FONTFAMILY = "'Times New Roman', 'Times Roman'" FONTSIZE = 11pt FONTWEIGHT = bold FONTSTYLE = roman COLOR = cx000000 BACKGROUNDCOLO R = cxbbbbbb
N	Linecontent	FONTFAMILY = "'Avenir Next for SAS', Arial, 'Albany AMT', Helvetica, Helv" FONTSIZE = 2	FONTFAMILY = "Arial, 'Albany AMT'" FONTSIZE = 8pt FONTWEIGHT =medium	FONTFAMILY = "'Times New Roman', 'Times Roman'" FONTSIZE = 10pt

Location	Style Element	HTML5 Style Attributes	PDF Style Attributes	RTF Style Attributes
		FONTWEIGHT = medium FONTSTYLE = roman COLOR = cx112277 BACKGROUNDCOLOR = cxfafbfe	FONTSTYLE = roman COLOR = cx000000 BACKGROUNDCOLOR = cxffffff BORDERWIDTH = NaN	FONTWEIGHT = medium FONTSTYLE = roman COLOR = cx000000
OBS	Rowheader	FONTFAMILY = "Avenir Next for SAS', Arial, 'Albany AMT', Helvetica, Helv" FONTSIZE = 2 FONTWEIGHT = bold FONTSTYLE = roman COLOR = cx112277 BACKGROUNDCOLOR = cxedf2f9	FONTFAMILY = "Arial, 'Albany AMT'" FONTSIZE = 8pt FONTWEIGHT = bold FONTSTYLE = roman COLOR = cx000000 BACKGROUNDCOLOR = cxffffff BORDERWIDTH = NaN	FONTFAMILY = "Times New Roman', 'Times Roman'" FONTSIZE = 11pt FONTWEIGHT = bold FONTSTYLE = roman COLOR = cx000000 BACKGROUNDCOLOR = cxbbbbbb
DATA	Data	FONTFAMILY = "Avenir Next for SAS', Arial, 'Albany AMT', Helvetica, Helv" FONTSIZE = 2 FONTWEIGHT = medium FONTSTYLE = roman BACKGROUNDCOLOR = cxffffff	FONTFAMILY = "Albany AMT', Albany" FONTSIZE = 8pt FONTWEIGHT = medium FONTSTYLE = roman COLOR = cx000000 BORDERWIDTH = NaN COLOR = cx000000	FONTFAMILY = "Times New Roman', 'Times Roman'" FONTSIZE = 10pt FONTWEIGHT = medium FONTSTYLE = roman COLOR = cx000000
Titles	SystemTitle	FONTFAMILY = "Avenir Next for SAS', Arial, 'Albany AMT', Helvetica, Helv" FONTSIZE = 3 FONTWEIGHT = bold FONTSTYLE = roman BACKGROUNDCOLOR = cxfafbfe	FONTFAMILY = "Albany AMT', Albany" FONTSIZE = 11pt FONTWEIGHT = bold FONTSTYLE = roman COLOR = cx000000 BACKGROUNDCOLOR = cxffffff	FONTFAMILY = "Times New Roman', 'Times Roman'" FONTSIZE = 13pt FONTWEIGHT = bold FONTSTYLE = italic COLOR = cx000000

Location	Style Element	HTML5 Style Attributes	PDF Style Attributes	RTF Style Attributes
Footnotes	Footers	FONTFAMILY = "'Avenir Next for SAS', Arial, 'Albany AMT', Helvetica, Helv" FONTSIZE = 2 FONTWEIGHT = medium FONTSTYLE = roman BACKGROUNDCOLO R = cxffffff	FONTFAMILY = "'Albany AMT', Albany" FONTSIZE = 11pt FONTWEIGHT = bold FONTSTYLE = roman COLOR = cx000000 BACKGROUNDCOLO R = cxffffff	FONTFAMILY = "'Times New Roman', 'Times Roman'" FONTSIZE = 13pt FONTWEIGHT = bold FONTSTYLE = italic COLOR = cx000000

Error Processing in the PRINT Procedure Output

If an error occurs in the PRINT procedure or if the procedure is halted, output might be created for the observations that were processed until the error. SAS writes a message to the SAS log and ends the PRINT procedure.

Results: PRINT Procedure

About PROC PRINT Output

By default, PROC PRINT produces an HTML5 report. The PRINT procedure statements, PROC PRINT, BY, PAGEBY, SUMBY, ID, SUM, and VAR control the content of the report. The options for each statement control the appearance of the report.

To change the ODS destination for the report, use ODS statements before the PROC PRINT statement. If you do not want HTML5 output, be sure to close the ODS HTML5 destination before you run the procedure. For more information about using ODS, see the [SAS Output Delivery System: User's Guide](#).

See the [PRINT procedure examples on page 1453](#) for a sampling of the types of reports that the procedure produces.

Page Layout for HTML, the Default ODS Destination

A page of ODS HTML5 output is not limited in width or length. Therefore, each observation in a table is printed on a single line and all observations that are specified to print by the report appear on a single page of HTML5 output.

Each time that PROC PRINT runs, by default, SAS adds a page break after the output. A page break is rendered by separating output with a horizontal rule. For more information, see [“ODS HTML5 Statement” in SAS Output Delivery System: User’s Guide](#).

Page Layout for Limited Page Sizes

Observations

PROC PRINT uses an identical layout for all observations on a page for ODS destinations that produce output whose page size is limited in width and length. Some of these ODS destinations are RTF, PDF, and LISTING. First, it attempts to print observations on a single line, as shown in the following figure.

Figure 40.5 *Printing Observations on a Single Line*

Obs	Va r_1	Va r_2	Va r_3	1
1	~~~~	~~~~	~~~~	
2	~~~~	~~~~	~~~~	
3	~~~~	~~~~	~~~~	
4	~~~~	~~~~	~~~~	
5	~~~~	~~~~	~~~~	
6	~~~~	~~~~	~~~~	

If PROC PRINT cannot fit all the variables on a single line, it splits the observations into two or more sections and prints the observation number or the ID variables at the beginning of each line. For example, in the following figure, PROC PRINT prints the values for the first three variables in the first section of each page and the values for the second three variables in the second section of each page.

Figure 40.6 Splitting Observations into Multiple Sections on One Page

1			
Obs	Var_1	Var_2	Var_3
1	~~~~	~~~~	~~~~
2	~~~~	~~~~	~~~~
3	~~~~	~~~~	~~~~
Obs	Var_4	Var_5	Var_6
1	~~~~	~~~~	~~~~
2	~~~~	~~~~	~~~~
3	~~~~	~~~~	~~~~

2			
Var_2	Var_3		
~~	~~~~		
~~	~~~~		
~~	~~~~		

Obs	Var_4	Var_5	Var_6
4	~~~~	~~~~	~~~~
5	~~~~	~~~~	~~~~
6	~~~~	~~~~	~~~~

If PROC PRINT cannot fit all the variables on one page, the procedure prints subsequent pages with the same observations until it has printed all the variables. For example, in the following figure, PROC PRINT uses the first two pages to print values for the first three observations and the second two pages to print values for the rest of the observations.

Figure 40.7 Splitting Observations across Multiple Pages

1			
Obs	Var_1	Var_2	Var_3
1	~~~~	~~~~	~~~~
2	~~~~	~~~~	~~~~
3	~~~~	~~~~	~~~~
Obs	Var_4	Var_5	Var_6
1	~~~~	~~~~	~~~~
2	~~~~	~~~~	~~~~
3	~~~~	~~~~	~~~~

2			
Obs	Var_7	Var_8	Var_9
1	~~~~	~~~~	~~~~
2	~~~~	~~~~	~~~~
3	~~~~	~~~~	~~~~
Obs	Var_10	Var_11	Var_12
1	~~~~	~~~~	~~~~
2	~~~~	~~~~	~~~~
3	~~~~	~~~~	~~~~

3			
Obs	Var_1	Var_2	Var_3
4	~~~~	~~~~	~~~~
5	~~~~	~~~~	~~~~
6	~~~~	~~~~	~~~~
Obs	Var_4	Var_5	Var_6
4	~~~~	~~~~	~~~~
5	~~~~	~~~~	~~~~
6	~~~~	~~~~	~~~~

4			
Obs	Var_7	Var_8	Var_9
4	~~~~	~~~~	~~~~
5	~~~~	~~~~	~~~~
6	~~~~	~~~~	~~~~
Obs	Var_10	Var_11	Var_12
4	~~~~	~~~~	~~~~
5	~~~~	~~~~	~~~~
6	~~~~	~~~~	~~~~

Column Headings

The amount of spacing specifies whether PROC PRINT prints column headings horizontally or vertically. [Figure 40.29 on page 1448](#), [Figure 40.30 on page 1449](#), and [Figure 40.31 on page 1449](#) all illustrate horizontal headings. The following figure illustrates vertical headings.

Figure 40.8 Using Vertical Headings

	V	V	V	1
	a	a	a	
0	r	r	r	
b	—	—	—	
s	1	2	3	
1	~~~~	~~~~	~~~~	
2	~~~~	~~~~	~~~~	
3	~~~~	~~~~	~~~~	
4	~~~~	~~~~	~~~~	
5	~~~~	~~~~	~~~~	
6	~~~~	~~~~	~~~~	

Note: If you use LABEL and at least one variable has a label, PROC PRINT prints all column headings horizontally unless you specify HEADING=VERTICAL.

Column Width

By default, PROC PRINT uses a variable's formatted width as the column width. If the variable does not have a format that explicitly specifies a field width, PROC PRINT uses the widest data value for that variable on that page as the column width.

If the formatted value of a character variable or the data width of an unformatted character variable exceeds the line size minus the length of all the ID variables, PROC PRINT might truncate the value. Consider the following situation:

- The line size is 80.
- IdNumber is a character variable with a length of 10. It is used as an ID variable.
- State is a character variable with a length of 2. It is used as an ID variable.
- Comment is a character variable with a length of 200.

When PROC PRINT prints these three variables on a line, it uses 14 print positions for the two ID variables and the space after each one. This arrangement leaves 80–14, or 66, print positions for COMMENT. Longer values of COMMENT are truncated.

Note: Column width is affected not only by variable width but also by the length of column headings. Long column headings might lessen the usefulness of WIDTH=.

Examples: PRINT Procedure

Example 1: Print a CAS Table

Features: PROC PRINT DATA=CAS-table
CAS language elements:
CAS statement
LIBNAME statement for the CAS engine
PROC CASUTIL
PROC MDSUMMARY

Data set: Sashelp.cars

Details

This example demonstrates the following tasks:

- establishes a CAS session
- associates the Mycas libref with the CAS engine and the CAS session
- creates the CAS table mycas.cars
- uses PROC MDSUMMARY to summarize the cars data
- prints 15 rows of the summarized CAS table

Program: Run in the SAS Compute Server

```
options cashost="cloud.example.com" casport=5555;  
cas mysess sessopts=(caslib='casuser');  
libname mycas cas sessref=mysess;  
  
proc casutil outcaslib="casuser";  
  load data=sashelp.cars replace;  
run;
```

```

proc mdsummary data=mycas.cars;
  var mpg_highway;
  groupby origin type / out=mycas.mpghw_sum;
run;

options obs=15;
proc print data=mycas.mpghw_sum;
  var origin type _mean_;
  title "Average Highway Mileages";
run;

```

Program Description

Start the CAS server, set up the CAS session, create a libref for the CAS engine, and connect the engine to the CAS session. The OPTIONS statement connects SAS to the CAS server. The CAS statement creates the Mysess session using the CASUSER caslib. The LIBNAME statement creates the Mycas libref for the CAS engine, which uses the Mysess CAS session.

```

options cashost="cloud.example.com" casport=5555;
cas mysess sessopts=(caslib='casuser');
libname mycas cas sessref=mysess;

```

Load the table Sashelp.cars into the caslib Casuser. The OUTCASLIB= option names the caslib to where the table is loaded. Use the LOAD statement to load the table from Sashelp.cars. The REPLACE option replaces the table and names the table to load.

```

proc casutil outcaslib="casuser";
  load data=sashelp.cars replace;
run;

```

Summarize the data using PROC MDSUMMARY. The VAR statement specifies the analysis variable to order the results. The GROUPBY statement creates BY groups and saves the output to the table Mycas.mpghw_sum.

```

proc mdsummary data=mycas.cars;
  var mpg_highway;
  groupby origin type / out=mycas.mpghw_sum;
run;

```

Print the first 15 rows of the summary results. With OBS=15, PROC PRINT prints only 15 rows of the CAS table. The VAR statement limits the output table to three columns, Origin, Type, and _Mean_.

```

options obs=15;
proc print data=mycas.mpghw_sum;
  var origin type _mean_;
  title "Average Highway Mileages";
run;

```

Average Highway Mileages

Obs	Origin	Type	_Mean_
1	Asia	Hybrid	56
2	Asia	SUV	21.68
3	Asia	Sedan	29.968085106
4	Asia	Sports	26.647058824
5	Asia	Truck	22
6	Asia	Wagon	28.181818182
7	Europe	SUV	18.7
8	Europe	Sedan	27.115384615
9	Europe	Sports	25.130434783
10	Europe	Wagon	26.583333333
11	USA	SUV	20.04
12	USA	Sedan	28.544444444
13	USA	Sports	24.222222222
14	USA	Truck	20.5
15	USA	Wagon	29.714285714

Example 2: Selecting Variables to Print

Features:

- PROC PRINT statement options
 - BLANKLINE
 - DOUBLE
 - STYLE
- VAR statement
- DATA step
- FOOTNOTE statement
- ODS HTML5 statement
- OPTIONS statement
- TITLE statement

Data set: [EXPREV](#)

ODS
destination: HTML5

Details

This example demonstrates the following tasks:

- selects three variables for the reports
- uses variable labels as column headings

- creates a stylized HTML5 report

Program: Creating an HTML5 Report

```
options obs=10;

proc print data=exprev;
    var country price sale_type;
    title 'Monthly Price Per Unit and Sale Type for Each Country';
    footnote '*prices in USD';
run;
```

Program Description

HTML5 is the default destination when SAS opens in SAS Data and AI Studio.

Set the OBS= system option to process 10 observations.

```
options obs=10;
```

Print the output The VAR statement specifies the variables to print.

```
proc print data=exprev;
    var country price sale_type;
    title 'Monthly Price Per Unit and Sale Type for Each Country';
    footnote '*prices in USD';
run;
```

Output: HTML5

Output 40.2 *Selecting Variables: Default HTML5 Output*

Monthly Price Per Unit and Sale Type for Each Country

Obs	Country	Price	Sale_Type
1	Antarctica	92.6	Internet
2	Puerto Rico	51.2	Catalog
3	Virgin Islands (U.S.)	31.1	In Store
4	Aruba	123.7	Catalog
5	Bahamas	113.4	Catalog
6	Bermuda	41.0	Catalog
7	Belize	146.4	In Store
8	British Virgin Islands	40.2	Catalog
9	Canada	11.8	Catalog
10	Cayman Islands	71.0	In Store

*prices in USD

Program: Create an HTML5 Report with the STYLE and BLANKLINE Options

```
options obs=5;

ods html5 file='your_file_styles.html';

proc print data=exprev
  style(header)={fontstyle=italic color= green}
  style(obs)={backgroundcolor=#a8a44ff8a color=blue}
  blankline=(count= 1 style={backgroundcolor=cx456789});
  var country price sale_type;
  title 'Monthly Price Per Unit and Sale Type for Each Country';
  footnote '*prices in USD';
run;
```

Program Description

You can go a step further and add more formatting to your HTML5 output. The following example uses the STYLE option to add shading and spacing to your HTML5 report.

```
options obs=5;

ods html5 file='your_file_styles.html';
```

Create stylized HTML5 output. The first STYLE option specifies that the column headings are written in green italic font. The second STYLE option specifies that the observation number column has a background color of the RGB color a8a44ff8a and a text color of blue. The BLANKLINE option specifies to add a blank line between each observation and use a background color of the CMYK color cx456789. Because a style has not been defined for the OBSHEADER location, the Obs column heading in the output uses the default style color and not green.

```
proc print data=exprev
  style(header)={fontstyle=italic color= green}
  style(obs)={backgroundcolor=#a8a44ff8a color=blue}
  blankline=(count= 1 style={backgroundcolor=cx456789});
  var country price sale_type;
  title 'Monthly Price Per Unit and Sale Type for Each Country';
  footnote '*prices in USD';
run;
```

Output: HTML5 Output with Styles

Output 40.3 Selecting Variables: HTML5 Output Using Styles

Monthly Price Per Unit and Sale Type for Each Country

Obs	Country	Price	Sale_Type
1	Antarctica	92.6	Internet
2	Puerto Rico	51.2	Catalog
3	Virgin Islands (U.S.)	31.1	In Store
4	Aruba	123.7	Catalog
5	Bahamas	113.4	Catalog

*prices in USD

Example 3: Customizing Text in Column Headings

Features: PROC PRINT statement options

N

OBS=

VAR statement option

STYLE

LABEL statement

ODS PDF statement

FORMAT statement

TITLE statement

Data set: [EXPREV](#)

ODS

destination:

PDF

Details

This example demonstrates the following tasks:

- adds background color to the column headings for variables in PDF output
- customizes the column heading for the column that identifies observations by number
- shows the number of observations in the report

- writes the values of the variable Price with dollar signs and periods

Program: Creating a PDF Report

```
options nodate pageno=1 linesize=80 pagesize=30 obs=10;
ods pdf file='your_file.pdf';

proc print data=expres n obs='Observation Number';

    var country sale_type price;
        label country='Country Name'
              sale_type='Order Type'
              price='Price Per Unit in USD';
        format price dollar10.2;
        title 'Order Type and Price Per Unit in Each Country';
run;
ods pdf close;
```

Program Description

You can easily create PDF output by adding a few ODS statements, as shown in the following example.

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. The PAGENO= option specifies the starting page number. The LINESIZE= option specifies the output line length, and the PAGESIZE= option specifies the number of lines on an output page. The OBS= option specifies the number of observations to be displayed.

```
options nodate pageno=1 linesize=80 pagesize=30 obs=10;
```

Create PDF output and specify the file to store the output in. The ODS PDF statement opens the PDF destination and creates PDF output. The FILE= argument specifies the external file that contains the PDF output.

```
ods pdf file='your_file.pdf';
```

Set the procedure options. The N option prints the number of observations at the end of the report. OBS= specifies the column heading for the column that identifies each observation by number.

```
proc print data=expres n obs='Observation Number';
```

Process the variables in the data set. The VAR statement specifies the variables to print. The LABEL statement creates text to print in place of the variable names. The FORMAT statement specifies to format the price variables using the DOLLARw. format. The TITLE statement creates a title for the report.

```
var country sale_type price;
    label country='Country Name'
          sale_type='Order Type'
```

```

        price='Price Per Unit in USD';
        format price dollar10.2;
        title 'Order Type and Price Per Unit in Each Country';
run;

```

Close the PDF destination. The ODS PDF CLOSE statement closes the PDF destination.

```
ods pdf close;
```

Output: PDF

Output 40.4 Customizing Column Heading: Default PDF Output

Order Type and Price Per Unit in Each Country

Observation Number	Country	Sale_Type	Price
1	Antarctica	Internet	\$92.60
2	Puerto Rico	Catalog	\$51.20
3	Virgin Islands (U.S.)	In Store	\$31.10
4	Aruba	Catalog	\$123.70
5	Bahamas	Catalog	\$113.40
6	Bermuda	Catalog	\$41.00
7	Belize	In Store	\$146.40
8	British Virgin Islands	Catalog	\$40.20
9	Canada	Catalog	\$11.80
10	Cayman Islands	In Store	\$71.00
N = 10			

Program: Creating a PDF Report with the STYLE Option

```

options obs=10;
ods pdf file='your_file.pdf';
proc print data=exprev n obs='Observation Number'
    style(n)={backgroundcolor=light blue fontstyle=italic}
    style(header obs obsheader)={backgroundcolor=light yellow
color=blue
                                fontstyle=italic};
    style(data)={backgroundcolor=very light blue}

```



```

var country sale_type price / style(data)=[backgroundcolor=very
light blue];
label country='Country Name'
      sale_type='Order Type'
      price='Price Per Unit in USD';
format price dollar10.2;
run;

title 'Order Type and Price Per Unit in Each Country';

ods pdf close;

```

Program Description

The OBS= system option specifies to process 10 observations.

```

options obs=10;

ods pdf file='your_file.pdf';

```

Create stylized PDF output. The first STYLE option specifies that the background color of the cell containing the value for N be changed to light blue and that the font style be changed to italic. The second STYLE option specifies that the background color of the observation column, the observation header, and the other variable's headers be changed to a light yellow, the text color is changed to blue, and the font style is changed to italic.

```

proc print data=exprev n obs='Observation Number'
      style(n)={backgroundcolor=light blue fontstyle=italic}
      style(header obs obsheader)={backgroundcolor=light yellow
color=blue
                                fontstyle=italic};
      style(data)={backgroundcolor=very light blue}

```

Create stylized PDF output. The STYLE option changes the color of the cells containing data to a very light blue.

```

var country sale_type price / style(data)=[backgroundcolor=very
light blue];
label country='Country Name'
      sale_type='Order Type'
      price='Price Per Unit in USD';
format price dollar10.2;
run;

title 'Order Type and Price Per Unit in Each Country';

```

Close the PDF destination. The ODS PDF CLOSE statement closes the PDF destination.

```

ods pdf close;

```

Output: PDF Report with Styles

Output 40.5 Customizing Column Headings: PDF Using Styles

Observation Number	Country	Sale_Type	Price
1	Antarctica	Internet	\$92.60
2	Puerto Rico	Catalog	\$51.20
3	Virgin Islands (U.S.)	In Store	\$31.10
4	Aruba	Catalog	\$123.70
5	Bahamas	Catalog	\$113.40
6	Bermuda	Catalog	\$41.00
7	Belize	In Store	\$146.40
8	British Virgin Islands	Catalog	\$40.20
9	Canada	Catalog	\$11.80
10	Cayman Islands	In Store	\$71.00
N = 10			

Example 4: Creating Separate Sections of a Report for Groups of Observations

Features:	PROC PRINT statement options LABEL N= NOOBS STYLE BY statement PAGEBY statement SORT procedure FORMAT statement LABEL statement ODS WORD statement TITLE statement
Data set:	EXPREV
ODS destinations:	HTML5, WORD

Details

This example demonstrates the following:

- suppresses the printing of observation numbers at the beginning of each row

- presents the data for each sale type in a separate section of the report
- creates a default HTML5 report
- creates default and stylized WORD reports

Program: Creating an HTML5 Report

```
options obs=10;

proc sort data=exprev;
  by sale_type order_date quantity;
run;

proc print data=exprev n='Number of observations for the month: '
  noobs label;

  var quantity cost price;

  by sale_type order_date;
  pageby order_date;

  label sale_type='Order Type' order_date='Order Date';

  format price dollar7.2 cost dollar7.2;
  title 'Prices and Cost Grouped by Date and Order Type';
  title2 'in USD';
run;

proc options option=bufno define;
run;
```

Program Description

The HTML5 destination is open by default. No ODS HTML5 statement is needed.

The OBS= system option specifies to process 10 observations.

```
options obs=10;
```

Sort the EXPREV data set. PROC SORT sorts the observations by Sale_Type, Order_Date, and Quantity.

```
proc sort data=exprev;
  by sale_type order_date quantity;
run;
```

Print the report, specify the total number of observations in each BY group, and suppress the printing of observation numbers. N= prints the number of observations in a BY group at the end of that BY group. The explanatory text that the N= option provides precedes the number. NOOBS suppresses the printing of observation numbers at the beginning of the rows. LABEL uses the variables' labels as column headings.

```
proc print data=exprev n='Number of observations for the month: '
  noobs label;
```

Specify the variables to include in the report. The VAR statement creates columns for Quantity, Cost, and Price, in that order.

```
var quantity cost price;
```

Create a separate section for each order type and specify page breaks for each BY group of Order_Date. The BY statement produces a separate section of the report for each BY group and prints a heading above each one. The PAGEBY statement starts a new page each time the value of Order_Date changes.

```
by sale_type order_date;  
pageby order_date;
```

Establish the column headings. The LABEL statement associates labels with the variables Sale_Type and Order_Date for the duration of the PROC PRINT step. When you use the LABEL option in the PROC PRINT statement, the procedure uses labels for column headings.

```
label sale_type='Order Type' order_date='Order Date';
```

Format the columns that contain numbers and specify a title and footnote. The FORMAT statement assigns a format to Price and Cost for this report. The TITLE statement specifies a title. The TITLE2 statement specifies a second title.

```
format price dollar7.2 cost dollar7.2;  
title 'Prices and Cost Grouped by Date and Order Type';  
title2 'in USD';  
run;  
  
proc options option=bufno define;  
run;
```

Output: HTML5

Output 40.6 *Creating Separate Sections of a Report for Groups of Observations: HTML5 Output*

Prices and Cost Grouped by Date and Order Type in USD

Order Type=Catalog Order Date=1/1/18

Quantity	Cost	Price
7	\$9.25	\$41.00
8	\$28.45	\$113.40
14	\$12.10	\$51.20
30	\$59.00	\$123.70
Number of observations for the month: 4		

Prices and Cost Grouped by Date and Order Type in USD

Order Type=Catalog Order Date=1/2/18

Quantity	Cost	Price
11	\$20.20	\$40.20
100	\$5.00	\$11.80
Number of observations for the month: 2		

Prices and Cost Grouped by Date and Order Type in USD

Order Type=In Store Order Date=1/1/18

Quantity	Cost	Price
25	\$15.65	\$31.10
Number of observations for the month: 1		

Prices and Cost Grouped by Date and Order Type in USD

Order Type=In Store Order Date=1/2/18

Quantity	Cost	Price
2	\$36.70	\$146.40
20	\$32.30	\$71.00
Number of observations for the month: 2		

Prices and Cost Grouped by Date and Order Type in USD

Order Type=Internet Order Date=1/1/18

Quantity	Cost	Price
2	\$20.70	\$92.60
Number of observations for the month: 1		

Program: Creating a Word Report

```
options obs=10;
ods word file='your_file.docx' startpage=no;
proc sort data=expres;
  by sale_type order_date quantity;
run;
proc print data=expres n='Number of observations for each order type:'
  noobs label;
  var quantity cost price;
  by sale_type order_date;
  pageby order_date;
  label sale_type='Order Type' order_date='Order Date';
  format price dollar7.2 cost dollar7.2;
  title 'Price and Cost Grouped by Date and Order Type';
  title2 'in USD';
run;
ods word close;
```

Program Description

The OBS= system option specifies to process 10 observations.

```
options obs=10;
```

Create output for Microsoft Word and specify the file to store the output in. The ODS WORD statement opens the WORD destination and creates output formatted for Microsoft Word. The FILE= option specifies the external file that contains the WORD output. The STARTPAGE=NO option specifies that no new pages be inserted explicitly at the start of each BY group.

```
ods word file='your_file.docx' startpage=no;

proc sort data=exprev;
  by sale_type order_date quantity;
run;

proc print data=exprev n='Number of observations for each order type:'
  noobs label;
  var quantity cost price;
  by sale_type order_date;
  pageby order_date;
  label sale_type='Order Type' order_date='Order Date';
  format price dollar7.2 cost dollar7.2;
  title 'Price and Cost Grouped by Date and Order Type';
  title2 'in USD';
run;
```

Close the WORD destination. The ODS WORD CLOSE statement closes the WORD destination.

```
ods word close;
```

Output: WORD

Output 40.7 *Creating Separate Sections of a Report for Groups of Observations:
Default WORD Output*

Price and Cost Grouped by Date and Order Type in USD

Order Type=Catalog Order Date=1/1/18

Quantity	Cost	Price
7	\$9.25	\$41.00
8	\$28.45	\$113.40
14	\$12.10	\$51.20
30	\$59.00	\$123.70
Number of observations for each order type:4		

Order Type=Catalog Order Date=1/2/18

Quantity	Cost	Price
11	\$20.20	\$40.20
100	\$5.00	\$11.80
Number of observations for each order type:2		

Order Type=In Store Order Date=1/1/18

Quantity	Cost	Price
25	\$15.65	\$31.10
Number of observations for each order type:1		

Order Type=In Store Order Date=1/2/18

Quantity	Cost	Price
2	\$36.70	\$146.40
20	\$32.30	\$71.00
Number of observations for each order type:2		

Order Type=Internet Order Date=1/1/18

Quantity	Cost	Price
2	\$20.70	\$92.60
Number of observations for each order type:1		

Program: Creating a Word Report with the STYLE Option

```
options obs=10;
ods word file='your_file.docx' startpage=no;
proc sort data=expres;
  by sale_type order_date quantity;
```



```

run;

proc print data=exprev n='Number of observations for the month: '
           noobs label style(N)={backgroundcolor=very light gray};
  var quantity / style(header)=[backgroundcolor=light yellow];
  var cost / style(header)=[backgroundcolor=light blue foreground =
white];
  var price / style(header)=[backgroundcolor=light green];
  by sale_type order_date;
  pageby order_date;
  label sale_type='Order Type' order_date='Order Date';
  format price dollar7.2 cost dollar7.2;

title 'Prices and Cost Grouped by Date and Order Type';
title2 '*prices in USD';
run;

ods word close;

```

Program Description

The OBS= system option specifies to process 10 observations.

```

options obs=10;

ods word file='your_file.docx' startpage=no;

proc sort data=exprev;
  by sale_type order_date quantity;
run;

```

Create a stylized WORD report. The first STYLE option specifies that the background color of the cell containing the number of observations be changed to light gray. The second STYLE option specifies that the background color of the column heading for the variable Quantity be changed to light yellow. The third STYLE option specifies that the background color of the column heading for the variable Cost be changed to light blue and the font color be changed to white. The fourth STYLE option specifies that the background color of the column heading for the variable Price be changed to light green.

```

proc print data=exprev n='Number of observations for the month: '
           noobs label style(N)={backgroundcolor=very light gray};
  var quantity / style(header)=[backgroundcolor=light yellow];
  var cost / style(header)=[backgroundcolor=light blue foreground =
white];
  var price / style(header)=[backgroundcolor=light green];
  by sale_type order_date;
  pageby order_date;
  label sale_type='Order Type' order_date='Order Date';
  format price dollar7.2 cost dollar7.2;

title 'Prices and Cost Grouped by Date and Order Type';
title2 '*prices in USD';
run;

ods word close;

```

Output: Word with Styles

Output 40.8 *Creating Separate Sections of a Report for Groups of Observations:
WORD Output Using Styles*

Prices and Cost Grouped by Date and Order Type *prices in USD

Order Type=Catalog Order Date=1/1/18

Quantity	Cost	Price
7	\$9.25	\$41.00
8	\$28.45	\$113.40
14	\$12.10	\$51.20
30	\$59.00	\$123.70
Number of observations for the month: 4		

Order Type=Catalog Order Date=1/2/18

Quantity	Cost	Price
11	\$20.20	\$40.20
100	\$5.00	\$11.80
Number of observations for the month: 2		

Order Type=In Store Order Date=1/1/18

Quantity	Cost	Price
25	\$15.65	\$31.10
Number of observations for the month: 1		

Order Type=In Store Order Date=1/2/18

Quantity	Cost	Price
2	\$36.70	\$146.40
20	\$32.30	\$71.00
Number of observations for the month: 2		

Order Type=Internet Order Date=1/1/18

Quantity	Cost	Price
2	\$20.70	\$92.60
Number of observations for the month: 1		

Example 5: Summing Numeric Variables with One BY Group

Features:

- PROC PRINT statement options
- N=
- SUMLABEL
- BY statement

SUM statement
 ODS CSVALL statement
 SORT procedure
 TITLE statement
 #BYVAL specification
 SAS system options:
 BYLINE
 NOBYLINE

Data set: [EXPREV](#)

ODS destinations:
 HTML5, CSV

Details

This example demonstrates the following tasks:

- sums expenses and revenues for each region and for all regions.
- shows the number of observations in each BY group and in the whole report.
- creates a customized title, containing the name of the region. This title replaces the default BY line for each BY group.
- creates a default HTML5 file.
- creates a CSV file.

Program: Creating an HTML5 Report

```
options obs=10 nobyline;

proc sort data=exprev;
  by sale_type;
run;

proc print data=exprev noobs label sumlabel
  n='Number of observations for the order type: '
  'Number of observations for the data set: ';

  var country order_date quantity price;

  label  sale_type='Sale Type'
         price='Total Retail Price* in USD'
         country='Country' order_date='Date' quantity='Quantity';

  sum price quantity;
  by sale_type;

  format price dollar7.2;

  title 'Retail and Quantity Totals for #byval(sale_type) Sales';
run;
```

```
options byline;
```

Program Description

The HTML5 destination is open by default. This program uses the default filename for the HTML5 output. No ODS HTML5 statement is needed.

Start each BY group on a new page and suppress the printing of the default BY line. The SAS system option NOBYLINE suppresses the printing of the default BY line. When you use PROC PRINT with the NOBYLINE option, each BY group starts on a new page. The OBS= option specifies the number of observations to process.

```
options obs=10 nobyline;
```

Sort the data set. PROC SORT sorts the observations by Sale_Type.

```
proc sort data=exprev;
  by sale_type;
run;
```

Print the report, suppress the printing of observation numbers, and print the total number of observations for the selected variables. NOOBS suppresses the printing of observation numbers at the beginning of the rows. SUMLABEL prints the BY variable label on the summary line of each. N= prints the number of observations in a BY group at the end of that BY group and (because of the SUM statement) prints the number of observations in the data set at the end of the report. The first piece of explanatory text that N= provides precedes the number for each BY group. The second piece of explanatory text that N= provides precedes the number for the entire data set.

```
proc print data=exprev noobs label sumlabel
  n='Number of observations for the order type: '
  'Number of observations for the data set: ';
```

Select the variables to include in the report. The VAR statement creates columns for Country, Order_Date, Quantity, and Price, in that order.

```
var country order_date quantity price;
```

Assign the variables' labels as column headings. The LABEL statement associates a label with each variable for the duration of the PROC PRINT step.

```
label  sale_type='Sale Type'
       price='Total Retail Price* in USD'
       country='Country' order_date='Date' quantity='Quantity';
```

Sum the values for the selected variables. The SUM statement alone sums the values of Price and Quantity for the entire data set. Because the PROC PRINT step contains a BY statement, the SUM statement also sums the values of Price and Quantity for each sale type that contains more than one observation.

```
sum price quantity;
by sale_type;
```

Format the numeric values for a specified column. The FORMAT statement assigns the DOLLAR7.2. format to Price for this report.

```
format price dollar7.2;
```

Specify and format a dynamic (or current) title. The TITLE statement specifies a title. The #BYVAL specification places the current value of the BY variable Sale_Type in the title. Because NOBYLINE is in effect, each BY group starts on a new page, and the title serves as a BY line.

```
title 'Retail and Quantity Totals for #byval(sale_type) Sales';
run;
```

Generate the default BY line. The SAS system option BYLINE resets the printing of the default BY line.

```
options byline;
```

Output: HTML5

Output 40.9 Summing Numeric Variables with One BY Group HTML5 Output

Retail and Quantity Totals for Catalog Sales

Country	Date	Quantity	Total Retail Price* in USD
Bermuda	1/1/18	7	\$41.00
Bahamas	1/1/18	8	\$113.40
Puerto Rico	1/1/18	14	\$51.20
Aruba	1/1/18	30	\$123.70
British Virgin Islands	1/2/18	11	\$40.20
Canada	1/2/18	100	\$11.80
Sale Type		170	\$381.30
Number of observations for the order type: 6			

Retail and Quantity Totals for In Store Sales

Country	Date	Quantity	Total Retail Price* in USD
Virgin Islands (U.S.)	1/1/18	25	\$31.10
Belize	1/2/18	2	\$146.40
Cayman Islands	1/2/18	20	\$71.00
Sale Type		47	\$248.50
Number of observations for the order type: 3			

Retail and Quantity Totals for Internet Sales

Country	Date	Quantity	Total Retail Price* in USD
Antarctica	1/1/18	2	\$92.60
		219	\$722.40
Number of observations for the order type: 1			
Number of observations for the data set: 10			

Program: Creating a CSV File

```
options obs=10 nobyline;
ods csvall file='your_file.csv';
proc sort data=exprev;
  by sale_type;
run;

proc print data=exprev noobs label sumlabel
  n='Number of observations for the order type: '
  'Number of observations for the data set: ';

var country order_date quantity price;

  label price='Total Retail Price* in USD'
  country='Country' order_date='Date' quantity='Quantity';

  sum price quantity;
  by sale_type;

  format price dollar7.2;

  title 'Retail and Quantity Totals for #byval(sale_type) Sales';
run;

options byline;
ods csvall close;
```

Program Description

```
options obs=10 nobyline;
```

Produce CSV formatted output and specify the file to store it in. The ODS CSVALL statement opens the CSVALL destination and creates a file containing tabular output with titles, notes, and BY lines. The FILE= argument specifies the external file that contains the CSV output.

```
ods csvall file='your_file.csv';

proc sort data=exprev;
  by sale_type;
run;

proc print data=exprev noobs label sumlabel
  n='Number of observations for the order type: '
  'Number of observations for the data set: ';

var country order_date quantity price;

  label price='Total Retail Price* in USD'
  country='Country' order_date='Date' quantity='Quantity';

  sum price quantity;
  by sale_type;

  format price dollar7.2;

  title 'Retail and Quantity Totals for #byval(sale_type) Sales';
run;
```

```
options byline;
```

Close the CSVALL destination. The ODS CSVALL CLOSE statement closes the CSVALL destination.

```
ods csvall close;
```

Output: CSV File

Output 40.10 *Summing Numeric Variables with One BY Group: CSV Output Viewed with Microsoft Excel*

	A	B	C	D	E	F
1	Retail and Quantity Totals for Catalog Sales					
2						
3	Country	Date	Quantity	Total Retail Price* in USD		
4	Bermuda	1/1/2018	7	\$41.00		
5	Bahamas	1/1/2018	8	\$113.40		
6	Puerto Rico	1/1/2018	14	\$51.20		
7	Aruba	1/1/2018	30	\$123.70		
8	British Virgin Islands	1/2/2018	11	\$40.20		
9	Canada	1/2/2018	100	\$11.80		
10	Sale_Type		170	\$381.30		
11	Number of observations for the order type: 6					
12						
13	Retail and Quantity Totals for In Store Sales					
14						
15	Country	Date	Quantity	Total Retail Price* in USD		
16	Virgin Islands	1/1/2018	25	\$31.10		
17	Belize	1/2/2018	2	\$146.40		
18	Cayman Islands	1/2/2018	20	\$71.00		
19	Sale_Type		47	\$248.50		
20	Number of observations for the order type: 3					
21						
22	Retail and Quantity Totals for Internet Sales					
23						
24	Country	Date	Quantity	Total Retail Price* in USD		
25	Antarctica	1/1/2018	2	\$92.60		
26			219	\$722.40		
27	Number of observations for the order type: 1					
28	Number of observations for the data set: 10					
29						

Example 6: Summing Numeric Variables with Multiple BY Variables

Features:

```
PROC PRINT statement options
  GRANDTOTAL_LABEL=
  N=
  NOOBS
  SUMLABEL=
```

BY statement
 SUM statement
 ODS HTML5 statement
 LABEL statement
 FORMAT statement
 SORT procedure
 TITLE statement
 Data set: [EXPREV](#)
 ODS
 destination: HTML5

Details

This example demonstrates the following tasks:

- sums quantities and retail prices for the following items:
 - ☐ each order date
 - ☐ each sale type with more than one row in the report
 - ☐ all rows in the report
- shows the number of observations in each BY group and in the whole report
- displays a customized label in place of the BY group variable name on the summary line
- displays a customized label for the grand total line
- creates a default HTML5 report
- creates a stylized HTML5 report

Program: Creating an HTML5 Report

```

options obs=10;

proc sort data=exprev;
  by sale_type order_date;
run;

proc print data=exprev n noobs sumlabel='Totals'
  grandtotal_label='Grand Total';
  by sale_type order_date;
  sum price quantity cost;

  label sale_type='Sale Type' order_date='Sale Date';
  format price dollar10.2 cost dollar10.2;
  title 'Retail and Quantity Totals for Each Sale Date and Sale Type';
run;

```


Program Description

```
options obs=10;
```

Produce HTML5 output. The SORT procedure first sorts the EXPREV data set by `sale_type` and `order_date`. The PROC PRINT step then prints the sorted data, grouping the output by those same variables and displaying subtotals and a grand total for the price, quantity, and cost columns.

```
proc sort data=exprev;  
    by sale_type order_date;  
run;  
  
proc print data=exprev n noobs sumlabel='Totals'  
    grandtotal_label='Grand Total';  
    by sale_type order_date;  
    sum price quantity cost;  
  
    label sale_type='Sale Type' order_date='Sale Date';  
    format price dollar10.2 cost dollar10.2;  
    title 'Retail and Quantity Totals for Each Sale Date and Sale Type';  
run;
```

Output: HTML5

Output 40.11 *Summing Numeric Variables with Multiple BY Variables: In Store Sales: Default HTML5 Output*

Retail and Quantity Totals for Each Sale Date and Sale Type

Sale Type=Catalog Sale Date=1/1/18

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Bermuda	99999999	1/4/18	7	\$41.00	\$9.25
Bahamas	99999999	1/4/18	8	\$113.40	\$28.45
Puerto Rico	99999999	1/5/18	14	\$51.20	\$12.10
Aruba	99999999	1/4/18	30	\$123.70	\$59.00
Totals			59	\$329.30	\$108.80
N = 4					

Sale Type=Catalog Sale Date=1/2/18

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
British Virgin Islands	99999999	1/5/18	11	\$40.20	\$20.20
Canada	99999999	1/5/18	100	\$11.80	\$5.00
Totals			111	\$52.00	\$25.20
Totals			170	\$381.30	\$134.00
N = 2					

Sale Type=In Store Sale Date=1/1/18

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Virgin Islands (U.S.)	99999999	1/4/18	25	\$31.10	\$15.65
N = 1					

Sale Type=In Store Sale Date=1/2/18

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Belize	120458	1/2/18	2	\$146.40	\$36.70
Cayman Islands	120454	1/2/18	20	\$71.00	\$32.30
Totals			22	\$217.40	\$69.00
Totals			47	\$248.50	\$84.65
N = 2					

Sale Type=Internet Sale Date=1/1/18

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Antarctica	99999999	1/7/18	2	\$92.60	\$20.70
Grand Total			219	\$722.40	\$239.35
N = 1					
Total N = 10					

Program: Creating an HTML5 Report with the STYLE Option

```
options obs=10;

proc sort data=expres;
  by sale_type order_date;
run;
```

```
proc print data=exprev n noobs sumlabel='Totals'
grandtotal_label='Grand Total';

    by sale_type order_date;
sum price / style(GRANDTOTAL)=[backgroundcolor=white color=blue];
sum quantity / style(TOTAL)=[backgroundcolor=dark blue color=white];

    label sale_type='Sale Type' order_date='Sale Date';
    format price dollar10.2 cost dollar10.2;
    title 'Retail and Quantity Totals for Each Sale Date and Sale Type';
run;
```

Program Description

```
options obs=10;

proc sort data=exprev;
    by sale_type order_date;
run;

proc print data=exprev n noobs sumlabel='Totals'
grandtotal_label='Grand Total';
```

Create stylized HTML5 output. The STYLE option in the first SUM statement specifies that the background color of the cell containing the grand total for the variable Price be changed to white and the font color be changed to blue. The STYLE option in the second SUM statement specifies that the background color of cells containing totals for the variable Quantity be changed to dark blue and the font color be changed to white.

```
    by sale_type order_date;
sum price / style(GRANDTOTAL)=[backgroundcolor=white color=blue];
sum quantity / style(TOTAL)=[backgroundcolor=dark blue color=white];

    label sale_type='Sale Type' order_date='Sale Date';
    format price dollar10.2 cost dollar10.2;
    title 'Retail and Quantity Totals for Each Sale Date and Sale Type';
run;
```

Output: HTML5 with Styles

Output 40.12 *Summing Numeric Variables with Multiple BY Variables: Catalog Sales: HTML5 Output Using Styles*

Retail and Quantity Totals for Each Sale Date and Sale Type

Sale Type=Catalog Sale Date=1/1/18

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Bermuda	99999999	1/4/18	7	\$41.00	\$9.25
Bahamas	99999999	1/4/18	8	\$113.40	\$28.45
Puerto Rico	99999999	1/5/18	14	\$51.20	\$12.10
Aruba	99999999	1/4/18	30	\$123.70	\$59.00
Totals			59	\$329.30	

N = 4

Sale Type=Catalog Sale Date=1/2/18

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
British Virgin Islands	99999999	1/5/18	11	\$40.20	\$20.20
Canada	99999999	1/5/18	100	\$11.80	\$5.00
Totals			111	\$52.00	
Totals			170	\$381.30	

N = 2

Sale Type=In Store Sale Date=1/1/18

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Virgin Islands (U.S.)	99999999	1/4/18	25	\$31.10	\$15.65

N = 1

Sale Type=In Store Sale Date=1/2/18

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Belize	120458	1/2/18	2	\$146.40	\$36.70
Cayman Islands	120454	1/2/18	20	\$71.00	\$32.30
Totals			22	\$217.40	
Totals			47	\$248.50	

N = 2

Sale Type=Internet Sale Date=1/1/18

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Antarctica	99999999	1/7/18	2	\$92.60	\$20.70
Grand Total			219	\$722.40	

N = 1

Total N = 10

Example 7: Limiting the Number of Sums in a Report

Features:

BY statement
SUM statement
SUMBY statement
FORMAT statement
LABEL statement

ODS PDF statement
 SORT procedure
 TITLE statement
 Data set: [EXPREV](#)
 ODS
 destination: PDF

Details

This example demonstrates the following tasks:

- creates a separate section of the report for each combination of sale type and sale date
- sums quantities and retail prices only for each sale type and for all sale types, not for individual dates
- creates a PDF file

Program: Creating a PDF File

```
options obs=10;
ods html5 close;
ods pdf file='your_file.pdf';

proc sort data=exprev;
  by sale_type order_date;
run;

proc print data=exprev noobs sumlabel='Total' grandtotal_label='Grand
Total';

  by sale_type order_date;
  sum price quantity;
  sumby sale_type;

  label sale_type='Sale Type' order_date='Sale Date';

  format price dollar10.2 cost dollar10.2;
  title 'Retail and Quantity Totals for Each Sale Type';
run;

ods pdf close;
```

Program Description

The OBS= system option specifies to process 10 observations.

```
options obs=10;
```

Produce PDF output and specify the file to store the output in. The ODS HTML5 CLOSE statement closes the default destination. The ODS PDF statement opens the PDF destination and creates a file that contains PDF output. The FILE= argument specifies the external file that contains the PDF output.

```
ods html5 close;
ods pdf file='your_file.pdf';
```

Sort the data set. PROC SORT sorts the observations by Sales_Type and Order_Date.

```
proc sort data=exprev;
  by sale_type order_date;
run;
```

Print the report and remove the observation numbers. NOOBS suppresses the printing of observation numbers at the beginning of the rows. SUMLABEL uses the label for the BY variables on the summary line of each BY group.

```
proc print data=exprev noobs sumlabel='Total' grandtotal_label='Grand
Total';
```

Sum the values for each region. The SUM and BY statements work together to sum the values of Price and Quantity for each BY group as well as for the whole report. The SUMBY statement limits the subtotals to one for each type of sale.

```
  by sale_type order_date;
  sum price quantity;
sumby sale_type;
```

Assign labels to specific variables. The LABEL statement associates a label with the variables Sale_Type and Order_Date for the duration of the PROC PRINT step. These labels are used in the BY group title or the summary line.

```
label sale_type='Sale Type' order_date='Sale Date';
```

Assign a format to the necessary variables and specify a title. The FORMAT statement assigns the COMMA10. format to Cost and Price for this report. The TITLE statement specifies a title.

```
format price dollar10.2 cost dollar10.2;
title 'Retail and Quantity Totals for Each Sale Type';
run;
```

Close the PDF destination. The ODS PDF CLOSE statement closes the PDF destination.

```
ods pdf close;
```

Output: PDF

Output 40.13 Limiting the Number of Sums in a Report: PDF Output

Retail and Quantity Totals for Each Sale Type

Sale Type=Catalog Sale Date=1/1/18

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Bermuda	99999999	1/4/18	7	\$41.00	\$9.25
Bahamas	99999999	1/4/18	8	\$113.40	\$28.45
Puerto Rico	99999999	1/5/18	14	\$51.20	\$12.10
Aruba	99999999	1/4/18	30	\$123.70	\$59.00

Sale Type=Catalog Sale Date=1/2/18

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
British Virgin Islands	99999999	1/5/18	11	\$40.20	\$20.20
Canada	99999999	1/5/18	100	\$11.80	\$5.00
Total			170	\$381.30	

Sale Type=In Store Sale Date=1/1/18

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Virgin Islands (U.S.)	99999999	1/4/18	25	\$31.10	\$15.65

Sale Type=In Store Sale Date=1/2/18

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Belize	120458	1/2/18	2	\$146.40	\$36.70
Cayman Islands	120454	1/2/18	20	\$71.00	\$32.30
Total			47	\$248.50	

Sale Type=Internet Sale Date=1/1/18

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Antarctica	99999999	1/7/18	2	\$92.60	\$20.70
Grand Total			219	\$722.40	

Program: Creating a PDF Report with the STYLE Option

```
options obs=10;
ods pdf file='your_file.pdf';
proc sort data=exprev;
  by sale_type order_date;
run;

proc print data=exprev noobs sumlabel='Total' grandtotal_label='Grand
Total';
```

```

    by sale_type order_date;

    sum quantity price / style(TOTAL)=[backgroundcolor=light blue
color=white];
    sum quantity price / style(GRANDTOTAL)=[backgroundcolor=green
color=white];
    sumby sale_type;

    label sale_type='Sale Type' order_date='Sale Date';

    format price dollar10.2 cost dollar10.2;
    title 'Retail and Quantity Totals for Each Sale Type';
run;

ods pdf close;

```

Program Description

```

options obs=10;

ods pdf file='your_file.pdf';

proc sort data=exprev;
    by sale_type order_date;
run;

proc print data=exprev noobs sumlabel='Total' grandtotal_label='Grand
Total';

    by sale_type order_date;

```

Create stylized PDF output. The STYLE option in the first SUM statement specifies that the background color of cells containing totals for the variable Price be changed to light blue and the font color be changed to white. The STYLE option in the second SUM statement specifies that the background color of the cell containing the grand total for the Quantity variable be changed to green and the font color be changed to white.

```

    sum quantity price / style(TOTAL)=[backgroundcolor=light blue
color=white];
    sum quantity price / style(GRANDTOTAL)=[backgroundcolor=green
color=white];
    sumby sale_type;

    label sale_type='Sale Type' order_date='Sale Date';

    format price dollar10.2 cost dollar10.2;
    title 'Retail and Quantity Totals for Each Sale Type';
run;

ods pdf close;

```


Output: PDF with Styles

Output 40.14 Limiting the Number of Sums in a Report: PostScript Output Using Styles

Retail and Quantity Totals for Each Sale Type

Sale Type=Catalog Sale Date=1/1/18

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Bermuda	99999999	1/4/18	7	\$41.00	\$9.25
Bahamas	99999999	1/4/18	8	\$113.40	\$28.45
Puerto Rico	99999999	1/5/18	14	\$51.20	\$12.10
Aruba	99999999	1/4/18	30	\$123.70	\$59.00

Sale Type=Catalog Sale Date=1/2/18

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
British Virgin Islands	99999999	1/5/18	11	\$40.20	\$20.20
Canada	99999999	1/5/18	100	\$11.80	\$5.00
Total			170	\$381.30	

Sale Type=In Store Sale Date=1/1/18

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Virgin Islands (U.S.)	99999999	1/4/18	25	\$31.10	\$15.65

Sale Type=In Store Sale Date=1/2/18

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Belize	120458	1/2/18	2	\$146.40	\$36.70
Cayman Islands	120454	1/2/18	20	\$71.00	\$32.30
Total			47	\$248.50	

Sale Type=Internet Sale Date=1/1/18

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Antarctica	99999999	1/7/18	2	\$92.60	\$20.70
Grand Total			219	\$722.40	

Example 8: Controlling the Layout of a Report with Many Variables

Features:

- PROC PRINT statement options
- ROWS=
- ID statement options
- STYLE

	ODS WORD statement
	SAS data set options
	OBS=
Data set:	EMPDATA
ODS	WORD
destination:	

Details

This example shows two ways of printing a data set with a large number of variables: one is the default, printing multiple rows when there are a large number of variables, and the other uses ROWS= option to print one row. The ROWS= option is valid only for the LISTING destination. For detailed explanations of the layouts of these two reports, see the option [ROWS= on page 1409](#) and “[Page Layout for Limited Page Sizes](#)” on page 1448.

These reports use a page size of 24 and a line size of 64 to help illustrate the different layouts.

Program: Creating a WORD Report

```
options pageno=1;
ods word file='your_file.docx';
proc print data=empdata(obs=12);
  id idnumber;
  title 'Personnel Data';
run;
ods word close;
```

Program Description

The WORD output shows all data in one page.

```
options pageno=1;
```

Create output for Microsoft Word and specify the file to store the output in. The ODS WORD statement opens the WORD destination and creates output formatted for Microsoft Word. The FILE= argument specifies the external file that contains the WORD output.

```
ods word file='your_file.docx';
proc print data=empdata(obs=12);
  id idnumber;
  title 'Personnel Data';
```

```
run;
```

Close the WORD destination. The ODS WORD CLOSE statement closes the WORD destination.

```
ods word close;
```

Output: WORD

Output 40.15 Layout for a Report with Many Variables: WORD Output

Personnel Data

IdNumber	LastName	FirstName	City	State	Gender	JobCode	Salary	Birth	Hired	HomePhone
1919	Adams	Gerald	Stamford	CT	M	TA2	34376	12SEP1990	04JUN2020	203/781-1255
1653	Ahmad	Azeem	Bridgeport	CT	F	ME2	35108	15OCT1984	09AUG2020	203/675-7715
1400	Alvarez	Gloria	New York	NY	M	ME1	29769	05NOV1987	16OCT2020	212/586-0808
1350	Arthur	Barbara	New York	NY	F	FA3	32886	31AUG1985	29JUL2020	718/383-1549
1401	Avery	Jerry	Paterson	NJ	M	TA3	38822	13DEC1970	17NOV2020	201/732-8787
1499	Barefoot	Joseph	Princeton	NJ	M	ME3	43025	26APR1974	07JUN2020	201/812-5665
1101	Basquez	Richard	New York	NY	M	SCP	18723	06JUN1982	01OCT2020	212/586-8060
1333	Beaulieu	Armando	Stamford	CT	M	PT2	88606	30MAR1983	10FEB2020	203/781-1777
1402	Blalock	Ralph	New York	NY	M	TA2	32615	17JAN1983	02DEC2020	718/384-2849
1479	Bostic	Marie	New York	NY	F	TA3	38785	22DEC1988	05OCT2020	718/384-8816
1403	Bowden	Earl	Bridgeport	CT	M	ME1	28072	28JAN1989	21DEC2020	203/675-3434
1739	Boyce	Jonathan	New York	NY	M	PT1	66517	25DEC1984	27JAN2020	212/587-1247

Example 9: Creating a Customized Layout with BY Groups and ID Variables

Features: PROC PRINT statement options
SUMLABEL
GRANDTOTAL_LABEL

BY statement
ID statement
SUM statement
VAR statement
SORT procedure

Data set: EMPDATA

ODS
destination: HTML5

Details

This customized report demonstrates the following tasks:

- selects variables to include in the report and the order in which they appear
- selects observations to include in the report
- groups the selected observations by JobCode
- sums the salaries for each job code and for all job codes
- displays numeric data with commas and dollar signs

Program: Creating an HTML5 Report

```
proc sort data=empdata out=tempemp;
    by jobcode gender;
run;

ods html5 file='your_file.html';

proc print data=tempemp (obs=10) sumlabel='Total'
    grandtotal_lable='Grand Total';

    id jobcode;
    by jobcode;
    var gender salary;

    sum salary;

    label jobcode='Job Code'
        gender='Gender'
        salary='Annual Salary';

    format salary dollar11.2;
    where jobcode contains 'FA' or jobcode contains 'ME';
    title 'Salary Expenses';
run;
```

Program Description

```
proc sort data=empdata out=tempemp;
    by jobcode gender;
run;
```

Produce HTML5 output and specify the file to store the output in. The HTML5 destination is the default ODS destination. The ODS HTML5 statement FILE= option specifies the external file that contains the HTML5 output.

```
ods html5 file='your_file.html';
```

Define the procedure options. The (obs=10) data set option sets the number of observations to process. The SUMLABEL option indicates to use the label 'Total' on the summary line for each BY group. The GRANDTOTAL_LABEL option indicates to use the label 'Grand Total' on the grand total line after all BY groups in the report.

```
proc print data=tempemp (obs=10) sumlabel='Total'
grandtotal_label='Grand Total';

  id jobcode;
  by jobcode;
  var gender salary;

  sum salary;

  label jobcode='Job Code'
        gender='Gender'
        salary='Annual Salary';

  format salary dollar11.2;
  where jobcode contains 'FA' or jobcode contains 'ME';
  title 'Salary Expenses';
run;
```

Output: HTML5

Output 40.16 *Creating a Customized Layout with BY Groups and ID Variables:
Default HTML5 Output*

Salary Expenses

JobCode	Gender	Salary
FA1	F	\$23,177.00
	F	\$22,454.00
	M	\$22,268.00
Total		\$67,899.00

JobCode	Gender	Salary
FA2	F	\$28,888.00
	F	\$27,787.00
	M	\$28,572.00
Total		\$85,247.00

JobCode	Gender	Salary
FA3	F	\$32,886.00
	F	\$33,419.00
	M	\$32,217.00
Total		\$98,522.00

JobCode	Gender	Salary
ME1	M	\$29,769.00
Grand Total		\$281,437.00

Program: Creating an HTML5 Report with the STYLE Option

```
proc sort data=empdata out=tempemp;
  by jobcode gender;
run;

proc print data=tempemp (obs=10) sumlabel='Total'
  grandtotal_label='Grand Total'
  style(HEADER)={fontstyle=italic}
  style(DATA)={backgroundcolor=blue foreground=white};

  id jobcode;
  by jobcode;
  var gender salary;

  sum salary / style(total)={color=red};

  label jobcode='Job Code'
        gender='Gender'
```

```

        salary='Annual Salary';

format salary dollar11.2;
where jobcode contains 'FA' or jobcode contains 'ME';
title 'Expenses Incurred for';
title2 'Salaries for Flight Attendants and Mechanics';
run;

```

Program Description

```

proc sort data=empdata out=tempemp;
    by jobcode gender;
run;

```

Create stylized HTML5 output. The first STYLE option specifies that the font of the headers be changed to italic. The second STYLE option specifies that the background of cells that contain data be changed to blue and the foreground of these cells be changed to white. The SUMLABEL and GRANDTOTAL_LABEL options use a label in the summary and grand total lines, respectively, in place of variable names.

```

proc print data=tempemp (obs=10) sumlabel='Total'
grandtotal_label='Grand Total'
    style(HEADER)={fontstyle=italic}
    style(DATA)={backgroundcolor=blue foreground=white};

id jobcode;
by jobcode;
var gender salary;

```

Create total values that are written in red. The STYLE option specifies that the color of the foreground of the cell that contain the totals be changed to red.

```

sum salary / style(total)={color=red};

label jobcode='Job Code'
      gender='Gender'
      salary='Annual Salary';

format salary dollar11.2;
where jobcode contains 'FA' or jobcode contains 'ME';
title 'Expenses Incurred for';
title2 'Salaries for Flight Attendants and Mechanics';
run;

```

Output: HTML5 with Styles

Output 40.17 *Creating a Customized Layout with BY Groups and ID Variables:
HTML5 Output Using Styles*

Expenses Incurred for Salaries for Flight Attendants and Mechanics

JobCode	Gender	Salary
FA1	F	\$23,177.00
	F	\$22,454.00
	M	\$22,268.00
Total		\$67,899.00

JobCode	Gender	Salary
FA2	F	\$28,888.00
	F	\$27,787.00
	M	\$28,572.00
Total		\$85,247.00

JobCode	Gender	Salary
FA3	F	\$32,886.00
	F	\$33,419.00
	M	\$32,217.00
Total		\$98,522.00

JobCode	Gender	Salary
ME1	M	\$29,769.00
Grand Total		\$281,437.00

Example 10: Printing All the Data Sets in a SAS Library

Features:	Macro facility DATASETS procedure PRINT procedure
Data sets:	PROCLIB.DELAY and PROCLIB.INTERNAT from the Raw Data and DATA Steps appendix
ODS destination:	HTML5

Details

This example prints all the data sets in a SAS library. You can use the same programming logic with any procedure. Just replace the PROC PRINT step near the end of the example with whatever procedure step you want to execute. The example uses the macro language. For details about the macro language, see [SAS Macro Language: Reference](#).

Program: Printing All of the Data Sets in a Library

```

libname printlib 'SAS-data-library';
libname proclib 'SAS-data-library';
options nodate pageno=1;

proc datasets library=proclib memtype=data nolist;
  copy out=printlib;
  select delay internat;
run;

%macro printall(libname,worklib=work);
  %local num i;

  proc datasets library=&libname memtype=data nodetails;
    contents out=&worklib..temp1(keep=memname) data=_all_ noprint;
  run;

  data _null_;
    set &worklib..temp1 end=final;
    by memname notsorted;
    if last.memname;
      n+1;
      call symput('ds' || left(put(n,8.)),trim(memname));

      if final then call symput('num',put(n,8.));
  run;

  %do i=1 %to &num;
    proc print data=&libname..&ds&i noobs;
      title "Data Set &libname..&ds&i";
    run;
  %end;
%mend printall;

%printall(printlib)

```

Program Description

```

libname printlib 'SAS-data-library';
libname proclib 'SAS-data-library';

```

```
options nodate pageno=1;
```

Copy the desired data sets from the WORK library to a permanent library. PROC DATASETS copies two data sets from the WORK library to the PRINTLIB library in order to limit the number of data sets available to the example.

```
proc datasets library=proclib memtype=data nolist;
  copy out=printlib;
  select delay internat;
run;
```

Create a macro and specify the parameters. The %MACRO statement creates the macro PRINTALL. When you call the macro, you can pass one or two parameters to it. The first parameter is the name of the library whose data set you want to print. The second parameter is a library used by the macro. If you do not specify this parameter, the WORK library is the default.

```
%macro printall(libname,worklib=work);
```

Create the local macro variables. The %LOCAL statement creates two local macro variables, NUM and I, to use in a loop.

```
%local num i;
```

Produce an output data set. This PROC DATASETS step reads the library that you specify as a parameter when you invoke the macro. The CONTENTS statement produces an output data set called TEMP1 in WORKLIB. This data set contains an observation for each variable in each data set in the library LIBNAME. By default, each observation includes the name of the data set that the variable is included in as well as other information about the variable. However, the KEEP= data set option writes only the name of the data set to TEMP1.

```
proc datasets library=&libname memtype=data nodetails;
  contents out=&worklib..temp1(keep=memname) data=_all_ noprint;
run;
```

Specify the unique values in the data set, assign a macro variable to each one, and assign DATA step information to a macro variable. This DATA step increments the value of N each time it reads the last occurrence of a data set name (when IF LAST.MEMNAME is true). The CALL SYMPUT statement uses the current value of N to create a macro variable for each unique value of MEMNAME in the data set TEMP1. The TRIM function removes extra blanks in the TITLE statement in the PROC PRINT step that follows.

```
data _null_;
  set &worklib..temp1 end=final;
  by memname notsorted;
  if last.memname;
  n+1;
  call symput('ds' || left(put(n,8.)),trim(memname));
```

Determine the number of observations in the DATA step. When it reads the last observation in the data set (when FINAL is true), the DATA step assigns the value of N to the macro variable NUM. At this point in the program, the value of N is the number of observations in the data set.

```
if final then call symput('num',put(n,8.));
```

Run the DATA step. The RUN statement is crucial. It forces the DATA step to run, thus creating the macro variables that are used in the CALL SYMPUT statements before the %DO loop, which uses them, executes.

```
run;
```

Print the data sets and end the macro. The %DO loop issues a PROC PRINT step for each data set. The %MEND statement ends the macro.

```
%do i=1 %to &num;
  proc print data=&libname..&&ds&i noobs;
    title "Data Set &libname..&&ds&i";
  run;
%end;
%mend printall;
```

Print all the data sets in the PRINTLIB library. This invocation of the PRINTALL macro prints all the data sets in the library PRINTLIB.

```
%printall(printlib)
```

Output: HTML5

Output 40.18 Data Set PRINTLIB.DELAY

Data Set PRINTLIB.DELAY

flight	date	orig	dest	delaycat	destype	delay
114	01MAR18	LGA	LAX	1-10 Minutes	Domestic	8
202	01MAR18	LGA	ORD	No Delay	Domestic	-5
219	01MAR18	LGA	LON	11+ Minutes	International	18
622	01MAR18	LGA	FRA	No Delay	International	-5
132	01MAR18	LGA	YYZ	11+ Minutes	International	14
271	01MAR18	LGA	PAR	1-10 Minutes	International	5
302	01MAR18	LGA	WAS	No Delay	Domestic	-2
114	02MAR18	LGA	LAX	No Delay	Domestic	0
202	02MAR18	LGA	ORD	1-10 Minutes	Domestic	5
219	02MAR18	LGA	LON	11+ Minutes	International	18
622	02MAR18	LGA	FRA	No Delay	International	0
132	02MAR18	LGA	YYZ	1-10 Minutes	International	5
271	02MAR18	LGA	PAR	1-10 Minutes	International	4
302	02MAR18	LGA	WAS	No Delay	Domestic	0
114	03MAR18	LGA	LAX	No Delay	Domestic	-1
202	03MAR18	LGA	ORD	No Delay	Domestic	-1
219	03MAR18	LGA	LON	1-10 Minutes	International	4
622	03MAR18	LGA	FRA	No Delay	International	-2
132	03MAR18	LGA	YYZ	1-10 Minutes	International	6
271	03MAR18	LGA	PAR	1-10 Minutes	International	2
302	03MAR18	LGA	WAS	1-10 Minutes	Domestic	5
114	04MAR18	LGA	LAX	11+ Minutes	Domestic	15
202	04MAR18	LGA	ORD	No Delay	Domestic	-5
219	04MAR18	LGA	LON	1-10 Minutes	International	3
622	04MAR18	LGA	FRA	11+ Minutes	International	30
132	04MAR18	LGA	YYZ	No Delay	International	-5
271	04MAR18	LGA	PAR	1-10 Minutes	International	5
302	04MAR18	LGA	WAS	1-10 Minutes	Domestic	7
114	05MAR18	LGA	LAX	No Delay	Domestic	-2
202	05MAR18	LGA	ORD	1-10 Minutes	Domestic	2
219	05MAR18	LGA	LON	1-10 Minutes	International	3
622	05MAR18	LGA	FRA	No Delay	International	-6
132	05MAR18	LGA	YYZ	1-10 Minutes	International	3
271	05MAR18	LGA	PAR	1-10 Minutes	International	5
114	06MAR18	LGA	LAX	No Delay	Domestic	-1
202	06MAR18	LGA	ORD	No Delay	Domestic	-3
219	06MAR18	LGA	LON	11+ Minutes	International	27
132	06MAR18	LGA	YYZ	1-10 Minutes	International	7
302	06MAR18	LGA	WAS	1-10 Minutes	Domestic	1
114	07MAR18	LGA	LAX	No Delay	Domestic	-1
202	07MAR18	LGA	ORD	No Delay	Domestic	-2
219	07MAR18	LGA	LON	11+ Minutes	International	15
622	07MAR18	LGA	FRA	11+ Minutes	International	21
132	07MAR18	LGA	YYZ	No Delay	International	-2
271	07MAR18	LGA	PAR	1-10 Minutes	International	4
302	07MAR18	LGA	WAS	No Delay	Domestic	0

Output 40.19 Data Set PRINTLIB.INTERNAT**Data Set PRINTLIB.INTERNAT**

flight	date	dest	boarded
219	01MAR18	LON	198
622	01MAR18	FRA	207
132	01MAR18	YYZ	115
271	01MAR18	PAR	138
219	02MAR18	LON	147
622	02MAR18	FRA	176
132	02MAR18	YYZ	106
271	02MAR18	PAR	172
219	03MAR18	LON	197
622	03MAR18	FRA	180
132	03MAR18	YYZ	75
271	03MAR18	PAR	147
219	04MAR18	LON	232
622	04MAR18	FRA	137
132	04MAR18	YYZ	117
271	04MAR18	PAR	146
219	05MAR18	LON	160
622	05MAR18	FRA	185
132	05MAR18	YYZ	157
271	05MAR18	PAR	177
219	06MAR18	LON	163
132	06MAR18	YYZ	150
219	07MAR18	LON	241
622	07MAR18	FRA	210
132	07MAR18	YYZ	164
271	07MAR18	PAR	155

PRINTTO Procedure

Overview: PRINTTO Procedure	1497
What Does the PRINTTO Procedure Do?	1497
Syntax: PRINTTO Procedure	1498
PROC PRINTTO Statement	1498
Usage: PRINTTO Procedure	1503
Set Page Numbers Using SAS System Options	1503
Route SAS Log or Procedure Output Directly to a Printer	1503
Restore the Previous SAS Log or LISTING Output File Location	1504
Send Program Output to an Alternate Location	1504
Examples: PRINTTO Procedure	1505
Example 1: Routing to External Files	1505
Example 2: Routing to SAS Catalog Entries	1509
Example 3: Using Procedure Output as an Input File	1512
Example 4: Routing to a Printer	1516

Overview: PRINTTO Procedure

What Does the PRINTTO Procedure Do?

The PRINTTO procedure defines destinations, other than ODS destinations, for SAS procedure output and for the SAS log. By default, SAS procedure output and the SAS log are routed to the default procedure output file and the default SAS log file for your method of operation. The PRINTTO procedure does not define ODS destinations. See the following table for SAS log and procedure output default destinations.

You can store the SAS log or procedure output in an external file or in a SAS catalog entry. To write SAS output to a file or a catalog entry, the ODS LISTING destination must be open. With additional programming, you can use SAS output as input data within the same job.

Table 41.1 Default Destinations for SAS Log and Procedure Output

Method of Running SAS	Default SAS Log Destination	Default Procedure Output Destination
Windowing environment	Log	Results
Noninteractive mode or batch mode	Depends on the host operating system	Depends on the operating environment

Syntax: PRINTTO Procedure

PROC PRINTTO <options>;

Statement	Task	Example
PROC PRINTTO	Define destinations, other than ODS destinations, for SAS procedure output and for the SAS log	Ex. 1, Ex. 2, Ex. 3

PROC PRINTTO Statement

Defines destinations, other than ODS destinations, for SAS procedure output and for the SAS log.

Restrictions: To route SAS log and procedure output directly to a printer, you must use a FILENAME statement with the PROC PRINTTO statement. See [“Route SAS Log or Procedure Output Directly to a Printer” on page 1503](#). [“Example 4: Routing to a Printer” on page 1516](#).

The PRINTTO procedure does not define ODS destinations.

When SAS is started in objectserver mode, the PRINTTO procedure does not route log messages to the log specified by the ALTLOG= system option.

Note: LOG=LOG and PRINT=PRINT route the log and procedure output to the default destinations. However, specifying LOG=PRINT or PRINT=LOG to route log or procedure output to the same default destination is not valid.

Tips: To reset the destination for the SAS log and procedure output to the default, use the PROC PRINTTO statement without options.

To route the SAS log and procedure output to the same file, specify the same file with both the LOG= and PRINT= options.

Examples: [“Example 1: Routing to External Files” on page 1505](#)
[“Example 2: Routing to SAS Catalog Entries” on page 1509](#)
[“Example 3: Using Procedure Output as an Input File” on page 1512](#)
[“Example 4: Routing to a Printer” on page 1516](#)

Syntax

```
PROC PRINTTO <LABEL='description' ><LOG= LOG | file-specification | SAS-  
catalog-entry><NEW ><PRINT= PRINT | file-specification | SAS-catalog-  
entry><UNIT=nn>;
```

Without Arguments

When no options are specified, the PROC PRINTTO statement does the following:

- closes any files opened by a PROC PRINTTO statement
- points both the SAS log and SAS procedure output to their default destinations
- closes the LISTING destination

Interaction: To close the appropriate file and to return only the SAS log or procedure output to its default destination, use LOG=LOG or PRINT=PRINT.

Examples:

[“Example 1: Routing to External Files” on page 1505](#)
[“Example 2: Routing to SAS Catalog Entries” on page 1509](#)

Optional Arguments

LABEL='description'

provides a description for a catalog entry that contains a SAS log or procedure output.

Range 1–256 characters

Interaction Use the LABEL= option only when you specify a catalog entry as the value for the LOG= option or the PRINT= option.

Example [“Example 2: Routing to SAS Catalog Entries” on page 1509](#)

LOG=LOG | *file-specification* | SAS-catalog-entry

routes the SAS log to one of three locations:

LOG

routes the SAS log to its default destination.

file-specification

routes the SAS log to an external file. *file-specification* can be one of the following:

'external-file'

the name of an external file specified in quotation marks.

Restriction *external-file* cannot be longer than 1024 characters.

log-filename

is an unquoted alphanumeric text string. SAS creates a log that uses *log-filename.log* as the log filename.

fileref

a fileref previously assigned to an external file.

SAS-catalog-entry

routes the SAS log to a SAS catalog entry. Express *SAS-catalog-entry* in the following way:

libref.catalog.entry<.LOG>

a SAS catalog entry stored in the SAS library and SAS catalog specified.

fileref

a fileref previously assigned to a SAS catalog entry. See [“FILENAME Statement: CATALOG Access Method” in SAS Global Statements: Reference](#).

Default LOG

Interactions The SAS log and procedure output cannot be routed to the same catalog entry at the same time.

The NEW option replaces the existing contents of a file with the new log. Otherwise, the new log is appended to the file.

To route the SAS log and procedure output to the same file, specify the same file with both the LOG= and PRINT= options.

When routing the log to a SAS catalog entry, you can use the LABEL option to provide a description for the entry in the catalog directory.

When the log is routed to a file other than the default log file and programs are submitted from multiple sources, the final SAS system messages that contain the real and CPU times are written to the default SAS log.

Tips After routing the log to an external file or a catalog entry, you can specify LOG to route the SAS log back to its default destination.

When routing the SAS log, include a RUN statement in the PROC PRINTTO statement. If you omit the RUN statement, the first line of the following DATA or PROC step is not routed to the new file. (This occurs because a statement does not execute until a step boundary is crossed.)

If you create a macro that contains a password and you do not want the password to appear in the SAS log, use the LOG=*file-specification* option to redirect the log to an external file.

When you specify LOG=, SAS stores the path of the SAS log file in the &SYSPRINTTOLOG automatic macro variable. You can use this macro variable to restore the previous SAS log file location. For more information, see [“Restore the Previous SAS Log or LISTING Output File Location” on page 1504](#).

- Examples [“Example 1: Routing to External Files” on page 1505](#)
- [“Example 2: Routing to SAS Catalog Entries” on page 1509](#)
- [“Example 3: Using Procedure Output as an Input File” on page 1512](#)

NEW

clears any information that exists in a file and prepares the file to receive the SAS log or procedure output.

Default If you omit NEW, the new information is appended to the existing file.

Interaction If you specify both LOG= and PRINT=, NEW applies to both.

- Examples [“Example 1: Routing to External Files” on page 1505](#)
- [“Example 2: Routing to SAS Catalog Entries” on page 1509](#)
- [“Example 3: Using Procedure Output as an Input File” on page 1512](#)

PRINT= PRINT | *file-specification* | *SAS-catalog-entry*

routes procedure output to one of three locations:

PRINT

routes procedure output to its default destination.

Tip After routing it to an external file or a catalog entry, you can specify PRINT to route subsequent procedure output to its default destination.

file-specification

routes procedure output to an external file. *file-specification* can be one of the following:

'external-file'

the name of an external file specified in quotation marks.

Restriction *external-file* cannot be longer than 1024 characters.

print-filename

is an unquoted alphanumeric text string. SAS creates a print file that uses *print-filename* as the print filename.

fileref

a fileref previously assigned to an external file.

SAS-catalog-entry

routes procedure output to a SAS catalog entry. Express *SAS-catalog-entry* in the following way:

libref.catalog.entry<.OUTPUT>

a SAS catalog entry stored in the SAS library and SAS catalog specified.

fileref

a fileref previously assigned to a SAS catalog entry. See [“FILENAME Statement: CATALOG Access Method” in SAS Global Statements: Reference](#).

Aliases FILE=

NAME=

Default PRINT

Interactions When you specify PRINT, FILE=, or NAME=, and the LISTING destination is not open, the PRINTTO procedure opens the LISTING destination for the duration of routing the procedure output. If the LISTING destination was open before PRINT, FILE=, or NAME= was specified, it remains open after the output has been routed to its destination.

The procedure output and the SAS log cannot be routed to the same catalog entry at the same time.

The NEW option replaces the existing contents of a file with the new procedure output. If you omit NEW, the new output is appended to the file.

To route the SAS log and procedure output to the same file, specify the same file with both the LOG= and PRINT= options.

When routing procedure output to a SAS catalog entry, you can use the LABEL option to provide a description for the entry in the catalog directory.

Tip When you specify PRINT=, SAS stores the path of the LISTING output file in the &SYSPRINTTOLIST automatic macro variable. You can use this macro variable to restore the previous LISTING output file location. For more information, see [“Restore the Previous SAS Log or LISTING Output File Location” on page 1504](#).

Example [“Example 3: Using Procedure Output as an Input File” on page 1512](#)

UNIT=nn

routes the output to the file identified by the fileref FTnnFO01, where *nn* is an integer between 1 and 99.

Range 1–99, integer only.

Tips You can define this fileref yourself. However, some operating systems predefine certain filerefs in this form.

When you specify UNIT=, SAS stores the path of the LISTING output file in the &SYSPRINTTOLIST automatic macro variable. You can use this macro variable to restore the previous LISTING output file location. For more information, see [“Restore the Previous SAS Log or LISTING Output File Location” on page 1504](#).

Usage: PRINTTO Procedure

Set Page Numbers Using SAS System Options

When the NUMBER SAS system option is in effect, there is a single page-numbering sequence for all output in the current job or session. When NONUMBER is in effect, output pages are not numbered.

You can specify the beginning page number for the output that you are currently producing by using the PAGENO= in an OPTIONS statement.

Route SAS Log or Procedure Output Directly to a Printer

To route SAS log or procedure output directly to a printer, use a FILENAME statement to associate a fileref with the printer name, and then use that fileref in the LOG= or PRINT= option. For an example, see [“Example 4: Routing to a Printer” on page 1516](#).

For more information, see [“FILENAME Statement” in SAS Global Statements: Reference](#).

The PRINTTO procedure does not support the COLORPRINTING system option. If you route the SAS log or procedure output to a color printer, the output does not print in color.

Restore the Previous SAS Log or LISTING Output File Location

When you specify the LOG=, PRINT=, or the UNIT= options in the PROC PRINTTO statement, SAS stores the appropriate file location in automatic macro variables:

SYSPRINTTOLOG	contains the path of the SAS log file location prior to redirection by the PRINTTO procedure
SYSPRINTTOLIST	contains the path of the LISTING output file location prior to redirection by the PRINTTO procedure

To restore the previous file locations, you specify the appropriate automatic macro variable as the value of the LOG=, PRINT=, or UNIT= options. Here are some examples:

```
/* Restore the previous log and the listing file locations. */
proc printto log=&sysprinttolog print=&sysprinttolist;
run;

/* Restore the previous listing file location. */
proc printto unit=&sysprinttolist;
run;
```

Send Program Output to an Alternate Location

The following statements send any SAS log entries that are generated after the RUN statement to the external file that is associated with the fileref myfile:

```
filename myfile '/users/myid/mydir/mylog';
proc printto log=myfile;
run;
```

If myfile has not been defined as a fileref, then PROC PRINTTO creates the file myfile.log in the current directory.

The following statements send any procedure output that is generated after the RUN statement to the file /users/myid/mydir/myout:

```
proc printto print='/users/myid/mydir/myout';
run;
```

The following statements send the procedure output from the CONTENTS procedure directly to the system printer:

```
filename myfile printer;
proc printto print=myfile;
run;
proc contents data=oranges;
```

```
run;
```

To redirect the SAS log and procedure output to their original default destinations, run PROC PRINTTO without any options:

```
proc printto;
run;
```

If filerefs `myprint` and `mylog` have not been defined, then the following statements send any SAS procedure output to `myprint.lst` and any log output to `mylog.log` in the current directory:

```
proc printto print=myprint log=mylog;
run;
```

If filerefs `myprint` and `mylog` had been defined, the output would have gone to the files that are associated with those filerefs.

Examples: PRINTTO Procedure

Example 1: Routing to External Files

Features:	PRINTTO statement without options
	PRINTTO statement options
	LOG=
	NEW
	PRINT=

Details

This example uses PROC PRINTTO to route the log and procedure output to an external file and then reset both destinations to the default.

Program

```
options nodate pageno=1 linesize=80 pagesize=60 source;

proc printto log='log-file';
run;

data numbers;
input x y z;
```

```

        datalines;
14.2    25.2    96.8
10.8    51.6    96.8
  9.5    34.2   138.2
  8.8    27.6    83.2
11.5    49.4   287.0
  6.3    42.0   170.7
        ;

proc printto print='output-file'
new;
run;

proc print data=numbers;
    title 'Listing of NUMBERS Data Set';
run;

proc printto;
run;

```

Program Description

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page. The SOURCE option writes lines of source code to the default destination for the SAS log.

```
options nodate pageno=1 linesize=80 pagesize=60 source;
```

Route the SAS log to an external file. PROC PRINTTO uses the LOG= option to route the SAS log to an external file. By default, this log is appended to the current contents of *log-file*.

```
proc printto log='log-file';
run;
```

Create the NUMBERS data set. The DATA step uses list input to create the NUMBERS data set.

```

data numbers;
    input x y z;
    datalines;
14.2    25.2    96.8
10.8    51.6    96.8
  9.5    34.2   138.2
  8.8    27.6    83.2
11.5    49.4   287.0
  6.3    42.0   170.7
    ;

```

Route the procedure output to an external file. PROC PRINTTO routes output to an external file. Because the LISTING destination must be open in order to route SAS output to an external file, SAS opens the LISTING destination if it is not already open. You do not need to include the ODS LISTING statement. Because

NEW is specified, any output written to *output-file* will overwrite the file's current contents.

```
proc printto print='output-file'
new;
run;
```

Print the NUMBERS data set. The PROC PRINT output is written to the specified external file.

```
proc print data=numbers;
title 'Listing of NUMBERS Data Set';
run;
```

Reset the SAS log and procedure output destinations to default. PROC PRINTTO routes subsequent logs and procedure output to their default destinations and closes both of the current files.

```
proc printto;
run;
```

Log

Example Code 41.1 Portion of Log Routed to the Default Destination

```
01  options nodate pageno=1 linesize=80 pagesize=60 source;
02
03  proc printto log='file-path/em/log1.log';
04  run;
```

```
NOTE: PROCEDURE PRINTTO used (Total process time):
      real time          0.01 seconds
      cpu time           0.00 seconds
```

Example Code 41.2 Portion of Log Routed to an External File

```

NOTE: PROCEDURE PRINTTO used (Total process time):
      real time          0.00 seconds
      cpu time           0.00 seconds

5
6   data numbers;
7       input x y z;
8       datalines;

NOTE: The data set WORK.NUMBERS has 6 observations and 3 variables.
NOTE: DATA statement used (Total process time):
      real time          0.01 seconds
      cpu time           0.01 seconds

15  ;
16
17  proc printto print='print1.out' new;
18  run;

NOTE: Writing HTML Body file: sashtml.htm
NOTE: PROCEDURE PRINTTO used (Total process time):
      real time          4.04 seconds
      cpu time           0.62 seconds

19
20  proc print data=numbers;
21      title 'Listing of NUMBERS Data Set';
22  run;

NOTE: There were 6 observations read from the data set WORK.NUMBERS.
NOTE: PROCEDURE PRINT used (Total process time):
      real time          0.56 seconds
      cpu time           0.09 seconds

23
24  proc printto;
25  run;

```

Output

Output 41.1 Procedure Output Routed to an External File**Listing of NUMBERS Data Set**

Obs	x	y	z
1	14.2	25.2	96.8
2	10.8	51.6	96.8
3	9.5	34.2	138.2
4	8.8	27.6	83.2
5	11.5	49.4	287.0
6	6.3	42.0	170.7

Example 2: Routing to SAS Catalog Entries

Features:

- PRINTTO statement without options
- PRINTTO statement options
 - LABEL=
 - LOG=
 - NEW
 - PRINT=

Details

This example uses PROC PRINTTO to route the SAS log and procedure output to a SAS catalog entry and then to reset both destinations to the default.

Program

```
options source;

libname lib1 'SAS-library';

proc printto log=lib1.cat1.test.log label='Inventory program' new;
run;

data lib1.inventory;
  length Dept $ 4 Item $ 6 Season $ 6 Year 4;
  input dept item season year @@;
  datalines;
3070 20410  spring 2011 3070 20411  spring 2012
3070 20412  spring 2012 3070 20413  spring 2012
3070 20414  spring 2011 3070 20416  spring 2009
3071 20500  spring 2011 3071 20501  spring 2009
3071 20502  spring 2011 3071 20503  spring 2011
3071 20505  spring 2010 3071 20506  spring 2009
3071 20507  spring 2009 3071 20424  spring 2011
;

proc printto print=lib1.cat1.inventory.output
            label='Inventory program' new;
run;

proc report data=lib1.inventory nowindows headskip;
  column dept item season year;
  title 'Current Inventory Listing';
run;

proc printto;
run;
```

Program Description

Set the SAS system options. The SOURCE option specifies to write source statements to the SAS log.

```
options source;
```

Assign a libref.

```
libname lib1 'SAS-library';
```

Route the SAS log to a SAS catalog entry. PROC PRINTTO routes the SAS log to a catalog entry, TEST.LOG, in the LIB1.CAT1 catalog. . LABEL= assigns a description for the catalog entry.

```
proc printto log=lib1.cat1.test.log label='Inventory program' new;
run;
```

Create the LIB1.INVENTORY data set. The DATA step creates a permanent SAS data set.

```
data lib1.inventory;
  length Dept $ 4 Item $ 6 Season $ 6 Year 4;
  input dept item season year @@;
  datalines;
3070 20410  spring 2011 3070 20411  spring 2012
3070 20412  spring 2012 3070 20413  spring 2012
3070 20414  spring 2011 3070 20416  spring 2009
3071 20500  spring 2011 3071 20501  spring 2009
3071 20502  spring 2011 3071 20503  spring 2011
3071 20505  spring 2010 3071 20506  spring 2009
3071 20507  spring 2009 3071 20424  spring 2011
;
```

Route the procedure output to a SAS catalog entry. PROC PRINTTO routes opens the LISTING destination in order to route the procedure output from the subsequent PROC REPORT step to the SAS catalog entry LIB1.CAT1.INVENTORY.OUTPUT. LABEL= assigns a description for the catalog entry. After the procedure output is routed to the SAS catalog, PROC PRINTTO closes the LISTING destination.

```
proc printto print=lib1.cat1.inventory.output
              label='Inventory program' new;
run;

proc report data=lib1.inventory nowindows headskip;
  column dept item season year;
  title 'Current Inventory Listing';
run;
```

Reset the SAS log and procedure output back to the default and close the file. PROC PRINTTO closes the current files that were opened by the previous PROC PRINTTO step and reroutes subsequent SAS logs and procedure output to their default destinations.

```
proc printto;
run;
```

Log

Example Code 41.3 SAS Log Routed to SAS Catalog Entry *lib1.cat1.TEST.LOG*.

```
NOTE: PROCEDURE PRINTTO used (Total process time):
      real time          0.00 seconds
      cpu time           0.00 seconds

49
50  data lib1.inventory;
51      length Dept $ 4 Item $ 6 Season $ 6 Year 4;
52      input dept item season year @@;
53      datalines;

NOTE: SAS went to a new line when INPUT statement reached past the end of a
      line.
NOTE: The data set LIB1.INVENTORY has 14 observations and 4 variables.
NOTE: DATA statement used (Total process time):
      real time          0.03 seconds
      cpu time           0.00 seconds

61  ;
62  ods listing;
63  proc printto print=lib1.cat1.inventory.output
64              label='Inventory program' new;
65  run;

NOTE: PROCEDURE PRINTTO used (Total process time):
      real time          0.00 seconds
      cpu time           0.00 seconds

66
67  proc report data=lib1.inventory nowindows headskip;
68      column dept item season year;
69      title 'Current Inventory Listing';
70  run;

NOTE: There were 14 observations read from the data set LIB1.INVENTORY.
NOTE: PROCEDURE REPORT used (Total process time):
      real time          0.09 seconds
      cpu time           0.04 seconds

71
72  proc printto;
73  run;
NOTE: PROCEDURE PRINTTO used (Total process time):
      real time          0.00 seconds
      cpu time           0.00 seconds

74  ods listing close;
```

Output

Output 41.2 Procedure Output Routed to SAS Catalog Entry
LIB1.CAT1.INVENTORY.OUTPUT.

Current Inventory Listing

Dept	Item	Season	Year
3070	20410	spring	2011
3070	20411	spring	2012
3070	20412	spring	2012
3070	20413	spring	2012
3070	20414	spring	2011
3070	20416	spring	2009
3071	20500	spring	2011
3071	20501	spring	2009
3071	20502	spring	2011
3071	20503	spring	2011
3071	20505	spring	2010
3071	20506	spring	2009
3071	20507	spring	2009
3071	20424	spring	2011

Example 3: Using Procedure Output as an Input File

Features:

- PRINTTO statement without options
- PRINTTO statement options
 - LOG=
 - NEW
 - PRINT=

Details

This example uses PROC PRINTTO to route procedure output to an external file and then uses that file as input to a DATA step.

Program

```
data test;
  do n=1 to 1000;
    x=int(ranuni(77777)*7);
```

```

        y=int(ranuni(77777)*5);
        output;
    end;
run;

filename routed 'output-filename';

proc printto print=routed new;
run;

proc freq data=test;
    tables x*y / chisq;
run;

proc printto print=print;
run;

data probtest;
    infile routed;
    input word1 $ @;
    if word1='Chi-Squa' then
        do;
            input df chisq prob;
            keep chisq prob;
            output;
        end;
run;

proc print data=probtest;
    title 'Chi-Square Analysis for Table of X by Y';
run;

```

Program Description

Generate random values for the variables. The DATA step uses the RANUNI function to randomly generate values for the variables X and Y in the data set A.

```

data test;
    do n=1 to 1000;
        x=int(ranuni(77777)*7);
        y=int(ranuni(77777)*5);
        output;
    end;
run;

```

Assign a fileref and route procedure output to the file that is referenced. The FILENAME statement assigns a fileref to an external file. PROC PRINTTO routes subsequent procedure output to the file that is referenced by the fileref ROUTED. PROC PRINTTO opens the LISTING destination for the duration of routing the procedure option. See PROC FREQ Output Routed to the External File Referenced as ROUTED below.

```

filename routed 'output-filename';

proc printto print=routed new;
run;

```

Produce the frequency counts. PROC FREQ computes frequency counts and a chi-square analysis of the variables X and Y in the data set TEST. This output is routed to the file that is referenced as ROUTED.

```
proc freq data=test;
  tables x*y / chisq;
run;
```

Close the file. You must use another PROC PRINTTO to close the file that is referenced by fileref ROUTED so that the following DATA step can read it. The step also routes subsequent procedure output to the default destination. PRINT= causes the step to affect only procedure output, not the SAS log.

```
proc printto print=print;
run;
```

Create the data set PROBTTEST. The DATA step uses ROUTED, the file containing PROC FREQ output, as an input file and creates the data set PROBTTEST. This DATA step reads all records in ROUTED but creates an observation only from a record that begins with **Chi-Squa**.

```
data probtest;
  infile routed;
  input word1 $ @;
  if word1='Chi-Squa' then
    do;
      input df chisq prob;
      keep chisq prob;
      output;
    end;
run;
```

Print the PROBTTEST data set. PROC PRINT produces a simple listing of data set PROBTTEST. This output is routed to the default destination. See PROC PRINT Output of Data Set PROBTTEST, Routed to Default Destination in the Output section.

```
proc print data=probtest;
  title 'Chi-Square Analysis for Table of X by Y';
run;
```


Output

Output 41.3 PROC FREQ Output Routed to the External File Referenced as ROUTED

The FREQ Procedure

Frequency Percent Row Pct Col Pct	Table of x by y						
	x	y					Total
		0	1	2	3	4	
	0	29	33	12	25	27	126
		2.90	3.30	1.20	2.50	2.70	12.60
		23.02	26.19	9.52	19.84	21.43	
		15.18	16.18	6.25	11.74	13.50	
	1	23	26	29	20	19	117
		2.30	2.60	2.90	2.00	1.90	11.70
		19.66	22.22	24.79	17.09	16.24	
		12.04	12.75	15.10	9.39	9.50	
	2	28	26	32	30	25	141
		2.80	2.60	3.20	3.00	2.50	14.10
		19.86	18.44	22.70	21.28	17.73	
		14.66	12.75	16.67	14.08	12.50	
	3	26	24	36	32	45	163
		2.60	2.40	3.60	3.20	4.50	16.30
		15.95	14.72	22.09	19.63	27.61	
		13.61	11.76	18.75	15.02	22.50	
	4	25	31	28	36	29	149
		2.50	3.10	2.80	3.60	2.90	14.90
		16.78	20.81	18.79	24.16	19.46	
		13.09	15.20	14.58	16.90	14.50	
	5	32	29	26	33	27	147
		3.20	2.90	2.60	3.30	2.70	14.70
		21.77	19.73	17.69	22.45	18.37	
		16.75	14.22	13.54	15.49	13.50	
	6	28	35	29	37	28	157
		2.80	3.50	2.90	3.70	2.80	15.70
		17.83	22.29	18.47	23.57	17.83	
		14.66	17.16	15.10	17.37	14.00	
	Total	191	204	192	213	200	1000
		19.10	20.40	19.20	21.30	20.00	100.00

Statistics for Table of x by y

Statistic	DF	Value	Prob
Chi-Square	24	27.2971	0.2908
Likelihood Ratio Chi-Square	24	28.1830	0.2524
Mantel-Haenszel Chi-Square	1	0.6149	0.4330
Phi Coefficient		0.1652	
Contingency Coefficient		0.1630	
Cramer's V		0.0826	

Sample Size = 1000

Output 41.4 PROC PRINT Output of Data Set PROBTTEST, Routed to the Default Destination

Chi-Square Analysis for Table of X by Y

Obs	chisq	prob
1	27.2971	0.2908

Example 4: Routing to a Printer

Features: PRINTTO statement option
 PRINT=

Details

This example uses PROC PRINTTO to route procedure output directly to a printer.

Program

```
options nodate pageno=1 linesize=80 pagesize=60;

filename your_fileref printer
'printer-name';

proc printto print=your_fileref;
run;
```

Program Description

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=60;
```

Associate a fileref with the printer name. The FILENAME statement associates a fileref with the printer name that you specify. If you want to associate a fileref with the default printer, omit *'printer-name'*.

```
filename your_fileref printer
'printer-name';
```

Specify the file to route to the printer. The PRINT= option specifies the file that PROC PRINTTO routes to the printer.

```
proc printto print=your_fileref;
run;
```


PROTO Procedure

Overview: PROTO Procedure	1519
What Does the PROTO Procedure Do?	1520
Concepts: PROTO Procedure	1520
Registering Function Prototypes	1520
Supported C Return Types	1520
Supported C Argument Types	1521
C Structures in SAS	1522
Syntax: PROTO Procedure	1528
PROC PROTO Statement	1528
LINK Statement	1530
MAPMISS Statement	1531
FUNCTION-PROTOTYPE-N Statement	1532
Usage: PROTO Procedure	1534
Basic C Language Types	1534
Working with Character Variables	1534
Working with Numeric Variables	1535
Working with Missing Values	1535
Function Names	1535
Interfacing with External C Functions	1536
Scope of Packages in PROC PROTO	1539
C Helper Functions and CALL Routines	1541
Example: Splitter Function Example	1544

Overview: PROTO Procedure

What Does the PROTO Procedure Do?

The PROTO procedure enables you to register, in batch mode, external functions that are written in the C or C++ programming languages. You can use these functions in SAS as well as in C-language structures and types. After the C-language functions are registered in PROC PROTO, they can be called from any SAS function or subroutine that is declared in the FCMP procedure.

Concepts: PROTO Procedure

Registering Function Prototypes

Function prototypes are registered (declared) in the PROTO procedure. For more information, see [“FUNCTION-PROTOTYPE-N Statement” on page 1532](#).

Supported C Return Types

The following C return types are supported in the PROTO procedure.

Table 42.1 Supported C Return Types

Function Prototype	SAS Variable Type	C Variable Type
short	numeric	short, short *, short **
short *	numeric, array	short, short *, short **
int	numeric	int, int *, int **

Function Prototype	SAS Variable Type	C Variable Type
int *	numeric, array	int, int *, int **
long	numeric	long, long *, long **
long *	numeric, array	long, long *, long **
double	numeric	double, double *, double **
double *	numeric, array	double, double *, double **
char *	character	char *, char **
struct *	struct	struct *, struct **
void		void

Supported C Argument Types

The following C argument types are supported in the PROTO procedure.

Table 42.2 Supported C Argument Types

Function Prototype	SAS Variable Type	C Variable Type
short	numeric	short, short *, short **
short *	numeric, array	short, short *, short **
short **	array	short *, short **
int	numeric	int, int *, int **
int *	numeric, array	int, int *, int **
int **	array	int *, int **
long	numeric	long, long *, long **
long *	numeric, array	long, long *, long **
long **	array	long *, long **

Function Prototype	SAS Variable Type	C Variable Type
double	numeric	double, double *, double **
double *	numeric, array	double, double *, double **
double **	array	double *, double **
char *	character	char *, char **
char **	character	char *, char **
struct *	structure	struct *, struct **
struct **	structure	struct *, struct **

C Structures in SAS

Basic Concepts

Many C language libraries contain functions that have structure pointers as arguments. In SAS, structures can be defined only in PROC PROTO. After being defined, they can be declared and instantiated within many PROC PROTO compatible procedures, such as PROC COMPILE.

A C structure is a template that is applied to a contiguous piece of memory. Each entry in the template is given a name and a type. The type of each element determines the number of bytes that are associated with each entry and how each entry is to be used. Because of various alignment rules and base type sizes, SAS relies on the current machine compiler to determine the location of each entry in the memory of the structure.

Declaring and Referencing Structures in SAS

The syntax of a structure declaration in SAS is the same as for C non-pointer structure declarations. A structure declaration has the following form:

struct *structure_name* *structure_instance*;

Each structure is set to zero values at declaration time. The structure retains the value from the previous pass through the data to start the next pass.

Structure elements are referenced by using the static period (.) notation of C. There is no pointer syntax for SAS. If a structure points to another structure, the only way to reference the structure that is pointed to is by assigning the pointer to a declared structure of the same type. You use that declared structure to access the elements.

If a structure entry is a short, int, or long type, and it is referenced in an expression, it is first cast to a double type and then used in the calculations. If a structure entry is a pointer to a base type, then the pointer is dereferenced and the value is returned. If the pointer is null, then a missing value is returned. The missing value assignments that are made in the PROC PROTO code are used when conversions fail or when missing values are assigned to non-double structure entities.

The length of arrays must be known to SAS so that an array entry in a structure can be used in the same way as an array in SAS, as long as its dimension is declared in the structure. This requirement includes arrays of short, int, and long types. If the entry is actually a pointer to an array of a double type, then the array elements can be accessed by assigning that pointer to a SAS array. Pointers to arrays of other types cannot be accessed by using the array syntax.

Structure Example

```
proc proto package =
work.mylib.struct
  label = "package of structures";

  #define MAX_IN 20;

  typedef char * ptr;
  struct foo {
    double hi;
    int mid;
    ptr buf1;
    long * low;
    struct {
      short ans[MAX_IN + 1];
      struct { /* inner */
        int inner;
      } n2;
      short outer;
    } n;
  };

  typedef struct foo *str;

  struct foo2 {
    str tom;
  };

  str get_record(char *name, int userid);
run;

proc fcmp library = work.mylib;
  struct foo result;
```

```

        result = get_record("Mary", 32);
        put result=;
run;

```

Enumerations in SAS

Enumerations are mnemonics for integer numbers. Enumerations enable you to set a literal name as a specific number and aid in the readability and supportability of C programs. Enumerations are used in C language libraries to simplify the return codes. After a C program is compiled, you can no longer access enumeration names.

Enumerated Types Example

The following example shows how to set up two enumerated value types in PROC PROTO: YesNoMaybeType, and Tens. Both are referenced in the structure ESTRUCTURE:

```

proc proto package=work.mylib.str2
    label="package of structures";

    #define E_ROW 52;
    #define L_ROW 124;
    #define S_ROW 15;

    typedef double ExerciseArray[S_ROW][2];
    typedef double LadderArray[L_ROW];
    typedef double SamplingArray[S_Row];

    typedef enum
    {
        True, False, Maybe
    } YesNoMaybeType;

    typedef enum {
        Ten=10, Twenty=20, Thirty=30, Forty=40, Fifty=50
    } Tens;

    typedef struct {
        short      rows;
        short      cols;
        YesNoMaybeType type;
        Tens        dollar;
        ExerciseArray  dates;
    } EStructure;
run;

```

The following PROC FCMP example shows how to access these enumerated types. In this example, the enumerated values that are set up in PROC PROTO are implemented in SAS as macro variables. Therefore, they must be accessed using the & symbol:

```

proc fcmp library=work.mylib;

```

```

struct EStructure mystruct;

mystruct.type=&True;
mystruct.dollar=&Twenty;
run;

```

C-Source Code in SAS

You can use PROC PROTO in a limited way to compile external C functions. The C source code can be specified in PROC PROTO in the following way:

```

C-function-prototype;
externc function-name;
... C-source-statements ...
externcend;

```

The function name tells PROC PROTO which function's source code is specified between the EXTERNC and EXTERNCEND statements. When PROC PROTO compiles source code, it includes any structure definitions and C function prototypes that are currently declared. However, typedef and #define are not included.

This functionality is provided to enable the creation of simple “helper” functions that facilitate the interface to preexisting external C libraries. Any valid C statement is permitted except for the #include statement. Only a limited subset of the C-stdlib functions is available. However, you can call any other C function that is already declared within the current PROC PROTO step.

The following C-stdlib functions are supported:

Table 42.3 Supported stdlib Functions

Function	Description
double sin(double x)	returns the sine of x (radians)
double cos(double x)	returns the cosine of x (radians)
double tan(double x)	returns the tangent of x (radians)
double asin(double x)	returns the arcsine of x (-pi/2 to pi/2 radians)
double acos(double x)	returns the arccosine of x (0 to pi radians)
double atan(double x)	returns the arctangent of x (-pi/2 to pi/2 radians)
double atan2(double x, double y)	returns the arctangent of y/x (-pi to pi radians)
double sinh(double x)	returns the hyperbolic sine of x (radians)

Function	Description
double cosh(double x)	returns the hyperbolic cosine of x (radians)
double tanh(double x)	returns the hyperbolic tangent of x (radians)
double exp(double x)	returns the exponential value of x
double log(double x)	returns the logarithm of x
double log2(double x)	returns the logarithm of x base-2
double log10(double x)	returns the logarithm of x base-10
double pow(double x, double y)	returns x raised to the y power of x**y
double sqrt(double x)	returns the square root of x
double ceil(double x)	returns the smallest integer not less than x
double fmod(double x, double y)	returns the remainder of (x/y)
double floor(double x)	returns the largest integer not greater than x
int abs(int x)	returns the absolute value of x
double fabs(double)	returns the absolute value of x
int min(int x, int y)	returns the minimum of x and y
double fmin(double x, double y)	returns the minimum of x and y
int max(int x, int y)	returns the maximum of x and y
double fmax(double x, double y)	returns the maximum of x and y
char* malloc(int x)	allocates memory of size x
void free(char*)	frees memory allocated with malloc

The following example shows a simple C function written directly in PROC PROTO:

```
proc proto
package=work.mylib.foo;
  struct mystruct {
    short a;
    long b;
  };
  int fillMyStruct(short a, short b,
    struct mystruct * s);
```

```

externc fillMyStruct;
int fillMyStruct(short a, short b,
struct mystruct * s) {
    s ->a = a;
    s ->b = b;
    return(0);
}
externcend;
run;

```

Limitations for C Language Specifications

The limitations for the C language specifications in the PROTO procedure are as follows:

- #define statements must be followed by a semicolon (;) and must be numeric in value.
- The #define statement functionality is limited to simple replacement and unnested expressions. The only symbols that are affected are array dimension references.
- The C preprocessor statements #include and #if are not supported. The SAS macro %INC can be used in place of #include.
- A maximum of two levels of indirection are allowed for structure elements. Elements like "double ****" are not allowed. If these element types are needed in the structure, but are not accessed in SAS, you can use placeholders.
- The float type is not supported.
- Unsigned is the only type specifier that is currently supported.
- A specified bit size for structure variables is not supported.
- Function pointers and definitions of function pointers are not supported.
- The union type is not supported. However, if you plan to use only one element of the union, you can declare the variable for the union as the type for that element.
- All non-pointer references to other structures must be defined before they are used.
- You cannot use the ENUM key word in a structure. In order to specify ENUM in a structure, use the TYPEDEF key word.
- Structure elements with the same alphanumeric name but with different cases (for example, ALPHA, Alpha, and alpha) are not supported. SAS is not case-sensitive. Therefore, all structure elements must be unique when compared in a case-insensitive program.

Syntax: PROTO Procedure

See: [“PROTOLIBS System Option” in SAS System Options: Reference](#)

```
PROC PROTO PACKAGE=entry <options>;
  MAPMISS type1=value1 type2=value2 ...;
  LINK load-module <NOUNLOAD>;
  function-prototype-1 <function-prototype-n ...;>
```

Statement	Task	Example
PROC PROTO	Register, in batch mode, external functions that are written in the C or C++ programming languages	Ex. 1
LINK	Specify the name, path, and load module that contains your functions	
MAPMISS	Specify alternative values, by type, to pass to functions if values are missing	
FUNCTION-PROTOTYPE-N	Registers function prototypes in the PROTO procedure.	

PROC PROTO Statement

Register, in batch mode, external functions that are written in the C or C++ programming languages.

Example: [“Example: Splitter Function Example” on page 1544](#)

Syntax

```
PROC PROTO PACKAGE=entry <options>;
```

Summary of Optional Arguments

ENCRYPT
HIDE

for XML databases only, enables the code to be encoded within a data set.

LABEL=*package-label*

specifies a text string to describe or label a package.

NOSIGNALS

specifies that none of the functions in a package will produce exceptions.

STDCALL

for Windows PC platforms only, indicates that functions be called using the "_stdcall" convention.

STRUCTPACK

for 32 bit Windows PC platforms or Power Architecture platform running on Linux only, specifies that all structures in a package be compiled with a specific N-BYTE packing pragma.

Required Arguments

PACKAGE=*entry*

specifies the SAS entry where the prototype information is saved. *Entry* is a three-level name having the following form: *library.dataset.package*.

Package enables you to specify grouping in the GUI.

function-prototype-1

function-prototype-n

contains the C code of the function prototypes.

Optional Arguments

ENCRYPT | HIDE

specifies that encoding within a database is allowed.

Restriction This option is available for XML databases only.

LABEL=*package-label*

specifies a text string that is used to describe or label the package. The maximum length of the label is 256 characters.

NOSIGNALS

specifies that none of the functions in a package will produce exceptions or signals.

STDCALL

for Windows PC platforms only, indicates that all functions in the package are called using the "_stdcall" convention.

STRUCTPACK

for the Power Architecture platform running on Linux only, specifies that all structures in this package were compiled with the given N-BYTE packing pragma. That is, STRUCTURE4 specifies that all structures in the package were compiled with the "#pragma pack(4)" option.

Alias PACK

LINK Statement

Specifies the name of the load module that contains your functions. Specifying the path is optional.

Syntax

LINK *load-module* <NOUNLOAD>;

Required Argument

load-module

specifies the load module that contains your functions. You can add more LINK statements to include as many libraries as you need for your prototypes. *Load-module* can have the following forms, depending on your operating environment:

```
'c:\mylibs\xxx.dll';
```

```
'c:\mylibs\xxx';
```

```
'/users/me/mylibs/xxx';
```

Tip Full pathname specification is the safest and recommended way to link your modules with the PROTO procedure.

Optional Argument

NOUNLOAD

IMPORTANT Use the NOUNLOAD option to prevent the PROTO procedure from crashing due to runtime incompatibilities in certain libraries.

specifies that selected libraries remain loaded when the SAS session ends.

Details

Specifying Extensions and Functions

You do not need to specify your module's extension. SAS loads your module with the extension that is specific to your operating environment.

All functions must be declared externally in your load module so that SAS can find them. For most platforms, external declaration is the default behavior for the compiler. However, many C compilers do not export function names by default. The

following examples show how to declare your functions for external loading for most PC compilers:

```
_declspec(dllexport) int myfunc(int, double);
_declspec(dllexport) int price2(int a, double foo);
```

The PROTOLIBS System Option

CAUTION

Security Issue Calling a third-party library to a valid path for a load library that is set by the LINK statement can cause a security issue.

SAS administrator at your site can use the PROTOLIBS system option to control the function of the LINK statement. The administrator can use the PROTOLIBS option to specify a list of valid paths where modules that are specified by the LINK statement can be loaded. The administrator can also use the PROTOLIBS option to disable the LINK statement.

The PROTOLIBS option is a restricted option. Only the SAS administrator at your site can specify a value for it. The default value of the PROTOLIBS option is NONE, which means that the LINK statement is disabled.

You can use the VALUE argument of PROC OPTIONS to find where you can register load modules.

MAPMISS Statement

Specifies alternative values, by type, to pass to functions if values are missing.

Syntax

```
MAPMISS <POINTER=NULL > <CHARPOINTER=NULL> <INT=integer-value >
<DOUBLE=double-value>
< LONG=long-value > <SHORT=short-value>;
```

Optional Arguments

POINTER=NULL

specifies to pass NULL pointers to functions for missing numeric values.

Default null

CHARPOINTER=NULL

Default null

Restriction The CHARPOINTER option is available starting with the 2025.02 release.

specifies to pass NULL pointers to functions for missing character values

INT=*integer-value*

specifies an integer value to pass to functions for integer values that are missing.

DOUBLE=*double-value*

specifies a double value to pass to functions for double values that are missing.

LONG=*long-value*

specifies a long value to pass to functions for long values that are missing.

SHORT=*short-value*

specifies a short value to pass to functions for short values that are missing.

Details

The MAPMISS statement is used to specify alternative values, by data type or pointer value. These values are passed to functions if values are missing. The values are specified as arguments in the MAPMISS statement.

If you set POINTER=NULL or CHARPOINTER=NULL, a null value pointer is passed to the functions for pointer variables that are missing. If you do not specify a mapping for a type that is used as an argument to a function, the function is not called when a numeric argument of that type is missing.

MAPMISS values have no affect on arrays because array elements are not checked for missing values when they are passed as parameters to C functions.

FUNCTION-PROTOTYPE-N Statement

Registers function prototypes in the PROTO procedure.

Syntax

```
function-prototype-1 <function-prototype-n ...> return-type function-name
(argument-type <argument-name> / <iotype>
<argument-label>, ...) <options>;
```

Required Arguments

return-type

specifies a C language type for the returned value.

Tip *Return-type* can be preceded by either the UNSIGNED or EXCELDATE modifiers. You need to use EXCELDATE if the return type is a Microsoft Excel date.

See [“Supported C Return Types” on page 1520](#)

function-name

specifies the name of the function to be registered.

Tip Function names within a given package must be unique in the first 32 characters.

argument-type

specifies the C language type for the function argument.

You must specify *argument-type* for each argument in the function's argument list. The argument list must be enclosed in parentheses. If the argument is an array, then you must specify the argument name prefixed to square brackets that contain the array size (for example, `double A[10]`). If the size is not known or if you want to disable verification of the length, then use `type*name` instead (for example, `double*A`).

Tip *Argument-type* can be preceded by either the UNSIGNED, CONST, or EXCELDATE modifiers. You need to use EXCELDATE if the return type is a Microsoft Excel date.

See [“Supported C Argument Types” on page 1521](#)

argument-name

specifies the name of the argument.

iotype

specifies the I/O type of the argument. Use I for input, O for output, and U for update.

Alias IO

Tips In the program code, use `IOTYPE=I | O | U`.

By default, all parameters that are pointers are assumed to be input type U. All non-pointer values are assumed to be input type I. This behavior parallels the C language parameter passing scheme.

See [“Example: Splitter Function Example” on page 1544](#) for an example of how *iotype* is used.

argument-label

specifies a description or label for the argument.

LABEL="text-string"

specifies a description or a label for the function. Enclose the text string in quotation marks.

Note The LABEL option can be used with the PROTO procedure.

KIND | GROUP=group-type

specifies the group that the function belongs to. The KIND= or GROUP= option allows for convenient grouping of functions in a package.

You can use any string (up to 40 characters) in quotation marks to group similar functions.

Note The KIND or GROUP option can be used with the PROTO procedure.

Tip The following special cases provided for Risk Dimensions do not require quotation marks: INPUT (Instrument Input), TRANS (Risk Factor Transformation), PRICING (Instrument Pricing), and PROJECT. The default is PRICING.

Usage: PROTO Procedure

Basic C Language Types

The SAS language supports two data types: character and numeric. These types correspond to an array of characters and a double (double-precision floating point) data type in the C programming language. When SAS variables are used as arguments to external C functions, they are converted (cast) into the proper types.

Working with Character Variables

You can use character variables for arguments that require a "char *" value only. The character string that is passed is a null string that is terminated at the current length of the string. The current length of the character string is the minimum of the allocated length of the string and the length of the last value that was stored in the string. The allocated length of the string (by default, 32 bytes) can be specified by using the LENGTH statement. Functions that return "char *" can return a null or zero-delimited string that is copied to the SAS variable. If the current length of the character string is less than the allocated length, the character string is padded with blanks.

In the following example, the allocated length of *str* is 10, but the current length is 5. When the string is null-terminated at the allocated length, "hello " is passed to the function xxx:

```
length str $ 10;
str = "hello";
call xxx(str);
```

To avoid the blank padding, use the SAS TRIM function on the parameter within the function call:

```
length str $ 10;
```

```
str = "hello";  
call xxx(trim(str));
```

In this case, the value "hello" is passed to the function xxx.

Working with Numeric Variables

You can use numeric variables for an argument that requires a short, int, long, or double data type, as well as for pointers to those types. Numeric variables are converted to the required type automatically. If the conversion fails, then the function is not called and the output to the function is set to missing. If pointers to these types are requested, the address of the converted value is passed. On return from the call, the value is converted back to a double type and stored in the SAS variable. SAS scalar variables cannot be passed as arguments that require two or more levels of indirection. For example, a SAS variable cannot be passed as an argument that requires a cast to a "long ***" type.

Working with Missing Values

SAS variables that contain missing values are converted according to how the function that is being called has mapped missing values when using the PROTO procedure. All variables that are returned from the function are checked for the mapped missing values and converted to SAS missing values.

For example, if an argument to a function is missing, and the argument is to be converted to an integer, and an integer was mapped to -99, then -99 is passed to the function. If the same function returns an integer with the value -99, then the variable that this value is returned to would have a value of missing.

Function Names

External functions and FCMP functions can have the same name as long as they are saved to different packages. When these packages are loaded, a warning message in the log identifies which package contains the default definition for a given function. To use a function definition from a package other than the default, call the function using *package-name.function-name*.

When you load multiple packages of external functions, all function names must be unique. If two or more external functions of the same name are loaded, the first function that is loaded will be used. Duplicate external functions are ignored. A warning message in the log indicates which package contains the function that will be used, and which package contained the discarded definition.

Interfacing with External C Functions

To make it easier to interface with external C functions, many PROTO-compatible procedures have been enhanced to support most of these C types.

In working with arrays, it is important to note that SAS arrays are passed to EXTERNC routines as single dimensional arrays. They will be read as single dimensional arrays when returning to an FCMP function.

The following example shows log output that is generated when you use arrays and the double ** variable type.

```
proc proto package=work.proto_ds.test;
  void idmat(double** a,int m,int n);
  externc idmat;
  void idmat(double** a,int m,int n)
  {
    int i=0,j=0;
    for(i=0;i<m;i++)
    {
      for(j=0;j<n;j++)
      {
        if(i==j)
          a[0][i*m+j]=1;
        else
          a[0][i*m+j]=0;
      }
    }
  };
externcend;
run;

proc fcmp outlib=work.proto_ds.test inlib=work.proto_ds;
  subroutine sas_idmat(b[*,*]);
    outargs b;
    m=dim(b,1);
    n=dim(b,2);
    call idmat(b,m,n);
    put b=;
    put ' ';
  endsub;
quit;
run;

options cmplib=work.proto_ds;

data cmat;
  array ctmp[3,3] _temporary_;
  array c[3,3];
  output;
  call sas_idmat(ctmp);
  do i=1 to dim(c,1);
```

```

do j=1 to dim(c,2);
  c[i,j]=ctmp[i,j];
end;
end;
output;
drop i j;
run;

```

SAS writes the following results to the log:

```

b[1, 1]=1 b[1, 2]=0 b[1, 3]=0 b[2, 1]=0 b[2, 2]=1 b[2, 3]=0 b[3, 1]=0 b[3, 2]=0 b[3, 3]=1

```

There is no way to return and save a pointer to any type in a SAS variable. Pointers are always dereferenced, and their contents are converted and copied to SAS variables.

The EXTERNC statement is used to specify C variables in PROTO compatible procedures. The syntax of the EXTERNC statement has the following form:

EXTERNC DOUBLE | INT | LONG | SHORT | CHAR <[*][*]> var-1 <var-2 ... var-n>;

The following table (Table 42.89 on page 1537) shows how these variables are treated when they are positioned on the left side of an expression. The table shows the automatic casting that is performed for a short type on the right side of an assignment. (Explicit type conversions can be forced in any expression, with a unary operator called a cast.) The table lists all the allowed combinations of short types that are associated with SAS variables.

Note: A table for int, long, and double types can be created by substituting any of these types for "short" in this table.

If any of the pointers are null and require dereferencing, then the result is set to missing if there is a missing value set for the result variable. For more information, see "MAPMISS Statement" on page 1531.

Table 42.4 Automatic Type Casting for the short Data Type in an Assignment Statement

Type for Left Side of Assignment	Type for Right Side of Assignment	Cast Performed
short	SAS numeric	y = (short) x
short	short	y = x
short	short *	y = * x
short	short **	y = ** x
short *	SAS numeric	* y = (short) x

Type for Left Side of Assignment	Type for Right Side of Assignment	Cast Performed
short *	short	y = & x
short *	short *	y = x
short *	short **	y = * x
short **	SAS numeric	**y = (short) x
short **	short *	y = & x
short **	short **	y = x
SAS numeric	short	y = (double) x
SAS numeric	short *	y = (double) * x
SAS numeric	short **	y = (double) ** x

The following table shows how these variables are treated when they are passed as arguments to an external C function.

Table 42.5 Types That Are Allowed for External C Arguments

Function Prototype	SAS Variable Type	C Variable Type
short	numeric	short, short *, short **
short *	numeric, array	short, short *, short **
short **	array	short *, short **
int	numeric	int, int *, int **
int *	numeric, array	int, int *, int **
int **	array	int *, int **
long	numeric	long, long *, long **
long *	numeric, array	long, long *, long **
long **	array	long *, long **

Function Prototype	SAS Variable Type	C Variable Type
double	numeric	double, double *, double **
double *	numeric, array	double, double *, double **
double **	array	double *, double **
char *	character	char *, char **
char **	character	char *, char **
struct *	structure	struct *, struct **
struct **	structure	struct *, struct **

Note: Automatic conversion between two different C types is never performed.

Scope of Packages in PROC PROTO

PROC PROTO packages are loaded in the order that is specified in the CMLIB= system option, and the contents are used globally. Packages that are loaded through a PROC statement option are considered local in scope. Local definitions are loaded last, and in all cases, local scope overrides global scope.

Definitions are loaded regardless of whether they have unique names or duplicate names. Multiple definitions of certain PROC PROTO elements (for example, enumeration names and function prototypes) can cause name conflicts and generate errors. To prevent name conflicts between PROC PROTO packages, ensure that elements such as enumerated types and function definitions have unique names.

The following example loads three PROC PROTO packages, and shows how the order in which typedef and #define statements override one another.

This part of the example loads the first two PROC PROTO packages:

```
proc proto package = work.pl.test1;
  typedef struct { int a; int b; } AB_t;
  #define NUM 1;
  int p1(void);
  externc p1;
  int p1(void)
  {
    return NUM;
  }
endproc;
```

```

    }
    externcend;
run;

proc proto package = work.p2.test2;
    typedef struct { int a; int b; } AB_t;
    #define NUM 2;
    int p2(void);
    externc p2;
    int p2(void)
    {
        return NUM;
    }
    externcend;
run;

options CMPLIB = (work.p1 work.p2);

proc fcmp;
    x = p1();
    put "Should be 2: " x=;
run;

```

The result from executing the programs above is 2, because the packages are loaded in order.

In the following example, PROC PROTO adds a third package and includes it in PROC FCMP locally, keeping the CMPLIB= system option set as above:

```

proc proto package = work.p3.test3;
    typedef struct { int a; int b; } AB_t;
    #define NUM 3;
    int p3(void);
    externc p3;
    int p3(void)
    {
        return NUM;
    }
    externcend;
run;

proc fcmp libname = work.p3;
    x = p1();
    put "Should be 3: " x=;
run;

```

In this example, the local definition of NUM in *work.p3* is used instead of the global definitions that are loaded through *work.p1* and *work.p2*.

C Helper Functions and CALL Routines

What Are C Helper Functions and CALL Routines?

Several helper functions and CALL routines are provided with the package to handle C-language constructs in PROC FCMP. Most C-language constructs must be defined in a catalog package that is created by PROC PROTO before the constructs can be referenced or used by PROC FCMP. The ISNULL function and the STRUCTINDEX and SETNULL CALL routines have been added to extend the SAS language to handle C-language constructs that do not naturally fit into the SAS language.

The following C helper functions and CALL routines are available:

Table 42.6 C Helper Functions and CALL Routines

C Helper Function or CALL Routine	Description
“ISNULL C Helper Function” on page 1541	Determines whether a pointer element of a structure is null
“SETNULL C Helper CALL Routine” on page 1542	Sets a pointer element of a structure to null
“STRUCTINDEX C Helper CALL Routine” on page 1542	Enables you to access each structure element in an array of structures

ISNULL C Helper Function

The ISNULL function determines whether a pointer element of a structure is NULL. The function has the following form:

double **ISNULL** (*pointer-element*);

Pointer-element refers to the pointer element.

In the following example, the LINKLIST structure and the GET_LIST function are defined by using PROC PROTO. The GET_LIST function is an external C routine that generates a linked list with as many elements as requested:

```
struct linklist{
    double value;
    struct linklist * next;
```

```
};

struct linklist * get_list(int);
```

The following example shows how to use the ISNULL helper function to loop over the linked list that is created by the GET_LIST function:

```
struct linklist list;

list = get_list(3);
put list.value=;

do while (^isnull(list.next));
    list = list.next;
    put list.value=;
end;
```

Example Code 42.1 *Looping over a Linked List*

```
LIST.value=0
LIST.value=1
LIST.value=2
```

SETNULL C Helper CALL Routine

The SETNULL CALL routine sets a pointer element of a structure to null. It has the following form:

CALL SETNULL(*pointer-element*);

Pointer-element is a pointer to a structure.

If you specify a variable that has a pointer value (a structure entry), then SETNULL sets the pointer to null:

```
call setnull(list.next);
```

The following example assumes a LINKLIST structure that is defined using PROC PROTO. The SETNULL CALL routine can be used to set the next element to null:

```
proc proto;
    struct linklist list;
    call setnull(list.next);
run;
```

STRUCTINDEX C Helper CALL Routine

The STRUCTINDEX CALL routine enables you to access each structure element in an array of structures. When a structure contains an array of structures, you can access each structure element of the array by using the STRUCTINDEX CALL routine. The STRUCTINDEX CALL routine has the following form:

CALL STRUCTINDEX(*struct_array*, *index*, *struct_element*);

Struct_array specifies an array; *index* is a 1-based index as used in SAS arrays; and *struct_element* points to an element in the array.

The following example consists of two parts. Copy and paste the two parts of the example into your SAS editor, and run them as one SAS program.

In the first part of this example, the following structures and function are defined using PROC PROTO:

```
options cmlib=(work.proto_ds work.fcmp_ds);
proc proto package=work.proto_ds.cfcns;
    struct POINT {
        short s;
        int i;
        long l;
        double d;
    };
    struct POINT_ARRAY {
        int length;
        struct POINT * p;
        char name[32];
    };
    struct POINT * struct_array( int );

    externc struct_array;
        struct POINT * struct_array( int num ) {
            return(malloc(sizeof(struct POINT) * num));
        }
    externcend;
run;
```

In the second part of this example, the PROC FCMP code segment shows how to use the STRUCTINDEX CALL routine to get and set each POINT structure element of an array called P in the POINT_ARRAY structure:

```
proc fcmp;
    struct point_array pntarray;
    struct point pnt;

    /* Call struct_array to allocate an array of 2 POINT structures. */
    pntarray.p = struct_array(2);
    pntarray.plen = 2;
    pntarray.name = "My funny structure";

    /* Get each element using the STRUCTINDEX CALL routine and set
    values. */
    do i = 1 to 2;
        call structindex(pntarray.p, i, pnt);
        put "Before setting the" i "element: " pnt=;
        pnt.s = 1;
        pnt.i = 2;
        pnt.l = 3;
        pnt.d = 4.5;
        put "After setting the" i "element: " pnt=;
    end;
```

```
run;
```

Example Code 42.2 Results from the STRUCTINDEX CALL Routine

```
Before setting the 1 element: PNT {s=0, i=0, l=0, d=0}
After setting the 1 element: PNT {s=1, i=2, l=3, d=4.5}
Before setting the 2 element: PNT {s=0, i=0, l=0, d=0}
After setting the 2 element: PNT {s=1, i=2, l=3, d=4.5}
```

Example: Splitter Function Example

Features:

- INT statements
- KIND= prototype argument
- PROC FCMP

Details

This example shows how to use PROC PROTO to prototype two external C language functions called SPLIT and CASHFLOW. These functions are contained in the two shared libraries that are specified by the LINK statements.

Program

```
options nodate pageno=1 linesize=80 pagesize=40;

proc proto package =
  sasuser.myfuncs.mathfun
    label = "package of math functions";

    link "link-library";
    link "link-library";

    int split(int x "number to split")
      label = "splitter function" kind=PRICING;

    int cashflow(double amt, double rate, int periods,
      double * flows / iotype=0)
      label = "cash flow function" kind=PRICING;

run;

proc fcmp libname=sasuser.myfuncs;
  array flows[20];
  a = split(32);
  put a;
```

```

        b = cashflow(1000, .07, 20, flows);
        put b;
        put flows;
run;

run;

```

Program Description

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGNO= specifies the starting page number. LINESIZE= specifies the output line length. PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=40;
```

Specify the catalog entry where the function package information is saved. The catalog entry is a three-level name.

```

proc proto package =
  sasuser.myfuncs.mathfun
    label = "package of math functions";

```

Specify the libraries that contain the SPLIT and CASHFLOW functions. You can add more LINK statements to include as many libraries as you need for your prototypes.

```

link "link-library";
link "link-library";

```

Prototype the SPLIT function. The INT statement prototypes the SPLIT function and assigns a label to the function.

```

int split(int x "number to split")
  label = "splitter function" kind=PRICING;

```

Prototype the CASHFLOW function. The INT statement prototypes the CASHFLOW function and assigns a label to the function.

```

int cashflow(double amt, double rate, int periods,
             double * flows / iotype=0)
  label = "cash flow function" kind=PRICING;

```

Execute the PROTO procedure. The RUN statement executes the PROTO procedure.

```
run;
```

Call the SPLIT and CASHFLOW functions. PROC FCMP calls the SPLIT and CASHFLOW functions. Output from PROC FCMP is created.

```

proc fcmp libname=sasuser.myfuncs;
  array flows[20];
  a = split(32);
  put a;
  b = cashflow(1000, .07, 20, flows);
  put b;
  put flows;

```

```
run;
```

Execute the FCMP procedure. The RUN statement executes the FCMP procedure.

```
run;
```

Output: Prototyping Functions

Output 42.1 *Results from Prototyping the SPLIT and CASHFLOW Functions*

```

                                The SAS System                                1

                                The FCMP Procedure

16
12
70 105 128.33333333 145.83333333 159.83333333 171.5 181.5 190.25 198.02777778
205.02777778
211.39141414 217.22474747 222.60936286 227.60936286 232.27602953 236.65102953
240.76867658
244.65756547 248.341776 251.841776

```


PRTDEF Procedure

Overview: PRTDEF Procedure	1547
What Does the PRTDEF Procedure Do?	1547
Syntax: PRTDEF Procedure	1548
PROC PRTDEF Statement	1548
Usage: PRTDEF Procedure	1550
Input Data Set Variables That Are Used To Create Printer Definitions	1550
Examples: PRTDEF Procedure	1556
Example 1: Defining Multiple Printer Definitions	1556
Example 2: Creating a Ghostview Printer in SASUSER to Preview PostScript Printer Output in SASUSER	1557
Example 3: Creating a Single Printer Definition That Is Available to All Users ...	1559
Example 4: Adding, Modifying, and Deleting Printer Definitions	1561
Example 5: Deleting a Single Printer Definition	1563

Overview: PRTDEF Procedure

What Does the PRTDEF Procedure Do?

The PRTDEF procedure creates printer definitions in batch mode either for an individual user or for all SAS users at your site. Your system administrator can create printer definitions in the SAS registry and make these printers available to all SAS users at your site by using PROC PRTDEF with the USESASHELP option. An individual user can create personal printer definitions in the SAS registry by using PROC PRTDEF.

See Also

[Chapter 44, “PRTEXP Procedure,” on page 1567](#)

Syntax: PRTDEF Procedure

PROC PRTDEF *<options>*;

Statement	Task	Example
PROC PRTDEF	Create printer definitions in batch mode either for an individual user or for all SAS users at your site	Ex. 1 , Ex. 2 , Ex. 3 , Ex. 4 , Ex. 5

PROC PRTDEF Statement

Creates printer definitions in batch mode.

- Examples:
- [“Example 1: Defining Multiple Printer Definitions” on page 1556](#)
 - [“Example 2: Creating a Ghostview Printer in SASUSER to Preview PostScript Printer Output in SASUSER” on page 1557](#)
 - [“Example 3: Creating a Single Printer Definition That Is Available to All Users” on page 1559](#)
 - [“Example 4: Adding, Modifying, and Deleting Printer Definitions” on page 1561](#)
 - [“Example 5: Deleting a Single Printer Definition” on page 1563](#)

Syntax

PROC PRTDEF *<options>*;

Optional Arguments

DATA=SAS-*data-set*

specifies the SAS data set that contains the printer attributes.

Requirement Printer attributes variables that must be specified are DEST, DEVICE, MODEL, and NAME, except when the value of the variable OPCODE is DELETE. In that case, only the NAME variable is required.

See [“Input Data Set Variables That Are Used To Create Printer Definitions” on page 1550](#)

DELETE

specifies that the default operation is to delete the printer definitions from the registry.

Interaction If both DELETE and REPLACE are specified, then DELETE is the default operation.

Tip If the user-defined printer definition is deleted, then the administrator-defined printer can still appear if it exists in the Sashelp catalog.

Example [“Example 5: Deleting a Single Printer Definition” on page 1563](#)

FOREIGN

specifies that the registry entries are being created for export to a different host. As a consequence, tests of any host-dependent items, such as the TRANTAB, are skipped.

LIST

specifies that a list of printers that is created or replaced is written to the log.

Examples [“Example 2: Creating a Ghostview Printer in SASUSER to Preview PostScript Printer Output in SASUSER” on page 1557](#)

[“Example 4: Adding, Modifying, and Deleting Printer Definitions” on page 1561](#)

REPLACE

specifies that the default operation is to modify existing printer definitions. Any printer name that already exists is modified by using the information in the printer attributes data set. Any printer name that does not exist is added.

Interaction If both REPLACE and DELETE are specified, then a DELETE is performed.

Example [“Example 2: Creating a Ghostview Printer in SASUSER to Preview PostScript Printer Output in SASUSER” on page 1557](#)

USESASHELP

specifies that the printer definitions are to be placed in the Sashelp library, where they are available to all users. If the USESASHELP option is not specified, then the printer definitions are placed in the current Sasuser library, where they are available to the local user only.

Windows Specifics: You can create printer definitions with PROC PRTDEF in the Windows operating environment. However, because Universal Printing is turned off by default in Windows, these printer definitions do not appear in the Print window. If you want to use your printer definitions when Universal Printing is turned off, then either specify the printer definition as part of the PRINTERPATH system option or, from the Output Delivery System (ODS), issue the following code:

```
ODS PRINTER SAS PRINTER=myprinter;
```

where *myprinter* is the name of your printer definition.

Restriction To use the USESASHELP option, you must have permission to write to the Sashelp catalog.

Example [“Example 3: Creating a Single Printer Definition That Is Available to All Users” on page 1559](#)

Usage: PRTDEF Procedure

Input Data Set Variables That Are Used To Create Printer Definitions

Summary of Valid Variables

To create your printer definitions, you must create a SAS data set whose variables contain the appropriate printer attributes. The following table lists and describes both the required and optional variables for this data set.

Table 43.1 Required and Optional Variable for Creating Printer Definition Records

Variable Name	Variable Description
Required	
DEST	Destination
DEVICE	Device
MODEL	Prototype
NAME	Printer name
Optional	
BOTTOM	Default bottom margin

Variable Name	Variable Description
CHARSET	Default font character set
DESC	Description
FONTSIZE	Point size of the default font
HOSTOPT	Host options
LEFT	Default left margin
LRECL	Output buffer size
OPCODE	Operation code
PAPERIN	Paper source or input tray
PAPEROUT	Paper destination or output tray
PAPERSIZ	Paper size
PAPERTYP	Paper type
PREVIEW	Preview
PROTOCOL	Protocol
RES	Default printer resolution
RIGHT	Default right margin
STYLE	Default font style
TOP	Default top margin
TRANTAB	Translation table
TYPEFACE	Default font
UNITS	CM or IN units
VIEWER	Viewer
WEIGHT	Default font weight

Required Variables

To create or modify a printer, you must supply the NAME, MODEL, DEVICE, and DEST variables. All the other variables use default values from the printer prototype that is specified by the MODEL variable.

To delete a printer, specify only the required NAME variable.

The following variables are required in the input data set:

DEST

specifies the output destination for the printer.

Operating Environment Information: DEST is case sensitive for some devices.

Restriction DEST is limited to 1023 characters.

DEVICE

specifies the type of I/O device to use when sending output to the printer. Valid devices are listed in the Printer Definition wizard and in the SAS Registry Editor.

Restriction DEVICE is limited to 31 characters.

MODEL

specifies the printer prototype to use when defining the printer.

For a valid list of prototypes or model descriptions, you can look in the SAS Registry Editor under CORE\PRINTING\PROTOTYPES.

Restriction MODEL is limited to 127 characters.

Tip While in interactive mode, you can invoke the registry with the REGEDIT command.

NAME

specifies the printer definition name that is associated with the rest of the attributes in the printer definition.

The name is unique within a given registry. If a new printer definition contains a name that already exists, then the record is not processed unless the REPLACE option has been specified or unless the value of the OPCODE variable is **Modify**.

Restriction NAME is limited to 127 characters, must have at least one nonblank character, and cannot contain a backslash. Leading and trailing blanks are stripped from the name.

Optional Variables

The following variables are optional in the input data set:

BOTTOM

specifies the default bottom margin in the units that are specified by the UNITS variable.

CHARSET

specifies the default font character set.

Restriction The value must be one of the character set names in the typeface that is specified by the TYPEFACE variable.

DESC

specifies the description of the printer.

Default DESC defaults to the prototype that is used to create the printer.

Restriction The description can have a maximum of 1023 characters.

FONTSIZE

specifies the point size of the default font.

HOSTOPT

specifies any host options for the output destination. The host options are not case sensitive.

Restriction The host options can have a maximum of 1023 characters.

LEFT

specifies the default left margin in the units that are specified by the UNITS variable.

LRECL

specifies the buffer size or record length to use when sending output to the printer.

Default If LRECL is less than zero when modifying an existing printer, the printer's buffer size is reset to the size that is specified by the printer prototype.

OPCODE

is a character variable that specifies what action (Add, Delete, or Modify) to perform on the printer definition.

Add

creates a new printer definition in the registry. If the REPLACE option has been specified, then this operation will also modify an existing printer definition.

Delete

removes an existing printer definition from the registry.

Restriction This operation requires only the NAME variable to be defined. The other variables are ignored.

Modify

changes an existing printer definition in the registry or adds a new one.

Restriction OPTCODE is limited to eight characters.

Tip If a user modifies and saves new attributes on a printer in the SASHELP library, then these modifications are stored in the SASUSER library. Values that are specified by the user will override values that are set by the administrator, but they will not replace them.

PAPERIN

specifies the default paper source or input tray.

Restriction The value of PAPERIN must be one of the paper source names in the printer prototype that is specified by the MODEL variable.

PAPEROUT

specifies the default paper destination or output tray.

Restriction The value of PAPEROUT must be one of the paper destination names in the printer prototype that is specified by the MODEL variable.

PAPERSIZ

specifies the default paper source or input tray.

Restriction The value of PAPERSIZ must be one of the paper size names listed in the printer prototype that is specified by the MODEL variable.

PAPERTYP

specifies the default paper type.

Restriction The value of PAPERTYP must be one of the paper source names listed in the printer prototype that is specified by the MODEL variable.

PREVIEW

specifies the printer application to use for print preview.

Restriction PREVIEW is limited to 127 characters.

PROTOCOL

specifies the I/O protocol to use when sending output to the printer.

Operating Environment Information: On mainframe systems, the protocol describes how to convert the output to a format that can be processed by a protocol converter that connects the mainframe to an ASCII device.

Restriction PROTOCOL is limited to 31 characters.

RES

specifies the default printer resolution.

Restriction The value of RES must be one of the resolution values available to the printer prototype that is specified by the MODEL variable.

RIGHT

specifies the default right margin in the units that are specified by the UNITS variable.

STYLE

specifies the default font style.

Restriction The value of STYLE must be one of the styles available to the typeface that is specified by the TYPEFACE variable.

TOP

specifies the default top margin in the units that are specified by the UNITS variable.

TRANTAB

specifies which translation table to use when sending output to the printer.

Operating Environment Information: The translation table is needed when an EBCDIC host sends data to an ASCII device.

Restriction TRANTAB is limited to eight characters.

TYPEFACE

specifies the typeface of the default font.

Restriction The typeface must be one of the typeface names available to the printer prototype that is specified by the MODEL variable.

UNITS

specifies the units CM or IN that are used by margin variables.

VIEWER

specifies the host system command that is to be used during print previews. As a result, PROC PRTDEF causes a preview printer to be created.

Preview printers are specialized printers that are used to display printer output on the screen before printing.

Restriction VIEWER is limited to 127 characters.

Tip The values of the PREVIEW, PROTOCOL, DEST, and HOSTOPT variables are ignored when a value for VIEWER has been specified. Place %s where the input filename would normally be in the viewer command. %s can be used as many times as needed.

WEIGHT

specifies the default font weight.

Restriction The value must be one of the valid weights for the typeface that is specified by the TYPEFACE variable.

Examples: PRTDEF Procedure

Example 1: Defining Multiple Printer Definitions

Features: PROC PRTDEF statement options
DATA=

Details

This example shows you how to set up various printers.

Program

```
data printers;
input name $ 1-14 model $ 16-42 device $ 46-53 dest $ 57-70;
datalines;
Myprinter      PostScript Level 1 (Color)    PRINTER    printer1
Laserjet       PCL 5 (DeltaRow)              PIPE       lp -dprinter5
Color LaserJet PostScript Level 2 (Color)    PIPE       lp -dprinter2
;

proc prtdef data=printers;
run;
```

Program Description

Create the PRINTERS data set. The INPUT statement contains the names of the four required variables. Each data line contains the information that is needed to produce a single printer definition.

```
data printers;
input name $ 1-14 model $ 16-42 device $ 46-53 dest $ 57-70;
datalines;
Myprinter      PostScript Level 1 (Color)    PRINTER    printer1
Laserjet       PCL 5 (DeltaRow)              PIPE       lp -dprinter5
Color LaserJet PostScript Level 2 (Color)    PIPE       lp -dprinter2
;
```

Specify the input data set that contains the printer attributes and create the printer definitions. PROC PRTDEF creates the printer definitions for the SAS registry, and the DATA= option specifies PRINTERS as the input data set that contains the printer attributes.

```
proc prtdef data=printers;  
run;
```

Log

Example Code 43.1 The SAS Log After Defining Printers

```
1  data printers;  
2  input name $ 1-14 model $ 16-42 device $ 46-53 dest $ 57-70;  
3  datalines;  
  
NOTE: The data set WORK.PRINTERS has 3 observations and 4 variables.  
NOTE: DATA statement used (Total process time):  
      real time           0.03 seconds  
      cpu time            0.03 seconds  
  
7  ;  
8  proc prtdef data=printers;  
9  run;  
  
NOTE: 3 printer definitions added to the registry.  
NOTE: 0 printer definitions modified in the registry.  
NOTE: 0 printer definitions deleted from the registry.  
NOTE: PROCEDURE PRTDEF used (Total process time):  
      real time           0.15 seconds  
      cpu time            0.01 seconds
```

Example 2: Creating a Ghostview Printer in SASUSER to Preview PostScript Printer Output in SASUSER

Features: PROC PRTDEF statement options
DATA=
LIST
REPLACE

Details

This example creates a Ghostview printer definition in the Sasuser library for previewing PostScript output.

Program

```
data gsview;
  name = "Ghostview";
  desc = "Print Preview with Ghostview";
  model= "PostScript Level 2 (Color)";
  viewer = 'ghostview %s';
  device = "Dummy";
  dest = " ";
run;

proc prtdef data=gsview list replace;
run;
```

Program Description

Create the GSVIEW data set, and specify the printer name, printer description, printer prototype, and commands to be used for print preview. The GSVIEW data set contains the variables whose values contain the information that is needed to produce the printer definitions. The NAME variable specifies the printer name that will be associated with the rest of the attributes in the printer definition data record. The DESC variable specifies the description of the printer. The MODEL variable specifies the printer prototype to use when defining this printer. The VIEWER variable specifies the host system commands to be used for print preview. GSVIEW must be installed on your system and the value for VIEWER must include the path to find it. You must enclose the value in single quotation marks because of the %s. If you use double quotation marks, SAS will assume that %s is a macro variable. DEVICE and DEST are required variables, but no value is needed in this example. Therefore, a “dummy” or blank value should be assigned.

```
data gsview;
  name = "Ghostview";
  desc = "Print Preview with Ghostview";
  model= "PostScript Level 2 (Color)";
  viewer = 'ghostview %s';
  device = "Dummy";
  dest = " ";
run;
```

Specify the input data set that contains the printer attributes, create the printer definitions, write the printer definitions to the SAS log, and replace a printer definition in the SAS registry. The DATA= option specifies GSVIEW as the input

data set that contains the printer attributes. PROC PRTDEF creates the printer definitions. The LIST option specifies that a list of printers that are created or replaced will be written to the SAS log. The REPLACE option specifies that a printer definition will replace a printer definition in the registry if the name of the printer definition matches a name already in the registry. If the printer definition names do not match, then the new printer definition is added to the registry.

```
proc prtdef data=gsview list replace;
run;
```

Log

Example Code 43.2 The SAS Log After Defining a GhostView Printer

```
10  data gsview;
11  name = "Ghostview";
12  desc = "Print Preview with Ghostview";
13  model= "PostScript Level 2 (Color)";
14  viewer = 'ghostview %s';
15  device = "Dummy";
16  dest = " ";

NOTE: The data set WORK.GSVIEW has 1 observations and 6 variables.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds

17  proc prtdef data=gsview list replace;
18  run;

NOTE: Printer Ghostview created.
NOTE: 1 printer definitions added to the registry.
NOTE: 0 printer definitions modified in the registry.
NOTE: 0 printer definitions deleted from the registry.
NOTE: PROCEDURE PRTDEF used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

Example 3: Creating a Single Printer Definition That Is Available to All Users

Features: PROC PRTDEF statement options
DATA=
USESASHELP

Restriction: To use the USESASHELP option, you must have permission to write to the Sashelp catalog.

Details

This example creates a definition for a Tektronix Phaser 780 printer with a Ghostview print previewer with the following specifications:

- bottom margin set to 1 inch
- font size set to 14 point
- paper size set to A4

Program

```
data tek780;
  name = "Tek780";
  desc = "Test Lab Phaser 780P";
  model = "Tek Phaser 780 Plus";
  device = "PRINTER";
  dest = "testlab3";
  preview = "Ghostview";
  units = "cm";
  bottom = 2.5;
  fontsize = 14;
  papersiz = "ISO A4";
run;

proc prtdef data=tek780 usesashelp;
run;
```

Program Description

Create the TEK780 data set and supply appropriate information for the printer destination. The TEK780 data set contains the variables whose values contain the information that is needed to produce the printer definitions. In the example, assignment statements are used to assign these variables. The NAME variable specifies the printer name that is associated with the rest of the attributes in the printer definition data record. The DESC variable specifies the description of the printer. The MODEL variable specifies the printer prototype to use when defining this printer. The DEVICE variable specifies the type of I/O device to use when sending output to the printer. The DEST variable specifies the output destination for the printer. The PREVIEW variable specifies which printer is used for print preview. The UNITS variable specifies whether the margin variables are measured in centimeters or inches. The BOTTOM variable specifies the default bottom margin in the units that are specified by the UNITS variable. The FONTSIZE variable specifies the point size of the default font. The PAPERSIZ variable specifies the default paper size.

```
data tek780;
```

```

name = "Tek780";
desc = "Test Lab Phaser 780P";
model = "Tek Phaser 780 Plus";
device = "PRINTER";
dest = "testlab3";
preview = "Ghostview";
units = "cm";
bottom = 2.5;
fontsize = 14;
papersiz = "ISO A4";
run;

```

Create the TEK780 printer definition and make the definition available to all users. The DATA= option specifies TEK780 as the input data set. The USESASHELP option specifies that the printer definition will be available to all users.

```

proc prtdef data=tek780 usesashelp;
run;

```

Example 4: Adding, Modifying, and Deleting Printer Definitions

Features: PROC PRTDEF statement options
DATA=
LIST

Details

This example does the following:

- adds two printer definitions
- modifies a printer definition
- deletes two printer definitions

Program

```

data printers;
length name $ 80
       model $ 80
       device $ 8
       dest $ 80
       opcode $ 3
;
input opcode $& name $& model $& device $& dest $&;

```

```

datalines;
add  Color PostScript      PostScript Level 2 (Color)      DISK
sasprt.ps
mod  LaserJet 5            PCL 5 (DeltaRow)                DISK
sasprt.pcl
del  Gray Postscript       PostScript Level 1 (Gray Scale)  DISK
sasprt.ps
del  test                  PostScript Level 2 (Color)      DISK
sasprt.ps
add  ColorPS               PostScript Level 2 (Color)      DISK
sasprt.ps
;

proc prtdef data=printers replace list;
run;

```

Program Description

Create the PRINTERS data set and specify which actions to perform on the printer definitions. The PRINTERS data set contains the variables whose values contain the information that is needed to produce the printer definitions. The MODEL variable specifies the printer prototype to use when defining this printer. The DEVICE variable specifies the type of I/O device to use when sending output to the printer. The DEST variable specifies the output destination for the printer. The OPCODE variable specifies which action (add, delete, or modify) to perform on the printer definition. The first Add operation creates a new printer definition for Color PostScript in the SAS registry. The second Add operation creates a new printer definition for ColorPS in the SAS registry. The Mod operation modifies the existing printer definition for LaserJet 5 in the registry. The Del operation deletes the printer definitions for test from the registry. The & specifies that two or more blanks separate character values. This allows the name and model value to contain blanks.

```

data printers;
length name $ 80
       model $ 80
       device $ 8
       dest $ 80
       opcode $ 3
;
input opcode $& name $& model $& device $& dest $&;
datalines;
add  Color PostScript      PostScript Level 2 (Color)      DISK
sasprt.ps
mod  LaserJet 5            PCL 5 (DeltaRow)                DISK
sasprt.pcl
del  Gray Postscript       PostScript Level 1 (Gray Scale)  DISK
sasprt.ps
del  test                  PostScript Level 2 (Color)      DISK
sasprt.ps
add  ColorPS               PostScript Level 2 (Color)      DISK
sasprt.ps
;

```


Create multiple printer definitions and write them to the SAS log. The DATA= option specifies the input data set PRINTERS that contains the printer attributes. PROC PRTDEF creates five printer definitions, two of which have been deleted. The LIST option specifies that a list of printers that are created or replaced will be written to the log.

```
proc prtdef data=printers replace list;
run;
```

Log

Example Code 43.3 The SAS Log After Modifying and Deleting Printers

```
15  data printers;
16  length name  $ 80
17      model  $ 80
18      device $ 8
19      dest   $ 80
20      opcode $ 3
21      ;
22  input opcode $& name $& model $& device $& dest $&;
23  datalines;

NOTE: The data set WORK.PRINTERS has 5 observations and 5 variables.
NOTE: DATA statement used (Total process time):
      real time          0.01 seconds
      cpu time           0.01 seconds

29  ;
30  proc prtdef data=printers list replace;
31  run;

NOTE: Printer Color PostScript modified.
NOTE: Printer LaserJet 5 modified.
NOTE: Printer Gray Postscript deleted.
NOTE: Printer test deleted.
NOTE: Printer ColorPS modified.
NOTE: PROCEDURE PRTDEF used (Total process time):
      real time          0.04 seconds
      cpu time           0.03 seconds
```

Example 5: Deleting a Single Printer Definition

Features: PROC PRTDEF statement option
DELETE

Details

This example shows you how to delete a printer from the registry.

Program

```
data deleteprt;  
  name='printer1';  
run;  
  
proc prtdef data=deleteprt delete list;  
run;
```

Program Description

Create the DELETEPRT data set. The NAME variable contains the name of the printer to delete.

```
data deleteprt;  
  name='printer1';  
run;
```

Delete the printer definition from the registry and write the deleted printer to the log. The DATA= option specifies DELETEPRT as the input data set. PROC PRTDEF creates printer definitions for the SAS registry. DELETE specifies that the printer is to be deleted. LIST specifies to write the deleted printer to the log.

```
proc prtdef data=deleteprt delete list;  
run;
```

Log

Example Code 43.4 The SAS Log After Deleting a Single Printer

```
45  data deleteprt;
46      name='printer1';
47      run;

NOTE: The data set WORK.DELETEPRT has 1 observations and 1 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.00 seconds

48      proc prtdef data=deleteprt delete list;
49      run;

NOTE: Printer printer1 deleted.
NOTE: 0 printer definitions added to the registry.
NOTE: 0 printer definitions modified in the registry.
NOTE: 1 printer definitions deleted from the registry.
NOTE: PROCEDURE PRTDEF used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```


PRTEXP Procedure

Overview: PRTEXP Procedure	1567
What Does the PRTEXP Procedure Do?	1567
Syntax: PRTEXP Procedure	1568
PROC PRTEXP Statement	1568
EXCLUDE Statement	1569
SELECT Statement	1569
Examples: PRTEXP Procedure	1570
Example 1: Writing Attributes to the SAS Log	1570
Example 2: Writing Attributes to a SAS Data Set	1571

Overview: PRTEXP Procedure

What Does the PRTEXP Procedure Do?

The PRTEXP procedure enables you to replicate, modify, and create printer definitions from the SAS registry, either for an individual user or for all SAS users at your site. PROC PRTEXP then writes these attributes to the SAS log or to a SAS data set. You can specify that PROC PRTEXP search for these attributes in the SASHELP portion of the registry or the entire SAS registry.

If you write printer definitions to a SAS data set, you can later replicate and modify them. You can then use PROC PRTDEF to create the printer definitions in the SAS registry from your input data set. For a complete discussion of PROC PRTDEF and the variables and attributes that are used to create the printer definitions, see [“Input Data Set Variables That Are Used To Create Printer Definitions”](#) on page 1550.

See Also

[Chapter 43, “PRTDEF Procedure,” on page 1547](#)

Syntax: PRTEXP Procedure

Tip: If neither the SELECT nor the EXCLUDE statement is used, then all of the printers will be included in the output.

```
PROC PRTEXP <options>;  
  SELECT printer(s);  
  EXCLUDE printer(s)
```

Statement	Task	Example
PROC PRTEXP	Obtain printer attributes from the SAS registry	Ex. 1 , Ex. 2
EXCLUDE	Obtain printer attributes for all printers except for the specified printers	
SELECT	Obtain printer attributes for the specified printers	Ex. 1 , Ex. 2

PROC PRTEXP Statement

Replicates, modifies, and creates printer definitions.

Examples: [“Example 1: Writing Attributes to the SAS Log” on page 1570](#)
[“Example 2: Writing Attributes to a SAS Data Set” on page 1571](#)

Syntax

```
PROC PRTEXP <options>;
```

Optional Arguments

USESASHELP
specifies that SAS search only the SASHELP portion of the registry for printer definitions.

Default The default is to search both the SASUSER and SASHELP portions of the registry for printer definitions.

OUT=SAS-*data-set*

specifies the SAS data set that contains the printer definitions.

The data set that is specified by the OUT=SAS-*data-set* option is the same type of data set that is specified by the DATA=SAS-*data-set* option in PROC PRTDEF to define each printer.

Default If OUT=SAS-*data-set* is not specified, then the data that is needed to define each printer is written to the SAS log.

EXCLUDE Statement

Names the printers whose information does not appear in output.

Syntax

EXCLUDE *printer(s)*;

Required Argument

printer(s)

specifies one or more printers that you do not want the output to contain information about.

SELECT Statement

Names the printers whose information is contained in the output.

Examples: [“Example 1: Writing Attributes to the SAS Log” on page 1570](#)
 [“Example 2: Writing Attributes to a SAS Data Set” on page 1571](#)

Syntax

SELECT *printer(s)*;

Required Argument

printer(s)

specifies one or more printers that you would like the output to contain information about.

Examples: PRTEXP Procedure

Example 1: Writing Attributes to the SAS Log

Features: PROC PRTEXP statement option
USESASHELP
SELECT statement

Details

This example shows you how to write the attributes that are used to define a printer to the SAS log.

Program

Specify the printer that you want information about, specify that only the SASHELP portion of the registry be searched, and write the information to the SAS log. The SELECT statement specifies that you want the attribute information that is used to define the printer PostScript to be included in the output. The USESASHELP option specifies that only the SASHELP registry is to be searched for PostScript's printer definitions. The data that is needed to define each printer is written to the SAS log because the OUT= option was not used to specify a SAS data set.

```
proc prtexp usesashelp;  
select postscript;  
run;
```

Log

Example Code 44.1 *The SAS Log After Extracting Printer Information from the SASHELP Portion of the Registry*

```
379 proc prtexp usesashelp;
380 select postscript;
381 run;

NAME:      PostScript
MODEL:     PostScript Level 1 (Color)
DEVICE:    DISK
DEST:      sasprt.ps
HOSTOPT:
PROTOCOL:
TRANTAB:
DESC:      Generic PostScript Level 1 Printer
PREVIEW:   Adobe Reader
VIEWER:
PAPERSIZ:
PAPERTYP:
PAPERIN:
PAPEROUT:
RES:       300 DPI
TOP:       0.50
LEFT:      0.50
RIGHT:     0.50
BOTTOM:    0.50
UNITS:     IN
TYPEFACE:  <MTmonospace>
WEIGHT:    Normal
STYLE:     Regular
CHARSET:   Western
FONTSIZE:  8.00
LRECL:     .
```

Example 2: Writing Attributes to a SAS Data Set

Features: PROC PRTEXP statement option
OUT=
SELECT statement

Details

This example shows you how to create a SAS data set that contains the data that PROC PRTDEF would use to define the printers PCL4, PCL5, PCL5E, and PCLC.

Program

Specify the printers that you want information about and create the PRDVTER data set. The SELECT statement specifies the printers PCL4, PCL5, PCL5E, and PCLC. The OUT= option creates the SAS data set PRDVTER, which contains the same attributes that are used by PROC PRTDEF to define the printers PCL4, PCL5, PCL5E, and PCLC. SAS will search both the SASUSER and SASHELP registries, because USESASHELP was not specified.

```
proc prtexp out=PRDVTER;
  select pcl4 pcl5 pcl5e pcl5c;
run;

proc print data=prdvter;
run;
```

Output

The following data set is a partial view of the Prdvter data set that contains 26 variables and four observations.

Output 44.1 The Output Data Set for Prdvter

The SAS System									
Obs	DEST	HOSTOPT	DESC	VIEWER	NAME	MODEL	PREVIEW	TYPEFACE	
1	sasprt.pcl		Generic PCL 4 Printer		PCL4	PCL 4	Adobe Reader		
2	sasprt.pcl		Generic PCL 5 Printer		PCL5	PCL 5 (DeltaRow)	Adobe Reader		
3	sasprt.pcl		Generic PCL 5 RGB Color Printer with Alpha Blending		PCL5c	PCL 5rgba (DeltaRow)	Adobe Reader		
4	sasprt.pcl		Generic PCL 5e Printer		PCL5e	PCL 5e (DeltaRow)	Adobe Reader		

PWENCODE Procedure

Overview: PWENCODE Procedure	1573
What Does the PWENCODE Procedure Do?	1573
Concepts: PWENCODE Procedure	1574
Using Encoded Passwords in SAS Programs	1574
Encoding versus Encryption	1574
Encoding Methods	1575
Syntax: PWENCODE Procedure	1576
PROC PWENCODE Statement	1576
Examples: PWENCODE Procedure	1578
Example 1: Encoding a Password	1578
Example 2: Saving an Encoded Password to the Paste Buffer	1579
Example 3: Specifying Method= SAS003 to Encode a Password	1580
Example 4: Specifying Method= SAS005 to Encode a Password	1581

Overview: PWENCODE Procedure

What Does the PWENCODE Procedure Do?

The PWENCODE procedure enables you to encode passwords. Encoding obfuscates the data. Unlike encryption, encoding is a reversible permutation of the data and uses no keys.

Encoded passwords can be used in place of plaintext passwords in SAS programs that access SAS 9 legacy metadata servers and workspace servers, the Compute Server and services in the SAS Viya platform, CAS servers and services, and relational database management systems (RDBMSs).

Concepts: PWENCODE Procedure

Using Encoded Passwords in SAS Programs

When a password is encoded with PROC PWENCODE, the output string includes a tag that identifies the string as having been encoded. An example of a tag is `{sas001}`. The tag indicates the encoding method. SAS servers and SAS/ACCESS engines recognize the tag and decode the string before using it. Encoding a password enables you to write SAS programs without having to specify a password in plaintext.

Note: PROC PWENCODE passwords can contain up to a maximum of 512 characters, which include alphanumeric characters, spaces, and special characters. Data set passwords, however, must follow SAS naming rules. For information, see [“SAS Names” in SAS Programmer’s Guide: Essentials](#).

The encoded password is never written to the SAS log in plain text. Instead, each character of the password is replaced by an X in the SAS log.

Encoding versus Encryption

Encoding techniques disguise passwords and the approach is intended to prevent casual, non-malicious viewing of passwords. With encoding, one character set is translated to another character set through some form of table lookup. SAS001–SAS005 encoding methods are considered encoding methods.

Encryption, by contrast, involves the transformation of data from one form to another through the use of mathematical operations and, usually, a “key” value. Encryption is generally more difficult to break than encoding.

SAS003, SAS004, and SAS005 methods specified in PROC PWENCODE designate encryption techniques that align with industry standards. These options support longer encryption keys (for example, 256-bit). Salting and multiple iterations are provided to the AES encryption algorithm to create passwords that are harder to break.

Encoding methods for PROC PWENCODE are shown in [“Encoding Methods” on page 1575](#).

Password protection is an important part of your security strategy, but you should not rely only on password protection for all your data security needs; a determined

and knowledgeable attacker can break passwords. Data should also be protected by other security controls such as file system permissions, other access control mechanisms, and encryption of data at rest and in motion.

Encoding Methods

The following encoding methods are supported.

Table 45.1 *Supported Encoding Methods*

Encoding Method	Data Encryption Algorithm Used	Encoded Password/key Description
<code>sas001</code>	None	Uses base64 to encode passwords.
<code>sas002</code> , which can also be specified as <code>sasenc</code>	SASProprietary is included in SAS software.	Uses a 32-bit fixed key.
<code>sas003</code>	AES (Advanced Encryption Standard)	Uses a 256-bit fixed key plus a 16-bit random salt value.
<code>sas004</code>	AES (Advanced Encryption Standard)	Uses a 256-bit fixed key and a 64-bit random salt value.
<code>sas005</code>	AES (Advanced Encryption Standard)	Uses a 256-bit fixed key, a 64-bit random salt value, and is hashed for additional iterations.

The SAS003, SAS004, and SAS005 encoded passwords use AES encryption and a 256-bit fixed key. In addition, a random salt value is applied to the encoding method. Therefore, each time that you use PROC PWENCODE to encode the same password, you get a different encoded password, because the salt values are random. In addition to the random salt value, SAS005 is hashed for additional iterations. SAS005 is the highest level of encoding that is used with PROC PWENCODE.

SASProprietary for SAS data set encryption with passwords is a cipher that uses parts of the passwords that are stored in the SAS data set as part of the 32-bit rolling key encoding of the data. This encryption provides a medium level of security. With the speed of today's computers, it could be subjected to a brute force attack on the 2,563,160,682,591 possible combinations of valid password values, many of which must produce the same 32-bit key.

SASProprietary data set password encryption is primarily used with customer data from legacy SAS systems. For example, SASProprietary might be used with SAS 9

data in the SAS Metadata Server, the workspace server in SAS Viya 3.5 and earlier releases, or the SAS Compute Server in the SAS Viya platform.

Syntax: PWENCODE Procedure

PROC PWENCODE IN=*'password'* < OUT=*fileref* > < METHOD=*encoding-method* >;

Statement	Task	Example
PROC PWENCODE	Encode a password	Ex. 1, Ex. 2, Ex. 3, Ex. 4

PROC PWENCODE Statement

Encodes a password.

- Examples:
- “Example 1: Encoding a Password” on page 1578
 - “Example 2: Saving an Encoded Password to the Paste Buffer” on page 1579
 - “Example 3: Specifying Method= SAS003 to Encode a Password” on page 1580
 - “Example 4: Specifying Method= SAS005 to Encode a Password” on page 1581

Syntax

PROC PWENCODE IN=*'password'* < OUT=*fileref* > < METHOD=*encoding-method* >;

Required Argument

IN=*'password'*
specifies the password to encode. The password can contain up to a maximum of 512 characters, which include alphanumeric characters, spaces, and special characters.

Note: Data set passwords must follow SAS naming rules. If the IN=*password* follows SAS naming rules, it can also be used for SAS data sets. For information, see “SAS Names” in *SAS Programmer’s Guide: Essentials*.

If the password contains embedded single or double quotation marks, use the standard SAS rules for quoting character constants. These rules can be found in “Constants” in *SAS Programmer’s Guide: Essentials*.

Note: Each character of the encoded password is replaced by an X when written to the SAS log.

See [“Example 1: Encoding a Password” on page 1578](#)

[“Example 2: Saving an Encoded Password to the Paste Buffer” on page 1579](#)

[“Example 3: Specifying Method= SAS003 to Encode a Password” on page 1580](#)

[“Example 4: Specifying Method= SAS005 to Encode a Password” on page 1581](#)

Optional Arguments

OUT=*fileref*

specifies a fileref to which the output string is to be written. If the OUT= option is not specified, the output string is written to the SAS log.

Note: The global macro variable

`_PWENCODE`

is set to the value that is written to the OUT= fileref or to the value that is displayed in the SAS log.

See [“Example 2: Saving an Encoded Password to the Paste Buffer” on page 1579](#)

METHOD=*encoding-method*

specifies the encoding method. Here are the supported values for *encoding-method*.

SAS001

uses base64 to encode passwords.

SAS002

uses a 32-bit fixed key. Uses SAS Proprietary encryption.

Alias SASENC

Default SAS002

SAS003

uses a 256-bit fixed key plus a 16-bit random salt value. Uses AES encryption.

SAS004

uses a 256-bit fixed key and a 64-bit random salt value. Uses AES encryption.

SAS005

uses a 256-bit fixed key, a 64-bit random salt value, and is hashed for additional iterations. Uses AES encryption.

If the METHOD= option is omitted, the default encoding method is used. The default method is **sas002** in most cases. **SAS002** is also the default method used if you specify an invalid method.

See [“Example 1: Encoding a Password” on page 1578](#)

[“Example 2: Saving an Encoded Password to the Paste Buffer” on page 1579](#)

[“Example 3: Specifying Method= SAS003 to Encode a Password” on page 1580](#)

[“Example 4: Specifying Method= SAS005 to Encode a Password” on page 1581](#)

Examples: PWENCODE Procedure

Example 1: Encoding a Password

Features: **IN=** argument

Details

This example shows a simple case of encoding a password and writing the encoded password to the SAS log.

Program

Encode the password.

```
proc pwencode in='my password';  
run;
```

Log

Note that each character of the password is replaced by an X in the SAS log.

```
19  proc pwencode in=XXXXXXXXXXXXX;  
20  run;  
  
{SAS002}DBCC571245AD0B31433834F80BD2B99E16B3C969  
  
NOTE: PROCEDURE PWENCODE used (Total process time):  
      real time          0.01 seconds  
      cpu time           0.01 seconds
```

Example 2: Saving an Encoded Password to the Paste Buffer

Features: IN= argument
 OUT= option
 FILENAME statement with CLIPBRD access method

DETAILS

This example saves an encoded password to the paste buffer. You can then paste the encoded password into another SAS program or into the password field of an authentication dialog box.

Program

```
filename clip clipbrd;  
  
proc pwencode in='my password' out=clip;  
run;
```

Program Description

Declare a fileref with the CLIPBRD access method.

```
filename clip clipbrd;
```

Encode the password and save it to the paste buffer. The OUT= option saves the encoded password to the fileref that was declared in the previous statement.

```
proc pwencode in='my password' out=clip;
run;
```

Log

Note that each character of the password is replaced by an X in the SAS log.

```
24
25  filename clip clipbrd;
26  proc pwencode in=XXXXXXXXXXXXX out=clip;
27  run;

NOTE: PROCEDURE PWENCODE used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

Example 3: Specifying Method= SAS003 to Encode a Password

Features: IN= argument
 METHOD= argument

Details

This example shows a simple case of encoding a password using the **sas003** encoding method and writing the encoded password to the SAS log. SAS003 uses a 16-bit salt to encode a password.

Program

Encode the password using SAS003. The encoded password is a 256-bit key with a 16 bit random salt.

```
proc pwencode in='mypassword' method=sas003;
run;
```

Log

Note that each character of the password is replaced by an X in the SAS log. SAS003 encoding uses AES encryption plus a 16-bit salt. Because SAS003 uses random salting, each time you run the following code, a different password is generated.

```
8  proc pwencode in=XXXXXXXXXXXX method=sas003;
29 run;

{SAS003}4837B146585CED2C9FED14A3C946D68E4389

NOTE: PROCEDURE PWENCODE used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

Example 4: Specifying Method= SAS005 to Encode a Password

Features: IN= argument
 METHOD= argument

Details

This example shows a simple case of encoding a password using the **sas005** encoding method and writing the encoded password to the SAS log. SAS005 uses a 256-bit fixed key that uses a 64-bit random salt to encode the password.

Program

Encode the password using SAS005.

```
proc pwencode in='mypassword' method=sas005;
run;
```

Log

Note that each character of the password is replaced by an X in the SAS log. SAS005 encoding uses AES encryption with a 256-bit fixed key and a 64-bit random salt value. SAS005 increases security for stored passwords by using the

SHA-256 hashing algorithm and is hashed for additional iterations. Because SAS005 uses random salting, each time you run the following code, a different password is generated.

```
230 proc pwencode in=XXXXXXXXXX method=sas005;
231 run;

{SAS005}ADD8AB7108595A7D1A69190D78CDFE6145C1EB849CC7A43D

NOTE: PROCEDURE PWENCODE used (Total process time):
      real time          0.01 seconds
      cpu time           0.01 seconds
```

PYTHON Procedure

Overview: PYTHON Procedure	1583
What Does the PYTHON Procedure Do?	1584
Concepts: PYTHON Procedure	1584
How PROC PYTHON Works	1584
Configuration for PROC PYTHON	1585
Comparison of SUBMIT and INTERACTIVE Python Code	1586
Establishing a Connection to SAS Cloud Analytic Services	1586
Syntax: PYTHON Procedure	1588
PROC PYTHON Statement	1589
SUBMIT Statement	1590
ENDSUBMIT Statement	1590
INTERACTIVE Statement	1591
ENDINTERACTIVE Statement	1591
Usage: PYTHON Procedure	1592
Submitting Python Statements within a SUBMIT Block	1592
Submitting Compound Python Statements in Interactive Blocks	1593
Using PROC PYTHON Callback Methods	1594
Examples: PYTHON Procedure	1599
Example 1: Specify Input from an External Python Script	1599
Example 2: Observe the Impact of Restarting and Terminating the PYTHON Procedure	1602
Example 3: Submit SAS Code and Call a SAS Function within PROC PYTHON ..	1607
Example 4: Create a SAS Macro Variable in PROC PYTHON	1609
Example 5: Invoke an FCMP Function in PROC PYTHON	1611
Example 6: Render a Python Plot	1613

Overview: PYTHON Procedure

What Does the PYTHON Procedure Do?

Python is an interpreted, high-level, general-purpose programming language. The PYTHON procedure enables you to run statements from the Python programming language within SAS code. You can submit Python statements from an external Python script or enter Python statements directly.

PROC PYTHON enables you to perform these tasks:

- run Python code within a SAS session
- call most SAS functions within Python statements
- submit SAS code from Python
- move data between SAS data sets or views and Pandas DataFrames
- transfer values between SAS macro variables and Python variables
- call PROC FCMP functions within Python statements
- display Pyplot graphics in SAS Data and AI Studio results

Note: You must install the matplotlib.pyplot module to display graphics.

You can also run Python code in batch scripts. For more information, see [“Introduction” in Using the batch Plug-In for the SAS Viya Platform Command-Line Interface](#).

Concepts: PYTHON Procedure

How PROC PYTHON Works

PROC PYTHON creates a subprocess from a SAS Compute Server process. This subprocess uses the Python deployment that you choose. PROC PYTHON enables you to submit Python code either as one entire program in a SUBMIT block or one

line at a time within an INTERACTIVE block. For more information, see the [SUBMIT statement on page 1590](#) or the [INTERACTIVE statement on page 1591](#).

When you submit a block of Python code within the SUBMIT and ENDSUBMIT statements, you observe the following behaviors:

- If a statement results in an error or an exception, processing for the block of code stops at the point of the error.
- Compound statements are Python statements that contain other statements, such as `for` loops or `while` statements. Continuation statements within a compound statement are all indented the same amount (usually 4 spaces). The end of a compound statement occurs when a new statement begins at the same level as the top statement of the compound statement. No blank line is required between the end of a compound statement and the next Python statement.

When you run the PROC PYTHON subprocess in interactive mode, Python statements are processed by using interactive syntax. This results in the following behaviors:

- If a statement results in an error or an exception, the Python interpreter continues to process subsequent statements instead of ending immediately.
- Compound statements in an interactive block of Python code require a blank line to exit continuation lines. Therefore, you must follow compound statements with a blank line in an interactive block of Python code.

The PYTHON procedure enables you to use the same subprocess across procedure calls, so that objects or data are available in subsequent PYTHON procedure calls. You can choose to terminate a Python subprocess and create a new subprocess when desired, either at the beginning or end of a PYTHON procedure call. For more information, see [“Example 2: Observe the Impact of Restarting and Terminating the PYTHON Procedure” on page 1602](#).

When a PYTHON subprocess is created, a module that is part of the PYTHON procedure is automatically imported. This module is called SAS, and it enables interaction between SAS and your Python instance. The SAS module provides a number of callback methods that you can use to interact between the SAS Compute Server and your Python instance. These callback methods enable you to share variables between Python and SAS (`SAS.symget` or `SAS.symput`), move data between SAS data sets and Pandas DataFrames (`SAS.sd2df` or `SAS.df2sd`), invoke SAS and FCMP functions (`SAS.sasfnc`), and submit SAS code within your Python statements (`SAS.submit`). With these callback methods, you can create a workflow that mixes SAS and Python programming as you require. For more information, see [“Using PROC PYTHON Callback Methods” on page 1594](#).

Configuration for PROC PYTHON

There are some configuration requirements that enable PROC PYTHON to interact with Python on your system. You or a SAS administrator must configure two environment variables:

- `PROC_PYPATH` points to your Python executable. Your Python deployment, which includes the Python executable, is typically mounted to a file system that

is accessible by the SAS Compute Server. The Python executable creates the PYTHON subprocess that is triggered by the PYTHON procedure.

- PROC_M2PATH specifies the location of the sas2py.py file that is part of your deployment. This environment variable initially points to the default location of this file. The sas2py.py file contains the SAS module that is imported by the Python deployment. This file provides the interface for the callback methods that interact with the Compute Server session.

For more information, see [“Configure SAS to Run the Python Language” in SAS Viya Platform: Programming Run-Time Servers](#).

Your SAS Viya offering is deployed in a locked-down state by default. As a result, PROC PYTHON is disabled. Use the ENABLE_AMS= option for PYTHON and SOCKET access methods in the LOCKDOWN statement to enable PROC PYTHON. For more information, see these topics:

- LOCKDOWN System Option and LOCKDOWN Statement in [“Security Features” in SAS Viya Platform: Programming Run-Time Servers](#)
- [“LOCKDOWN” in SAS Programmer’s Guide: Essentials](#)

Comparison of SUBMIT and INTERACTIVE Python Code

When you submit Python code using the SUBMIT and ENDSUBMIT statements, your Python statements are submitted as a block of code when the end of the procedure call is reached. The output appears in the SAS log after all of the Python statements have been processed. If you include any SAS callback methods, then the output from those calls appears in the SAS log prior to the output from the Python code.

When you submit Python code interactively by using the INTERACTIVE and ENDINTERACTIVE statements, each Python statement is executed as you enter it. Output from Python processing is interspersed with output from any callback method that uses the SAS module, such as SAS.symget. Running Python code interactively displays Python and SAS output chronologically in the log. This is similar to running Python code from a command line and can be useful for debugging Python code.

Establishing a Connection to SAS Cloud Analytic Services

To access SAS Cloud Analytic Services (CAS) from PROC PYTHON, you need to use the Python SWAT package. This requires the use of these two environment variables:

- SAS_CLIENT_TOKEN - this provides token-based authentication to SAS Viya
- SSLCALISTLOC - this provides an SSL certificate for secure communication between the Python run time and SAS Viya.

Note: For this step to work, make sure that PROC PYTHON is enabled to interact with Python on your system. See [“Configuration for PROC PYTHON” on page 1585](#) for complete information.

The following code example shows how to use these variables on the `swat.CAS` class to pass the host name, port, and authentication token to create the connection.

```
cas casauto;
proc python;
submit;
import swat, os
os.environ['CAS_CLIENT_SSL_CA_LIST']=os.environ['SSLCALISTLOC']
sascas = swat.CAS(hostname="sas-cas-server-default-client",port=5570, password=os.environ['SAS_CLIENT_TOKEN'])
r = sascas.caslibinfo()
SAS.show(r['CASLibInfo'])
endsubmit;
run;
cas casauto terminate
```

If you are using a non-default CAS server or CAS host name, make sure to use that host name rather than the default `sas-cas-server-default-client`. See [Confirm the Identities of the Hosts](#) for more information.

This example uses the `caslibinfo()` method to access the caslibs that you have access to. The `show()` method is used to provide a list of the libraries in CAS that you can access in the Output tab, similar to the table below.

Output

Obs	Name	Type	Description	Path	Subdirs
1	AioTPgMeta	PATH	Stores quality analytic suite postgres tables.	/cas/data/caslibs/aioTPgMeta/	0
2	alm202505	PATH	ALM Data	/cas/data/caslibs/alm202505/	1
3	CASTestTmp	PATH	DDT created caslib	/bigdisk/lax/castest/	1
4	CASUSER(user1)	PATH	Personal File System Caslib	/cas/data/caslibs/casuserlibraries/user1/	1
5	costAndProfitability	PATH	Stores output tables for different cost and profitability management models.	/cas/data/caslibs/costAndProfitability/	0

See Also

[SAS Scripting Wrapper for Analytics Transfer \(SWAT\)](#)

Syntax: PYTHON Procedure

Restriction: When SAS is in a locked-down state, the PYTHON procedure is not available. Your server administrator can re-enable this procedure so that it is accessible in the locked-down state. For more information, see "Enabling PROC PYTHON While in Lockdown Mode" in ["Examples" in SAS Viya Platform: Programming Run-Time Servers](#).

```
PROC PYTHON < INFILE='file-name' | fileref > < RESTART > < TERMINATE > <
TIMEOUT=n >;
```

```
< SUBMIT; >
```

```
    Python statements
```

```
< ENDSUBMIT; >
```

```
RUN;
```

```
PROC PYTHON < INFILE='file-name' | fileref > < ECHO > < RESTART > <
TERMINATE > < TIMEOUT=n >;
```

```
< INTERACTIVE; >
```

```
    Python statements, submitted interactively
```

```
< ENDINTERACTIVE; >
```

```
RUN;
```

Statement	Task	Example
PROC PYTHON	Executes Python statements within SAS code or specifies a file that contains Python statements to execute.	Ex. 1, Ex. 2, Ex. 3, Ex. 4, Ex. 5
SUBMIT	Identifies the beginning of a block of Python statements.	Ex. 1, Ex. 2, Ex. 3, Ex. 4, Ex. 5
ENDSUBMIT	Identifies the end of a block of Python statements.	Ex. 1, Ex. 2, Ex. 3, Ex. 4, Ex. 5
INTERACTIVE	Identifies the beginning of interactive Python statements.	Ex. 6
ENDINTERACTIVE	Identifies the end of interactive Python statements.	Ex. 6

PROC PYTHON Statement

Runs Python statements within a SAS session.

Syntax

Form 1: SUBMIT block

```
PROC PYTHON < INFILE='file-name' | fileref > < RESTART > < TERMINATE > < TIMEOUT=n >;
```

Form 2: Interactive Python block

```
PROC PYTHON < INFILE='file-name' | fileref > < ECHO > < RESTART > < TERMINATE > < TIMEOUT=n >;
```

Optional Arguments

ECHO

specifies that interactively submitted lines of Python code should be included in the SAS log. The lines of code appear prior to the resulting output.

Restriction This option applies only when you submit Python code interactively.

INFILE='file-name' | fileref

identifies a source file or fileref that contains Python statements to run within a SAS session.

Requirements If you specify a file name, enclose it in single or double quotation marks.

If you use the INFILE= option, then you must specify the path to the Python script.

Examples

Use the Python code in open_data.py:

```
proc python infile='/user/myscripts/open_data.py';
```

Define a fileref for a file that contains Python code:

```
filename script 'my_script.py';
```

```
proc python infile=script;
```

RESTART

terminates a PYTHON subprocess if one is running currently, and creates a new subprocess. The PYTHON subprocess is then used for the current and subsequent calls to PROC PYTHON.

You specify RESTART at the beginning of a new block of Python code.

Example `proc python restart;`

```

submit;
<Python statements>
endsubmit;
run;

```

TERMINATE

stops the PYTHON subprocess when the current PYTHON procedure call completes. The next call to the PYTHON procedure begins a new instance of the PYTHON subprocess.

Example

```

proc python terminate;
interactive;
<Python statements submitted individually>
endinteractive;
run;

```

TIMEOUT=*n*

specifies the number of seconds that can elapse before PROC PYTHON terminates when attempting to connect to a Python environment.

Default 60

Note The default for TIMEOUT= became 60 in 2023.05.

Example

```

proc python timeout=30 <additional options>;
submit;
<Python statements>
endsubmit;
run;

```

SUBMIT Statement

Identifies the beginning of a block of Python code. Enter Python statements between the SUBMIT and ENDSUBMIT statements.

Requirement: Each SUBMIT statement must have a corresponding ENDSUBMIT statement.

Syntax

SUBMIT;

ENDSUBMIT Statement

Specified on a line by itself, this statement identifies the end of a block of Python statements.

Syntax

ENDSUBMIT;

INTERACTIVE Statement

Identifies the beginning of Python code that you submit interactively.

Alias: **I**

Requirement: Each INTERACTIVE statement must have a corresponding ENDINTERACTIVE statement.

Syntax

INTERACTIVE;

ENDINTERACTIVE Statement

Specified on a line by itself, this statement identifies the end of Python code that you submit interactively.

Syntax

ENDINTERACTIVE;

Usage: PYTHON Procedure

Submitting Python Statements within a SUBMIT Block

You can submit Python statements within the PROC PYTHON invocation, between SUBMIT and ENDSUBMIT statements. The following code executes a single Python `print` statement.

```
proc python;
submit;
print("Hello from Python")
endsubmit;
run;
```

The following code assigns strings to macro variables. In this example, the macro variable `Mymacrovar` is defined using the `SAS.submit` callback method. The macro variable `Myvar` was defined using the `SAS.symput` callback method. Both macro variables can be accessed outside of the call to PROC PYTHON.

```
proc python restart;
submit;
SAS.submit("%let mymacrovar=Hi there;")
SAS.symput('myvar',"This variable has global scope")
txt = SAS.symget("mymacrovar")
print(txt)
endsubmit;
run;

%put &=mymacrovar;
%put &=myvar;
```

Output 46.1 *Output with Macro Variables in PROC PYTHON*

```

89  %let mymacrovar=Hi there;
>>>
Hi there
>>>
>>>
NOTE: PROCEDURE PYTHON used (Total process time):
      real time          1.17 seconds
      cpu time           0.07 seconds

90
91  %put &=mymacrovar;
MYMACROVAR=Hi there
92  %put &=myvar;
MYVAR=This variable has global scope

```

Submitting Compound Python Statements in Interactive Blocks

A compound Python statement is one where one or more statements are controlled by an initial statement. Some examples are `if`, `while`, or `for` statements. When you run PROC PYTHON in interactive mode, you must add a blank line between the end of a top-level compound statement and a statement that follows it. In addition, do not embed blank lines within the continuation lines of a top-level compound statement, because that tells the Python parser that you have ended the compound statement. You can include lines that contain only the number of spaces that match the current indentation level. These lines look like blank lines, but the Python parser treats them as continuation lines within the compound statement.

Here is an example that shows how to submit a `for` statement in interactive mode. The `for` statement loops over values of `x`, which is assigned to successive values in the array called `fruits`.

```

proc python;
interactive;
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    print(x)

print('first statement after for loop')
endinteractive;
run;

```

Here is the output from the `for` loop and the statement that follows it.

Output 46.2 For Loop Output

```

>>>
>>>
... .. apple
banana
cherry
>>>
first statement after for loop
>>>
>>>

```

If you do not have a blank line after your compound statement you receive an invalid syntax error in the log.

Output 46.3 Error When Missing a Blank Line after a Compound Statement

```

>>> fruits = ["apple", "banana", "cherry"]
>>> for x in fruits:
...     print
...     print('just after for-loop')
      File "<stdin>", line 3
        print('just after for-loop')
          ^
SyntaxError: invalid syntax
>>>

```

Using PROC PYTHON Callback Methods

Callback Methods That Interact with SAS

PROC PYTHON provides a module called SAS that enables you to communicate between your SAS session and the PYTHON subprocess. You can submit SAS code or interact with SAS variables within a Python code block by using the following callback methods. Because Python is a case-sensitive language, specify callback methods using the capitalization as it is shown in this list.

SAS.pyplot(*plot*<, filename='matplot.svg'><, filepath=None><, filetype='svg'><, **kwargs>)

renders a plot of a matplotlib.pyplot object or of any object with an equivalent `savefig(fname, format=None, **kwargs)` method. By default, the image is written to a temporary file called `matplot.svg` in the Work directory. To save the graphic file, provide the `FILEPATH=` argument. To specify an alternate name that differs from the default file name, provide the `FILENAME=` argument. Enclose the file name and file path values in quotation marks. You can specify any file format that is supported by ODS, such as PNG or GIF. For a complete list of file formats, see [“Supported Image Formats for Output Destinations” in SAS Output Delivery System: Procedures Guide](#). Optionally, specify `**kwargs`

parameters that SAS passes to the `savefig` method. For more information, see the [matplotlib.pyplot.savefig](#) documentation.

Note: If the Python package that you use does not include the `savefig` method, then you can use the `SAS.renderImage` callback method to display an image that you write to a file using another Python method.

Requirement Import the `matplotlib.pyplot` module to create and manipulate a plot object. Check with your SAS administrator to verify whether this module is available.

Notes SVG images might not render in the SAS session when the HTML ODS destination is used.

Support for the `FILETYPE=` and `**kwargs` arguments was added in 2023.03. Also, in this release, the default graphics file type is now SVG (rather than PNG).

Example See “[Example 6: Render a Python Plot](#)” on page 1613.

SAS.renderImage(filename)

renders an image file that is created by using a Python method other than `savefig`. Provide the fully qualified file name of the image file. Supported image formats are the same as those that are supported by ODS output. For more information, see “[Supported Image Formats for Output Destinations](#)” in *SAS Output Delivery System: Procedures Guide*.

Note Support for this callback method was added in 2023.03.

SAS.sasfnc("function-name"<, argument-1<, argument-2...>>)

Beginning with 2023.10, this method calls the specified SAS or PROC FCMP function. If the function requires arguments, specify those in a comma-delimited list after the function name.

You can assign the function result to a Python variable.

Example Call the `YEAR` function for a variable, `mydate`, that contains a SAS date value. Assign the result to `py_var`.

```
py_var = SAS.sasfnc("YEAR", mydate)
```

SAS.show(object,<title>,<count>,<kwargs>)

Beginning with 2025.03, this method is used instead of `print()` to display the information to the Results (Listing) instead of the SAS log.

object a Pandas dataframe, series, index, native Python type (including `str`, `int`, `float`, `list`, `dict`), or a matplotlib plot.

title string to use for the `title2` SAS statement, which will be the title on the output page.

count the number of items of the object to show, for other than numeric types. The default is 5.

kwargs these are passed to `pyplot`, if provided. This is not used for other types of objects.

Requirement Import the `matplotlib.pyplot` module to create and manipulate a plot object. Check with your SAS administrator to verify whether this module is available.

SAS.showMLA(*results*, <show_name={True|False}>)

Beginning with 2025.05, this method is used to display results from SAS Python models in the PYTHON procedure results.

Specify the results from a SAS Python model fit (`model.details_`) or from the `customize_results()` method (`model.customize_results()`). For more information about the `customize_results()` method, see [SAS Viya: Managing Plots in Python Automatic Graphics](#).

`SHOW_NAME=` shows the name of the output object as part of the title text. The default setting is `False`. Use this option to determine the output object names to be used with the `include_tables`, `exclude_tables`, and `order` options in `customize_results()`.

SAS.submit("SAS code")

submits and executes the specified string of SAS code when the callback method is issued within a block of Python statements.

Note: The code being submitted has to be a complete program. If it is not a complete program, the following message is displayed.

WARNING: Detected incomplete program in submit(). That step did not complete.

Example Create a SAS data set called Test.

```
SAS.submit("data test; x='python'; y=2; run;")
```

SAS.symget("SAS-macro-variable")

returns the value of the macro variable that you specify. You can assign the value to a Python variable.

Example Assign the value of macro variable `Myvar` to the Python variable `py_var`.

```
py_var=SAS.symget("myvar")
```

SAS.symput("SAS-macro-variable", value)

assigns a value to a macro variable. The value that you provide can be a string, a numeric value, or a variable name.

Macro variables that you create with this callback method have global scope.

You can assign the return code of the `SAS.symput` call to a Python variable.

Example Assign the value of the Python variable `py_var` to the macro variable `Myvar`. Assign the return code to the variable `rc`.

```
rc=SAS.symput("myvar", py_var)
```

SAS.logMessage("message","message-type")

Writes a well formed message to the SAS Log from PROC Python. Depending on the message type, this method reports a NOTE (in blue), WARNING (in green), or ERROR (in red). The method takes two parameters:

message	the message that should be written to the SAS log, this is required
messageType	the log message type in SAS. This is an optional parameter that defaults to NOTE. You can specify WARNING or ERROR when needed.

Example Send a warning to the user that a resource could not be found

```
proc python;
submit;
<Python code>
SAS.logMessage('Resource not found.', 'warning')
<more Python code>
endsubmit;
run;
```

Callback Methods That Transfer Data within the PYTHON Procedure

The following callback methods can be used to transfer data between Pandas DataFrames and SAS data sets.

**SAS.df2sd(data-frame, 'SAS-data-set'<(data-set-options)>,<
 datetimes={name:'value' pair(s)}><, outfmts={name:'value' pair(s)}><,
 labels={name:'value' pair(s)}><, char_lengths={name:'value' pair(s)}>)**
 transfers data from a Pandas DataFrame to a SAS data set. If the SAS data set already exists, the data set is overwritten by default.

DATETIMES= associates column names with the values 'date' or 'time'. The values in the column are used to generate DATE or TIME values, respectively, in SAS. If you do not specify 'date' or 'time' values with a column of DataFrame date-time values, then these values are loaded into SAS as DATETIME values.

OUTFMTS= associates column names with SAS formats that are defined for the output SAS data set.

LABELS= assigns labels to columns in the resulting SAS data set.

CHAR_LENGTHS= assigns lengths to character values that are loaded into the SAS data set. These length values override any lengths that are calculated based on the values in the DataFrame column.

The optional arguments DATETIMES=, OUTFMTS=, LABELS=, and CHAR_LENGTHS= each take a Python dictionary.

Examples Transfer data from Pandas DataFrame **my_frame** to SAS data set **Work.Cars**.

```
ds = SAS.df2sd(my_frame, 'work.cars')
```

Transfer data from Pandas DataFrame `my_frame` to SAS data set `Work.Cars_exact` with additional options.

```
ds = SAS.df2sd(my_frame, 'cars_exact',
               labels={'length': 'Length (IN)', 'mpg_City': 'MPG (City)',
                      'EngineSize': 'Engine Size (L)',
                      'MPG_Highway': 'MPG (Highway)',
                      'Weight': 'Weight (LBS)', 'WheelBase': 'Wheelbase (IN)'},
               outfmts={'Msrp': 'DOLLAR8.', 'invoice': 'DOLLAR8.'},
               char_lengths={'Model': 40, 'type': 8})
```

SAS.sd2df('SAS-data-set<(data-set-options)>')

transfers data from a SAS data set or view to a Pandas DataFrame. If desired, specify additional data set options to control the data that you pass to the Pandas DataFrame.

The resulting object from this callback method call is a Pandas DataFrame.

Examples Export data from `Sashelp.Cars` to a Pandas DataFrame.

```
df = SAS.sd2df('sashelp.cars')
```

Export the first five rows from `Sashelp.Cars` to a Pandas DataFrame.

```
df = SAS.sd2df('sashelp.cars(obs=5)')
```

Export selected columns from `Sashelp.Cars` to a Pandas DataFrame.

```
df = SAS.sd2df('sashelp.cars(keep=make model)')
```

Callback Methods That Control Log Output When You Transfer Data

Use these callback methods to modify how or whether output is displayed in the SAS log when you transfer data using the `SAS.df2sd` and `SAS.sd2df` callback methods:

SAS.hideLOG(True | False)

specifies whether to redirect Python output from PROC PYTHON to the `sas2py.log` file in the Work directory rather than to the SAS log. Output from callback methods that call the SAS module, such as `SAS.symput`, still displays in the SAS log.

This callback method uses PROC PRINTTO to redirect output. Therefore, you see information about PROC PRINTTO in the SAS log when PROC PYTHON completes.

Default True

Example Call `SAS.hideLOG` to redirect Python output

```
proc python;
submit;
#Send Log output to sas2py.log
SAS.hideLOG(True)
<Python code>
#Send output back to the SAS log
```

```
SAS.hideLOG(False)
<Python code>
endsubmit;
run;
```

SAS.printLOG()

displays in the SAS log all accumulated Python output since the SAS.hideLOG callback method was called.

You can use this callback method to verify whether any errors have occurred since calling SAS.hideLOG.

Example View Python output since calling SAS.hideLOG

```
proc python;
submit;
SAS.hideLOG(True)
<Python code>
SAS.printLOG()
<Additional Python code, still redirected to external Work file>
endsubmit;
run;
```

SAS.clearLOG()

This method deletes all of the currently cached LOG output, if any. Subsequent output from data transfer methods continue to cache their output, if set to hide that output, based upon the setting of hideLOG().

Example Clear the cached log output

```
proc python;
submit;
SAS.hideLOG(True)
<Python code>
SAS.printLOG()
<Python code>
SAS.clearLOG()
endsubmit;
run;
```

Examples: PYTHON Procedure

Example 1: Specify Input from an External Python Script

Features: FILENAME statement

PROC PYTHON statement, INFILE= option

Details

This example shows two ways to run a Python script called `my_script.py` in the `/usr/scripts` directory. The first program specifies the file name and location directly in the `INFILE=` option of the `PROC PYTHON` statement. The second program defines a fileref to the `my_script.py` file and uses that fileref in the `INFILE=` option.

Here are the contents of `my_script.py`:

```
print("Hello World")
y = "Hi!"
def my_function():
    print("Inside the my_script.py script")
```

Program: Specify a Python Script Location with the INFILE= Option

Execute the PROC PYTHON statement and specify the Python script location and name. Upload your file to a location in SAS Data and AI Studio. Use the `INFILE=` option to provide the name and location of the Python script. SAS executes the Python statements in `my_script.py` when you invoke the `INFILE=` option. Variables and functions that are created by running the script are available inside the `SUBMIT` and `ENDSUBMIT` block in `PROC PYTHON`.

```
proc python infile='/tmp/my_script.py';
submit;
my_function()
print("y = " + y)
endsubmit;
run;
```

Program: Specify a Python Script as a Fileref

```
filename script filesrv folderPath='/Users/myuser/My Folder/tmp/'
filename='my_script.py';

proc python infile=script;
submit;
my_function()
print("y = " + y)
endsubmit;
run;
```

Program Description

Define a fileref that specifies your Python script and its location. Upload your Python script into SAS Data and AI Studio. Use the FILENAME statement to create a fileref called `script` for your Python script.

```
filename script filesrvc folderPath='/Users/myuser/My Folder/tmp/'
filename='my_script.py';
```

Execute the PROC PYTHON statement and specify the fileref that identifies the Python script. The script is invoked when you specify the INFILE= option in the PROC PYTHON statement. Variables and functions that are created by running the script are available inside the SUBMIT and ENDSUBMIT block in PROC PYTHON.

```
proc python infile=script;
submit;
my_function()
print("y = " + y)
endsubmit;
run;
```

Example Code 46.1 SAS Log After Invoking an External Python Script

```
80  filename script '/tmp/my_script.py';
81
82  proc python infile=script;
83  submit
NOTE: Python initialized.
Python 3.6.5 (default, Jun 24 2020, 17:52:46)
[GCC 8.3.1 20191121 (Red Hat 8.3.1-5)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>>
83  !      ;
84  my_function()
85  print("y = " + y)
86  endsubmit;
87  run;
88  %let SAS_workpath = %sysfunc(pathname(work));
>>>
Hello World
>>>
>>>
>>>
Inside the my_script.py script
y = Hi!
>>>
>>>
NOTE: PROCEDURE PYTHON used (Total process time):
      real time          1.68 seconds
      cpu time           0.05 seconds
```

Example 2: Observe the Impact of Restarting and Terminating the PYTHON Procedure

Features: PROC PYTHON statement, RESTART option
PROC PYTHON statement, TERMINATE option

Details

This example demonstrates the effects of maintaining the status of the PYTHON procedure across multiple procedure calls. You can see that the status of the PYTHON procedure is reset when you use the RESTART or TERMINATE options.

Program

```
proc python;
submit;
y = "Hi!"
def my_function():
    print("Inside the function")
endsubmit;
run;

proc python;
submit;
my_function()
print("y = " + y)
endsubmit;
run;

proc python restart;
submit;
x = "New variable x is created"
print("x = " + x)
#References to function, variable from previous PROC PYTHON call
my_function()
print("y = " + y)
endsubmit;
run;

proc python terminate;
submit;
#Reference to variable defined in previous PROC PYTHON call
print("x = " + x)
x = "Updating the value for x"
print("x = " + x)
```



```
#Redefining a function called my_function
def my_function():
    print("Inside the proc step")
endsubmit;
run;

proc python;
submit;
print("x = " + x)
my_function()
endsubmit;
run;
```

Program Description

Call PROC PYTHON and define a variable *y* and a function called *my_function*.

Assign a string value to the variable *y*. Define a function called *my_function* that prints another string.

```
proc python;
submit;
y = "Hi!"
def my_function():
    print("Inside the function")
endsubmit;
run;
```

Refer to the previously defined function and variable in a subsequent PYTHON procedure call. Call the function *my_function* and print the value of the variable *y*. Because the status of the PYTHON procedure is maintained across subsequent procedure calls, the definition of these items is maintained.

```
proc python;
submit;
my_function()
print("y = " + y)
endsubmit;
run;
```

Use the RESTART option to reset the PYTHON procedure status. The status of the PYTHON procedure resets at the beginning of this procedure call. Define a new variable called *x*. Previously defined variables or functions are no longer available. References to these result in an error in the SAS log and the procedure call exits.

```
proc python restart;
submit;
x = "New variable x is created"
print("x = " + x)
#References to function, variable from previous PROC PYTHON call
my_function()
print("y = " + y)
endsubmit;
run;
```

Use the TERMINATE option to stop maintaining the status of PROC PYTHON.

When you use the TERMINATE option, the status of the PYTHON procedure resets

at the end of the procedure call. The variable `x` is still defined from the previous call to PROC PYTHON. Specify a new definition for the function `my_function`. The previous definition of this function was lost when you specified the RESTART option in the previous call to PROC PYTHON.

```
proc python terminate;
submit;
#Reference to variable defined in previous PROC PYTHON call
print("x = " + x)
x = "Updating the value for x"
print("x = " + x)
#Redefining a function called my_function
def my_function():
    print("Inside the proc step")
endsubmit;
run;
```

Verify that the status of PROC PYTHON was reset at the end of the preceding call to PROC PYTHON. Because you specified the TERMINATE option in the previous call to PROC PYTHON, no variables or function definitions were maintained after that procedure call ended. References to `x` result in an error in the SAS log and the procedure call exits.

```
proc python;
submit;
print("x = " + x)
my_function()
endsubmit;
run;
```

Output 46.4 *Output That Shows the Effect of the RESTART and TERMINATE Options*

```

80  proc python;
81  submit
NOTE: Python initialized.
Python 3.6.5 (default, Jun 24 2020, 17:52:46)
[GCC 8.3.1 20191121 (Red Hat 8.3.1-5)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>>
81  !          ;
82  y = "Hi!"
83  def my_function():
84      print("Inside the function")
85  endsubmit;
86  run;
87  %let SAS_workpath = %sysfunc(pathname(work));
>>>
>>>
>>>
NOTE: PROCEDURE PYTHON used (Total process time):
      real time          3.88 seconds
      cpu time           0.06 seconds

88
89  proc python;
90  submit
NOTE: Resuming Python state from previous PROC PYTHON invocation.
90  !          ;
91  my_function()
92  print("y = " + y)
93  endsubmit;
94  run;
>>>
Inside the function
y = Hi!

>>>
>>>
NOTE: PROCEDURE PYTHON used (Total process time):
      real time          0.00 seconds
      cpu time           0.00 seconds

95
96  proc python restart;
97  submit
NOTE: Python terminated.
NOTE: Previous Python state destroyed.
NOTE: Python initialized.
Python 3.6.5 (default, Jun 24 2020, 17:52:46)
[GCC 8.3.1 20191121 (Red Hat 8.3.1-5)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>>

```

```

97 !      ;
98 x = "New variable x is created"
99 print("x = " + x)
100 #References to function, variable from previous PROC PYTHON call
101 my_function()
102 print("y = " + y)
103 endsubmit;
104 run;
105 %let SAS_workpath = %sysfunc(pathname(work));
>>>
x = New variable x is created

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<string>", line 4, in <module>
NameError: name 'my_function' is not defined

>>>
>>>
NOTE: PROCEDURE PYTHON used (Total process time):
      real time          1.36 seconds
      cpu time           0.08 seconds

106
107 proc python terminate;
108 submit
NOTE: Resuming Python state from previous PROC PYTHON invocation.
108!      ;
109 #Reference to variable defined in previous PROC PYTHON call
110 print("x = " + x)
111 x = "Updating the value for x"
112 print("x = " + x)
113 #Redefining a function called my_function
114 def my_function():
115     print("Inside the proc step")
116 endsubmit;
117 run;
>>>
x = New variable x is created
x = Updating the value for x

>>>
>>>
NOTE: Python terminated.
NOTE: PROCEDURE PYTHON used (Total process time):
      real time          0.00 seconds
      cpu time           0.00 seconds

118
119 proc python;
120 submit
NOTE: Python initialized.
Python 3.6.5 (default, Jun 24 2020, 17:52:46)
[GCC 8.3.1 20191121 (Red Hat 8.3.1-5)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>>
120!      ;
121 print("x = " + x)
122 my_function()
123 endsubmit;
124 run;
125 %let SAS_workpath = %sysfunc(pathname(work));
>>>
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<string>", line 1, in <module>
NameError: name 'x' is not defined

>>>
>>>
NOTE: PROCEDURE PYTHON used (Total process time):
      real time          1.14 seconds
      cpu time           0.04 seconds

```

Example 3: Submit SAS Code and Call a SAS Function within PROC PYTHON

Features: SAS.submit
SAS.sasfnc

Details

This example demonstrates how the SAS.submit callback method can be used to submit SAS code within a block of Python code. In this example, use the SAS.submit callback method to create a data set called Test. The Test data set is available outside of the call to PROC PYTHON. This example also shows how you use the SAS.sasfnc callback method to call the SAS function SHA256HEX to display the SHA256 digest obtained for the value 'abc'.

Program

```
proc python;
submit;
var1 = "'python'"
var2 = 2

SAS.submit("data work.test; x={}; y={}; run;".format(var1,var2))

var3 = SAS.sasfnc("sha256hex","abc")
print("var3 = " + var3)

endsubmit;
run;

proc print data=test;
run;
```

Program Description

Begin your call to PROC PYTHON and use the SUBMIT statement to start a block of Python code. Assign values to two Python variables, var1 and var2. The string value takes an additional set of quotation marks so that the value 'python' is sent to the DATA step.

```
proc python;
submit;
```

```
var1 = 'python'
var2 = 2
```

Call the SAS.submit callback method and create a SAS data set Work.Test. Assign values to X and Y by using the Python format function.

```
SAS.submit("data work.test; x={}; y={}; run;".format(var1,var2))
```

Call a SAS function using the SAS.sasfnc callback method. Call the [SHA256HEX function](#) for the string "abc" and assign the return value to the Python variable var3. Call the Python print function to display a string that shows the value of var3.

```
var3 = SAS.sasfnc("sha256hex", "abc")
print("var3 = " + var3)
```

End the block of Python code, and then print the generated data set Work.Test.

Call the ENDSUBMIT statement and end the Python code block. Follow the call to PROC PYTHON with a PROC PRINT call that refers to the Work.Test data set. Note that the data set is available outside of the PROC PYTHON call.

```
endsubmit;
run;

proc print data=test;
run;
```

Output 46.5 *Output from SAS.submit and SAS.sasfnc Calls in PROC PYTHON*

Obs	x	y
1	python	2

Example Code 46.2 Log from SAS.submit and SAS.sasfnc Calls in PROC PYTHON

```

80  proc python;
81  submit
NOTE: Python initialized.
Python 3.6.5 (default, Jun 24 2020, 17:52:46)
[GCC 8.3.1 20191121 (Red Hat 8.3.1-5)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>>
81  !          ;
82  var1 = "python"
83  var2 = 2
84  SAS.submit("data work.test; x={}; y={}; run;".format(var1,var2))
85  var3 = SAS.sasfnc("sha256hex","abc")
86  print("var3 = " + var3)
87  endsubmit;
88  run;
89  %let SAS_workpath = %sysfunc(pathname(work));
90  data work.test; x='python'; y=2; run;
NOTE: The data set WORK.TEST has 1 observations and 2 variables.
NOTE: DATA statement used (Total process time):
      real time          0.00 seconds
      cpu time           0.00 seconds

>>>
var3 = BA7816BF8F01CFEA414140DE5DAE2223B00361A396177A9CB410FF61F20015AD
>>>
>>>
NOTE: PROCEDURE PYTHON used (Total process time):
      real time          3.29 seconds
      cpu time           0.04 seconds

91
92  proc print data=test;
93  run;
NOTE: There were 1 observations read from the data set WORK.TEST.
NOTE: The PROCEDURE PRINT printed page 1.
NOTE: PROCEDURE PRINT used (Total process time):
      real time          0.02 seconds
      cpu time           0.02 seconds

```

Example 4: Create a SAS Macro Variable in PROC PYTHON

Features:

- Defining SAS macro variables
- SAS.symget callback method
- SAS.symput callback method

Details

This example shows how you can create a SAS macro variable and assign the value of a Python variable to it. The macro variable has global scope and can be referenced outside of the PROC PYTHON call.

Program

```
%let myvar = Jane Doe;

proc python;
submit;
x = SAS.symget('myvar')
print('macro variable myvar = ' + x)

py_var = 'Inside python'
SAS.symput('macrovar', py_var)
endsubmit;
run;

%put &=macrovar;
%put &=myvar;
```

Program Description

Assign a macro variable, Myvar, outside of a call to PROC PYTHON.

```
%let myvar = Jane Doe;
```

Call PROC PYTHON and assign the value of Myvar to a Python variable. Assign the value of the SAS macro variable Myvar to the Python variable x by using the SAS.symget callback method. Use the Python print function to display the value of x.

```
proc python;
submit;
x = SAS.symget('myvar')
print('macro variable myvar = ' + x)
```

Define a Python variable and assign a string value. Define the Python variable py_var and assign a string value to it. Then call SAS.symput to replace the value of py_var with the value of the Macrovar variable.

```
py_var = 'Inside python'
SAS.symput('macrovar', py_var)
endsubmit;
run;
```

Refer to the global macro variable Macrovar outside of PROC PYTHON. Macro variables that you define with SAS.symput are global variables. Therefore, you can

refer to them outside of the call to PROC PYTHON. The macro variable Myvar is also still available.

```
%put &=macrovar;
%put &=myvar;
```

Example Code 46.3 SAS Log from Macro Variable Example

```
80 %let myvar = Jane Doe;
81
82 proc python;
83 submit
NOTE: Python initialized.
Python 3.6.5 (default, Jun 24 2020, 17:52:46)
[GCC 8.3.1 20191121 (Red Hat 8.3.1-5)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>>
83 !      ;
84 x = SAS.symget('myvar')
85 print('macro variable myvar = ' + x)
86 py_var = 'Inside python'
87 SAS.symsput('macrovar', py_var)
88 endsubmit;
89 run;
90 %let SAS_workpath = %sysfunc(pathname(work));
>>>
macro variable myvar = Jane Doe
>>>
>>>
NOTE: PROCEDURE PYTHON used (Total process time):
      real time          2.64 seconds
      cpu time           0.05 seconds

91
92 %put &=macrovar;
MACROVAR=Inside python
93 %put &=myvar;
MYVAR=Jane Doe
```

Example 5: Invoke an FCMP Function in PROC PYTHON

Features: SAS.sasfnc callback method
PROC FCMP

Details

The SAS `sasfnc` function can be used to call a function that you created by using the FCMP procedure. Here the example invokes the FCMP function `fnc1` to sum two numbers.

Program

```
proc fcmp outlib=work.funcs.trial;
    function fnc1(num1, num2);
        return(num1+num2);
    endsub;
quit;

options cmplib=work.funcs;

proc python;
submit;
x = SAS.sasfnc("fnc1",1,2)
print("x = ", int(x))
endsubmit;
run;
```

Program Description

Define a function called `fnc1` that returns the sum of two numeric arguments. Use the FCMP procedure to define the `fnc1` function. In this case, `fnc1` returns the sum of two arguments that are passed to it.

```
proc fcmp outlib=work.funcs.trial;
    function fnc1(num1, num2);
        return(num1+num2);
    endsub;
quit;
```

Specify the `CMPLIB=` system option. This system option specifies a SAS data set that contains compiler subroutines to include during program compilation.

```
options cmplib=work.funcs;
```

Call the `fnc1` function from a block of Python commands. The output from the block of Python commands is printed to the SAS log.

```
proc python;
submit;
x = SAS.sasfnc("fnc1",1,2)
print("x = ", int(x))
endsubmit;
run;
```

Example Code 46.4 SAS Log from PROC FCMP Example

```

78  proc fcmp outlib=work.funcs.trial;
79      function fnc1(num1, num2);
80          return(num1+num2);
81      endsub;
82  quit;
NOTE: Function fnc1 saved to work.funcs.trial.
NOTE: PROCEDURE FCMP used (Total process time):
      real time          0.02 seconds
      cpu time           0.05 seconds

83
84  options cmplib=work.funcs;
85
86  proc python;
87  submit;
NOTE: Python initialized.
Python 3.6.5 (default, Jun 24 2020, 17:52:46)
[GCC 8.3.1 20191121 (Red Hat 8.3.1-5)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>>
90  x = SAS.sasfnc("fnc1",1,2)
91  print("x = ", int(x))
92  endsubmit;
93  run;
94  %let SAS_workpath = %sysfunc(pathname(work));
>>>
x = 3
>>>
>>>
NOTE: PROCEDURE PYTHON used (Total process time):
      real time          2.29 seconds
      cpu time           0.11 seconds

```

Example 6: Render a Python Plot

Features: Importing the matplotlib.pyplot module
 SAS.pyplot callback method

Details

Use the `SAS.pyplot` callback method to render a plot object that you create and modify by using functions from the `matplotlib.pyplot` Python module. In this example, data is created using array functions from the NumPy library.

Program

```

proc python echo;
  interactive;
  import matplotlib.pyplot as plt
  import numpy as np

  plt.style.use('_mpl-gallery-nogrid')

  # make data:
  X, Y = np.meshgrid([1, 2, 3, 4], [1, 2, 3, 4])
  angle = np.pi / 180 * np.array([[15., 30, 35, 45],
                                   [25., 40, 55, 60],
                                   [35., 50, 65, 75],
                                   [45., 60, 75, 90]])

  amplitude = np.array([[5, 10, 25, 50],
                        [10, 15, 30, 60],
                        [15, 26, 50, 70],
                        [20, 45, 80, 100]])

  U = amplitude * np.sin(angle)
  V = amplitude * np.cos(angle)

  # plot:
  fig, ax = plt.subplots()

  ax.barbs(X, Y, U, V, barbcolor='C0', flagcolor='C0', length=7,
           linewidth=1.5)

  ax.set(xlim=(0, 4.5), ylim=(0, 4.5))

  SAS.pyplot(plt)

endinteractive;
run;

```

Program Description

Call PROC PYTHON and import necessary libraries and modules. The ECHO option causes interactively submitted lines of Python code to be included in the SAS log. The INTERACTIVE statement begins a block of interactively submitted Python commands. The Python `import` commands load the `matplotlib.pyplot` module and the NumPy library. The `matplotlib.pyplot` module enables you to access plotting functions. The NumPy library enables you to work with arrays. Access functions within the `pyplot` module with the `plt` alias, and access NumPy functions using the `np` alias.

```

proc python echo;
  interactive;
  import matplotlib.pyplot as plt
  import numpy as np

```

Specify the plot style. Specify a setting about how to render your plot. For more information about the `pyplot` module, see the [matplotlib documentation](#).

```

plt.style.use('_mpl-gallery-nogrid')

```

Generate data for plotting. Use the NumPy functions to generate data for your plot. For more information about NumPy functions, see the [NumPy documentation](#).

```
# make data:
X, Y = np.meshgrid([1, 2, 3, 4], [1, 2, 3, 4])
angle = np.pi / 180 * np.array([[15., 30, 35, 45],
                                [25., 40, 55, 60],
                                [35., 50, 65, 75],
                                [45., 60, 75, 90]])

amplitude = np.array([[5, 10, 25, 50],
                      [10, 15, 30, 60],
                      [15, 26, 50, 70],
                      [20, 45, 80, 100]])

U = amplitude * np.sin(angle)
V = amplitude * np.cos(angle)
```

Specify the contents of your plot. Call pyplot functions to design your plot by using the data that you generated above and by specifying additional plot options.

```
# plot:
fig, ax = plt.subplots()

ax.barbs(X, Y, U, V, barbcolor='C0', flagcolor='C0', length=7,
         linewidth=1.5)

ax.set(xlim=(0, 4.5), ylim=(0, 4.5))
```

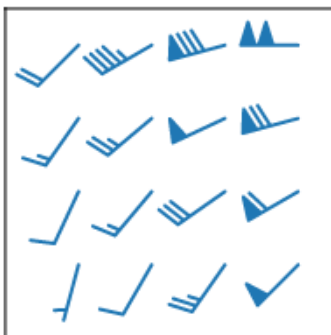
Render your plot with the SAS.pyplot callback method. Call the SAS.pyplot callback method for the plot object, referred to as `plt`. To save the resulting plot to a PNG file, specify the optional `filename=` and `filepath=` arguments. For example, to save the plot to a file called `MyPlot.png` in your data directory, you could specify `SAS.pyplot(plt,filename="MyPlot",filepath="/myuser/data")`.

```
SAS.pyplot(plt)
```

End your call to PROC PYTHON. Call the `ENDINTERACTIVE` statement to end the block of interactively submitted Python code.

```
endinteractive;
run;
```

Output 46.6 Output from SAS.pyplot Callback Method



QDEVICE Procedure

Overview: QDEVICE Procedure	1617
What Does the QDEVICE Procedure Do?	1618
Syntax: QDEVICE Procedure	1618
PROC QDEVICE Statement	1619
DEVICE Statement	1622
PRINTER Statement	1624
VAR Statement	1625
Usage: QDEVICE Procedure	1636
Variables Common to All Reports	1636
Create a GENERAL Report	1637
Create a FONT Report	1640
Create a DEVOPTION Report	1642
Create a LINESTYLE Report	1644
Create a RECTANGLE Report	1645
Create a SYMBOL Report	1647
Examples: QDEVICE Procedure	1648
Example 1: Generate a Report for the Default Display Device	1648
Example 2: Generate a General Report for All Devices	1649
Example 3: Generate a Report for SAS/GRAPH Device Drivers and Universal Printers	1651
Example 4: Generate a Report for the Default Printer	1652
Example 5: Generate a Font Report	1654
Example 6: Generate a Device Option Report	1658
Example 7: Specify a User Library and Catalog for a Report	1660

Overview: QDEVICE Procedure

What Does the QDEVICE Procedure Do?

The QDEVICE procedure produces reports about graphics devices and universal printers. You can use the information in these reports to determine the best device or printer to use for a specific application.

Six different reports are available. These reports summarize information such as color support, default output sizes, margin sizes, resolution, supported fonts, hardware symbols, hardware fill types, hardware line styles, and device options.

You can send the output of this procedure to the SAS log or to an output SAS data set.

Syntax: QDEVICE Procedure

Default: If you do not specify a report to create, a printer, or a device, the procedure generates a GENERAL report for the default display device if you are running SAS using the windowing environment, or the default universal printer in other modes.

Note: You can specify more than one DEVICE, PRINTER, or VAR statement. Statements are processed in the order in which they are specified.

PROC QDEVICE

< REPORT=GENERAL | FONT | DEVOPTION | LIFESTYLE | RECTANGLE |
SYMBOL >

< OUT=SAS-data-set >

< CATALOG=catalog-name >

< DEVLOC=GDEVICE_n | SASHELP | _ALL_ | libref >

< REGISTRY=SASHELP | SASUSER >

< SUPPORT=YES | NO | ALL >

< UNITS=IN | CM; >

DEVICE <device-name(s)> < _ALL_ > < _HTML_ > < _LISTING_ > < _RTF_ >;

PRINTER <printer-name(s)> < _ALL_ > < _PCL_ > < _PDF_ > < _PRINTER_ > <
PS >;

VAR variable-1 <variable-2 ...>;

Statement	Task	Example
PROC QDEVICE	Specify an (optional) output data set, which report to generate, which locations to search, whether to list supported or non-supported features, and sizing information	Ex. 1, Ex. 2, Ex. 3, Ex. 4, Ex. 5, Ex. 6, Ex. 7
DEVICE	Specify which SAS/GRAPH devices to generate a report for	Ex. 2, Ex. 7
PRINTER	Specify which universal printers to generate a report for	Ex. 6, Ex. 3
VAR	Specify the information (variables) to include in the generated reports	Ex. 5

PROC QDEVICE Statement

Controls the input data that is examined, the contents of a report, and the type of output to create.

Syntax

PROC QDEVICE

```
< REPORT=GENERAL | FONT | DEVOPTION | LIFESTYLE | RECTANGLE |  
SYMBOL >  
< OUT=SAS-data-set >  
< CATALOG=catalog-name >  
< DEVLOC=GDEVICEn | SASHELP | _ALL_ | libref >  
< REGISTRY=SASHELP | SASUSER >  
< SUPPORT=YES | NO | ALL >  
< UNITS=IN | CM >;
```

Summary of Optional Arguments

CATALOG=catalog-name

specifies the name of a SAS device catalog to search for a device.

DEVLOC=GDEVICEn | SASHELP | _ALL_ | libref

specifies the library or libraries where the device catalog is located.

OUT=SAS-data-set

specifies an output SAS data set for the report.

REGISTRY=SASHELP | SASUSER

specifies which section of the SAS registry to search when querying a universal printer.

REPORT=DEVOPTION | FONT | GENERAL | LIFESTYLE | RECTANGLE | SYMBOL

specifies the type of report that you want to generate.

SUPPORT=YES | NO | ALL

specifies whether to report only supported features, only unsupported features, or all features.

UNITS=IN | CM

specifies whether the values for the certain variables are reported in inches (IN) or centimeters (CM) in the GENERAL report.

Optional Arguments

CATALOG=*catalog-name*

specifies the name of a SAS device catalog to search for a device.

Aliases C=

CAT=

Default DEVICES

Interaction The CATALOG= option works with the DEVLOC= option. When you specify the CATALOG= option, SAS looks in the library that is specified by the DEVLOC= option (for example, sashelp.mycatalog).

Example [“Example 7: Specify a User Library and Catalog for a Report” on page 1660](#)

DEVLOC=GDEVICEn | SASHELP | _ALL_ | *libref*

specifies the library or libraries where the device catalog is located. You can specify these libraries:

GDEVICEn

specifies to search one of the SAS/GRAPH device libraries for a device. *n* can be 0–9.

SASHELP

specifies to search the Sashelp library for a device.

ALL

specifies to search the libraries Gdevice0 – Gdevice9 and the Sashelp library, in this order, for a device. All occurrences of a device from any of these libraries are reported.

libref

specifies a valid SAS library to search.

Defaults If you do not specify the DEVLOC= option, libraries are searched in the following order:

1. Gdevice0–Gdevice9
2. Sashelp

The first occurrence of the specified device is reported unless you specify DEVLOC=_ALL_.

Interaction The DEVLOC= option works with the CATALOG= option. When you specify the CATALOG= option, SAS looks in the library that is specified by the DEVLOC= option (for example, sashelp.mycatalog).

Example [“Example 7: Specify a User Library and Catalog for a Report” on page 1660](#)

OUT=SAS-data-set

specifies an output SAS data set for the report.

Default SAS Log

REGISTRY=SASHELP | SASUSER

specifies which section of the SAS registry to search when querying a universal printer.

SASHELP

search the SASHELP section of the registry.

SASUSER

search the SASUSER section of the registry.

Alias REG=

Default SASHELP and SASUSER

REPORT=DEVOPTION | FONT | GENERAL | LIFESTYLE | RECTANGLE | SYMBOL

specifies the type of report that you want to generate. You can request only one type of report. See [“Valid Variables for All Reports” on page 1625](#) for the descriptions of the variables that are included in each report.

DEVOPTION

produces a report of the hardware device options supported by the specified device.

Restriction This report is unavailable for universal printers.

FONT

produces a report of all system and device-resident fonts supported by the specified device or printer.

See See [“SAS/GRAPH, System, and Device-Resident Fonts” in SAS/GRAPH: Reference](#) for a description of font categories.

GENERAL

produces a report of general information about the specified device or printer. This report includes information such as destination, margin sizes, default font information, resolution, color information, and size by pixels. This is the default report.

LIFESTYLE

produces a report of the hardware line styles supported by the specified device.

Restriction This report is unavailable for universal printers.

RECTANGLE

produces a report of the hardware fill types supported by the specified device.

Restriction This report is unavailable for universal printers.

SYMBOL

produces a report of the hardware symbols supported by the specified device.

Restriction This report is unavailable for universal printers.

Default GENERAL

See See [“Valid Variables for All Reports” on page 1625](#) for the descriptions of the variables that are included in each report.

SUPPORT=YES | NO | ALL

specifies whether to report only supported features, only unsupported features, or all features.

YES

reports only hardware features and options that are supported.

NO

reports only the hardware features and options that are not supported.

ALL

reports both supported and unsupported hardware features and options.

Default YES

Restriction This option applies only to devices when producing a DEVOPTION, LIFESTYLE, RECTANGLE, or SYMBOL report.

UNITS=IN | CM

specifies whether the values for the certain variables are reported in inches (IN) or centimeters (CM) in the GENERAL report.

The HEIGHT, WIDTH, LEFT, LMIN, RIGHT, RMIN, BOTTOM, BMIN, TOP, TMIN, HRES, and VRES variables are reported in inches or centimeters. HRES and VRES values are reported as pixels-per-inch or pixels-per-centimeter.

Default IN

Restriction This option applies only when producing a GENERAL report.

DEVICE Statement

Specifies which SAS/GRAPH devices to generate a report for.

Requirement: You must specify at least one device name, _ALL_, _HTML_, _LISTING_, or _RTF_.

Syntax

DEVICE *<device-name(s)>* *<_ALL_>* *<_HTML_>* *<_LISTING_>* *<_RTF_>*;

Optional Arguments

device-name(s)

specifies the device for which you want to generate a report. Separate device names with a blank space. Enclose device names that contain spaces in quotation marks. You can use the wildcard characters * and ? to report all devices with similar names.

*

* represents any number of characters in that position of the device name.

Requirement Device names that contain wildcard characters must be enclosed in quotation marks.

Note You can specify the * and ? wildcard characters in the same device name.

Example 'svg*'

Example [“Example 3: Generate a Report for SAS/GRAPH Device Drivers and Universal Printers” on page 1651](#)

?

? represents one character in the device name. You can specify multiple consecutive ? characters in the device name.

Requirement Device names that contain wildcard characters must be enclosed in quotation marks.

Note You can specify the * and ? wildcard characters in the same device name.

Example 'tiff?300'

ALL

generates reports for all devices.

HTML

determines the default device that is used by the ODS HTML destination and generates a report for that device. The device is based on the default HTML version that is assigned in the ODS key of the SAS registry.

LISTING

determines the default device that is used by the ODS Listing destination and generates a report for that device. The default value is a host-specific display device.

RTF

determines the default device that is used by the ODS RTF destination and generates a report for that device.

PRINTER Statement

Specifies which universal printers to generate a report for.

Requirement: You must specify at least one printer name, `_ALL_`, `_PCL_`, `_PDF_`, `_PRINTER_`, or `_PS_`.

Syntax

```
PRINTER <printer-name(s)> < \_ALL\_ > < \_PCL\_ > < \_PDF\_ > < \_PRINTER\_ > < \_PS\_ >;
```

Optional Arguments

printer-name(s)

specifies the universal printers for which you want to generate a report. If the printer name contains spaces, enclose the printer name in quotation marks. Separate printer names with a blank space. You can use the wildcard characters `*` and `?` to report all printers with similar names.

`*`

indicates to report all printers that match any number of characters in the position of the `*` in the printer name.

Requirement Printer names that contain wildcard characters must be enclosed in quotation marks.

Note You can specify the `*` and `?` wildcard characters in the same printer name.

Example `'pcl*'` reports on the printers `pcl4`, `pcl5`, `pcl5c`, and `pcl5e`

`?`

indicates to report all printers that match the printer name, where the character in the `?` position can be any character. You can use multiple `?` characters in *printer-name* to represent the same number of characters in the same position in the printer name.

Requirement Printer names that contain wildcard characters must be enclosed in quotation marks.

Note You can specify the `*` and `?` wildcard characters in the same printer name.

Example `'tiff?'` reports on the printers `tiffa` and `tiffk`. It does not report on the printer `tiff` because the `tiff` printer is only four characters.

Example [“Example 3: Generate a Report for SAS/GRAPH Device Drivers and Universal Printers” on page 1651](#)

ALL

generates reports for all universal printers.

PCL

determines the default printer that is used by the ODS PCL destination and generates a report for that printer.

PDF

determines the default printer that is used by the ODS PDF destination and generates a report for that printer.

PRINTER

determines the default printer that is used by the ODS PRINTER destination and generates a report for that printer.

PS

determines the default printer that is used by the ODS PS destination and generates a report for that printer.

VAR Statement

Specifies which variables to include in a report. The order of the variables in the report is determined by the order in which they are specified in the VAR statement.

Default: If you do not specify a VAR statement, all of the variables for the report are included in a default order.

Tip: If you specify the VAR statement, you must specify at least one variable. Otherwise, the statement is ignored.

Syntax

VAR *variable-1* <*variable-2* ...>;

Valid Variables for All Reports

DESC

displays the default description of the device or printer.

LOCATION

for the device entry that was found, displays the physical location of the Gdevice0-Gdevice9 library, the Sashelp library, or the library that is specified by the DEVLOC= option. For universal printers, this variable displays the SAS registry (SASHELP or SASUSER) where the printer was found.

NAME

displays the name of the device or printer.

TYPE

displays the type of device or printer. Here are the types of devices and printers:

- Graph Device
- Printer Interface Device
- Shortcut Device
- System Display
- System Metafile
- Universal Previewer
- Universal Printer

See [“Device Categories and Modifying Default Output Attributes” in SAS/GRAPH: Reference](#)

DEVOPTION Report Variables

For more information, see [“Create a DEVOPTION Report” on page 1642](#).

BIT

displays the bit position in the DEVOPTS string for the corresponding device option.

See [“DEVOPTS= Device Parameter” in SAS/GRAPH: Reference](#)

BITSTRING

displays the bit pattern of the corresponding device option.

See [“DEVOPTS= Device Parameter” in SAS/GRAPH: Reference](#)

DESC

displays the default description of the device or printer.

LOCATION

for the device entry that was found, displays the physical location of the Gdevice0-Gdevice9 library, the Sashelp library, or the library that is specified by the DEVLOC= option.

NAME

displays the name of the device or printer.

ODESC

displays the descriptions of the hardware options in effect for the device.

OPTION

displays the names of the hardware options in effect for the device.

SUPPORT

displays the device options.

Interaction The value of the SUPPORT variable is affected by the SUPPORT= option in the PROC QDEVICE statement. If SUPPORT=YES, the report shows the supported device options. If SUPPORT=NO, the report shows device options that are not supported. If SUPPORT=ALL, the report shows all supported and non-supported device options.

TYPE

displays the type of device or printer.

Here are the types of devices and printers:

- Graph Device
- Printer Interface Device
- Shortcut Device
- System Display
- System Metafile

See [“Device Categories and Modifying Default Output Attributes” in SAS/GRAPH: Reference](#)

FONT Report Variables

For more information, see [“Create a FONT Report” on page 1640](#).

ALIAS

reports an alternate name for a font that is registered by the FONTREG procedure.

DESC

displays the default description of the device or printer.

FONT

displays the name of the default font.

See [“Default Fonts” in SAS/GRAPH: Reference](#)

[“Variable Labels in a FONT Report” on page 1641](#)

FSTYLE

displays the font style, such as Roman or Italic, for each font and font weight, in an output data set.

Restriction When the report output is directed to the SAS log, the FONT report displays only font family names, such as Courier, Helvetica, Times, and so on. The specific font style is not reported.

Note If the font name is acquired from a CHARREC list in a device entry, the style is not available. See [“CHARREC= Device Parameter” in SAS/GRAPH: Reference](#) for more information.

See [“Variable Labels in a FONT Report” on page 1641](#)

FTYPE

displays the type of font, such as Printer Resident, System, or Software.

Note The values for the FTYPE variable in the output data set are Printer Resident, System, or Software. The value Software appears only in a FONT report for a SAS/GRAPH device that has hardware font support disabled.

FWEIGHT

displays the font weight, such as Normal or Bold, for each font and font style, in an output data set.

Restriction When the report output is directed to the SAS log, the FONT report displays only font family names, such as Courier, Helvetica, Times, and so on. The specific font weight is not reported.

Note If the font name is acquired from a CHARREC list in a device entry, the weight is not available. See [“CHARREC= Device Parameter” in SAS/GRAPH: Reference](#) for more information.

FVERSION

specifies the font version.

Restriction When the report output is directed to the SAS log, the FONT report displays only font family names, such as Courier, Helvetica, Times, and so on. The specific font version is not reported.

LOCATION

for the device entry that was found, displays the physical location of the Gdevice0-Gdevice9 library, the Sashelp library, or the library that is specified by the DEVLOC= option. For universal printers, this variable displays the SAS registry (SASHELP or SASUSER) where the printer was found.

NAME

displays the name of the device or printer.

TYPE

displays the type of device or printer. Here are the types of devices and printers:

- Graph Device
- Printer Interface Device
- Shortcut Device
- System Display
- System Metafile
- Universal Previewer
- Universal Printer

See [“Device Categories and Modifying Default Output Attributes” in SAS/GRAPH: Reference](#)

GENERAL Report Variables

For more information, see [“Create a GENERAL Report” on page 1637](#).

ALIAS

reports an alternate name for a font that is registered by the FONTREG procedure.

ANIMATION

specifies whether animation is active, enabled, disabled, or unsupported for a Universal Printer:

Active	indicates that the graphs in the ODS HTML output are grouped together in the animation. Animate=Start and Animate=Stop are ignored.
Enabled	indicates that animation is supported. Animate=Start and Animate=Stop must be specified to start and stop the animation.
Disabled	indicates that animation is supported but has been disabled for the device or printer.
Unsupported	indicates that animation is not supported for the device or printer.

BMIN

displays the minimum size of the bottom margin.

BOTTOM

displays the current size of bottom margin.

CLRSPACE

displays the type of color support (color space) such as RGB, RGBA, CMYK, HLS, and so on.

See [“Color-Naming Schemes” in SAS/GRAPH: Reference](#)

COLS

displays the number of horizontal columns in the output.

See [“Cells” in SAS/GRAPH: Reference](#)

COMPRESSION

indicates the condition under which compression is used. Compression can always be in effect, in effect only when specified by a compression option, or never used by the device or printer.

Here are the compression values on the report:

Always	indicates that compression is always in effect.
Option	indicates that compression is specified by the UPRINTCOMPRESSION system option or the COMPRESS= option in the ODS PRINTER statement.
Never	specifies that compression is never used by the device or printer.

COMPMETHOD

indicates the compression method that is used if compression is supported by device or printer.

Note When Compression is Never and no compression is available, the Compression Method value does not appear in the SAS log.

DESC

displays the default description of the device or printer.

DEST

displays the default destination of the device or universal printer if the device or printer does not send output directly to a printer or a display device. If the device sends output directly to a printer or a display device, the value of DEST is blank.

A destination can have a blank value when output is going to a monitor, or on Windows, the output is going to a printer.

EMBEDDING

indicates whether font embedding is supported.

ALWAYS Font embedding is always in effect for the device or printer.

OPTION The FONTEMBEDDING system option controls whether font embedding is supported.

NEVER Font embedding is not supported.

FHEIGHT

displays the height, in the respective units, of the default font.

Note If the font name is acquired from a CHARREC list in a device entry, the height is not available. See [“CHARREC= Device Parameter” in SAS/GRAPH: Reference](#) for more information.

FONT

displays the name of the default font.

See [“Default Fonts” in SAS/GRAPH: Reference](#)

FORMAT

displays the output format type (for example, EMF, EMF Plus, EMF Dual, PostScript, GIF, Host Display, and so on).

See [“Commonly Used Devices” in SAS/GRAPH: Reference](#)

FSTYLE

displays the style of the default font (for example, Roman, Regular, and so on).

Interaction The results of specifying the FSTYLE variable in a GENERAL report where the output is directed to the SAS log differs from the results that you get when you specify the FSTYLE variable for a FONTS report. In a GENERAL report, the font style is reported to the SAS log. In a FONT report, the SAS log report displays only the font family names. The specific font style is not reported.

Note If the font name is acquired from a CHARREC list in a device entry, the style is not available. See [“CHARREC= Device Parameter” in SAS/GRAPH: Reference](#) for more information.

FWEIGHT

displays the weight of the default font (for example, Normal, Medium, and so on).

Interaction The results of specifying the FWEIGHT variable in a GENERAL report where the output is directed to the SAS log differs from the results that you get when you specify the FWEIGHT variable for a FONTS report. In a GENERAL report, the font weight is reported to the SAS log. In a FONT report, the SAS log report displays only the font family names. The specific font weights are not reported.

Note If the font name is acquired from a CHARREC list in a device entry, the weight is not available. See [“CHARREC= Device Parameter” in SAS/GRAPH: Reference](#) for more information.

FVERSION

specifies the version of the font.

HEIGHT

displays the default vertical height of output (in UNITS) sent to the device or printer.

HRES

displays the horizontal resolution (pixels per UNIT) of output sent to the device or printer. Horizontal resolution is calculated by the formula $HRES = XPIXELS / WIDTH$.

Interaction If either the HRES or VRES variables are specified in the VAR statement, the horizontal and vertical resolutions are displayed together in the SAS log using the label XxY Resolution. In an output data set, HRES and VRES are reported separately.

See [“Using the XPIXELS=, XMAX=, YPIXELS=, and YMAX= Graphics Options to Set the Resolution for the Traditional Devices” in SAS/GRAPH: Reference](#)

IOTYPE

displays the type of input/output used by the device or printer (for example, DISK, PRINTER, PIPE, GTERM, and so on).

See [“FILENAME Statement” in SAS Global Statements: Reference](#) and [“DEVTYPE= Device Parameter” in SAS/GRAPH: Reference](#)

LEFT

displays the size of the left margin of output.

LMIN

displays the minimum left margin.

LOCATION

for the device entry that was found, displays the physical location of the Gdevice0-Gdevice9 library, the Sashelp library, or the library that is specified by the DEVLOC= option. For universal printers, this variable displays the SAS registry (SASHELP or SASUSER) where the printer was found.

MAXCOLORS

displays the maximum number of colors that are supported by the device or printer.

See [“Maximum Number of Colors Displayed on a Device” in SAS/GRAPH: Reference](#)

MODULE

specifies the name of the device driver module.

NAME

displays the name of the device or printer.

PROTOTYPE

displays the prototype (model) that was used to define the universal printer.

See [“Universal Printing” in SAS Programmer's Guide: Essentials](#)

RIGHT

displays the size of the right margin.

RMIN

displays the minimum size of the right margin.

ROWS

displays the number of vertical rows in the output.

See [“Cells” in SAS/GRAPH: Reference](#)

TMIN

displays the minimum top margin of output.

TOP

displays the size of the top margin.

TYPE

displays the type of device or printer. Here are the types of devices and printers:

- Graph Device
- Printer Interface Device
- Shortcut Device
- System Display
- System Metafile
- Universal Previewer
- Universal Printer

See [“Device Categories and Modifying Default Output Attributes” in SAS/GRAPH: Reference](#)

UNITS

displays the units (IN for inches or CM for centimeters) in which sizes are displayed. In the SAS log, the value of UNITS appears respectively, as inches or centimeters. In an output data set, the value of UNITS appears as IN or CM.

Interaction If the VAR statement does not specify any variables for size, margins, or resolution, the SAS log shows the units that are used to measure size, margins or resolution. Here is an example:

Name: EMF
Units: inches

If the VAR statement specifies any variables for size, margins, or resolution, the SAS log shows the units with the value. Here is an example:

XxY Resolution: 96x96 pixels per inch

VISUAL

displays the visual color type (for example, Indexed Color, Direct Color, True Color, Monochrome, or Gray Scale).

VRES

displays the vertical resolution (pixels per UNIT) of output sent to the device or printer.

Vertical resolution is calculated by the formula $VRES=YPIXELS/HEIGHT$.

Interaction If either the HRES or VRES variables are specified in the VAR statement, the horizontal and vertical resolutions are displayed together in the SAS log using the label XxY Resolution. In an output data set, HRES and VRES are reported separately.

See [“Using the XPIXELS=, XMAX=, YPIXELS=, and YMAX= Graphics Options to Set the Resolution for the Traditional Devices” in SAS/GRAPH: Reference](#)

WIDTH

displays the width of output (in UNITS) sent to device or printer.

XPIXELS

displays the width of the output in pixels.

See [“XPIXELS= Device Parameter and Graphics Option” in SAS/GRAPH: Reference](#) and [“Using the XPIXELS=, XMAX=, YPIXELS=, and YMAX= Graphics Options to Set the Resolution for the Traditional Devices” in SAS/GRAPH: Reference](#)

YPIXELS

displays the height of the output in pixels.

See [“YPIXELS= Device Parameter and Graphics Option” in SAS/GRAPH: Reference](#) and [“Using the XPIXELS=, XMAX=, YPIXELS=, and YMAX= Graphics Options to Set the Resolution for the Traditional Devices” in SAS/GRAPH: Reference](#)

LINESTYLE Report Variables

For more information, see [“Create a LINESTYLE Report” on page 1644](#).

DESC

displays the default description of the device or printer.

LINE

displays the line styles supported by the device or printer.

Interaction In a SAS log LINESTYLE report, the LINE and SUPPORT variables are reported together. If either the LINE variable or the SUPPORT variable is specified in the VAR statement, the line styles are reported using the Supported Line Styles or Unsupported Line Styles variable labels.

See [“Line Types” in SAS/GRAPH: Reference](#)

LOCATION

displays the physical location of the Gdevice0-Device9 or Sashelp library that contains the Devices catalog where the device entry was found or the library that is specified by the DEVLOC= option.

NAME

displays the name of the device.

SUPPORT

displays the device lines styles.

Interaction The value of the SUPPORT variable is affected by the SUPPORT= option in the PROC QDEVICE statement. If SUPPORT=YES, the report shows the supported line styles. If SUPPORT=NO, the report shows line styles that are not supported. If SUPPORT=ALL, the report shows all supported and non-supported line styles.

TYPE

displays the type of device or printer:

- Graph Device
- Printer Interface Device
- Shortcut Device
- System Display
- System Metafile

See [“Device Categories and Modifying Default Output Attributes” in SAS/GRAPH: Reference](#)

RECTANGLE Report Variables

For more information, see [“Create a RECTANGLE Report” on page 1645](#).

DESC

displays the default description of the device or printer.

FILL

displays the hardware fill types that are supported by the device.

Interaction In a SAS log RECTANGLE report, the FILL and SUPPORT variables are reported together. If either the FILL variable or the SUPPORT variable is specified in the VAR statement, the fill names are

reported using either the label Supported Hardware Fills or the label Unsupported Hardware Fills.

See [“PATTERN Statement” in SAS/GRAPH: Reference](#)

LOCATION

displays the physical location of the Gdevice0-Device9 or Sashelp library that contains the Devices catalog where the device entry was found or the library that is specified by the DEVLOC= option.

NAME

displays the name of the device.

SUPPORT

displays the hardware fills.

Interaction The value of the SUPPORT variable is affected by the SUPPORT= option in the PROC QDEVICE statement. If SUPPORT=YES, the report shows the supported hardware fills. If SUPPORT=NO, the report shows hardware fills that are not supported. If SUPPORT=ALL, the report shows all supported and non-supported hardware fills.

TYPE

displays the type of device or printer.

Here is a list of types:

- Graph Device
- Printer Interface Device
- Shortcut Device
- System Display
- System Metafile

See [“Device Categories and Modifying Default Output Attributes” in SAS/GRAPH: Reference](#)

SYMBOL Report Variables

For more information, see [“Create a SYMBOL Report” on page 1647](#).

DESC

displays the default description of the device or printer.

LOCATION

displays the physical location of the Gdevice0-Device9 or Sashelp library that contains the Devices catalog where the device entry was found or the library that is specified by the DEVLOC= option.

NAME

displays the name of the device.

SUPPORT

displays the device or printer symbols.

Interaction The value of the SUPPORT variable is affected by the SUPPORT= option in the PROC QDEVICE statement. If SUPPORT=YES, the report shows the supported symbols. If SUPPORT=NO, the report shows symbols that are not supported. If SUPPORT=ALL, the report shows all supported and non-supported symbols.

SYMBOL

specifies the name of hardware symbols.

Interaction In a SAS log SYMBOL report, if either the SYMBOL or SUPPORT variable is specified in the VAR statement, symbol names are reported using the Supported Hardware Symbols label or Unsupported Hardware Symbols label.

See [“SYMBOL, NOSYMBOL Graphics Options” in SAS/GRAPH: Reference](#) and [“SYMBOLS= Device Parameter” in SAS/GRAPH: Reference](#)

TYPE

displays the type of device or printer.

Here is a list of types:

- Graph Device
- Printer Interface Device
- Shortcut Device
- System Display
- System Metafile

See [“Device Categories and Modifying Default Output Attributes” in SAS/GRAPH: Reference](#)

Usage: QDEVICE Procedure

Variables Common to All Reports

You can use the following variables in any of the reports:

- NAME
- DESC
- TYPE
- LOCATION

For a description of the variables, see [“Valid Variables for All Reports” on page 1625](#).

Create a GENERAL Report

The GENERAL report produces a report of general information about the specified device or printer. This information includes margin sizes, default font information, resolution, and color information.

About GENERAL Report Variables

For a description of the variables, see [“GENERAL Report Variables” on page 1628](#).

The following table lists the variables that you can use in a GENERAL report as well as the labels for the variables that are used either in the SAS log or the output data set. If you do not specify the VAR statement, the variables appear in the order in which they appear in the table.

Variable	SAS Log Label	Output Data Set Label
NAME	Name	NAME OF DEVICE OR PRINTER
DESC	Description	DESCRIPTION OF DEVICE OR PRINTER
MODULE	Module	DRIVER MODULE
TYPE	Type	TYPE OF DEVICE OR PRINTER
LOCATION	"Device Catalog" for a device "Registry" for a printer	LOCATION OF DEVICE OR PRINTER DEFINITION
PROTOTYPE	Prototype	PRINTER PROTOTYPE
FONT	Default Typeface	FONT TYPEFACE DEFAULT
ALIAS	Typeface Alias	FONT TYPEFACE ALIAS
FSTYLE	Font Style	FONT STYLE DEFAULT
FWEIGHT	Font Weight	FONT WEIGHT DEFAULT

Variable	SAS Log Label	Output Data Set Label
FHEIGHT	Font Height	FONT HEIGHT DEFAULT
FVERSION	Font Version	FONT VERSION
MAXCOLORS	Maximum Colors	MAXIMUM NUMBER OF SUPPORTED COLORS
VISUAL	Visual Color	TYPE OF VISUAL COLOR
CLRSPACE	Color Support	TYPE OF COLOR SUPPORT
DEST	Destination	OUTPUT DESTINATION DEFAULT
IOTYPE	I/O Type	TYPE OF I/O DEFAULT
FORMAT	Data Format	OUTPUT DATA FORMAT
HEIGHT	Height	HEIGHT OF OUTPUT
WIDTH	Width	WIDTH OF OUTPUT
UNITS	Units ¹	UNITS FOR SIZE, MARGINS AND RESOLUTION
YPIXELS	Ypixels	VERTICAL PIXELS
XPIXELS	Xpixels	HORIZONTAL PIXELS
ROWS	Rows (vpos)	ROWS
COLS	Columns (hpos)	COLUMNS
LEFT	Left Margin	LEFT MARGIN
LMIN	Minimum Left Margin	MINIMUM LEFT MARGIN
RIGHT	Right Margin	RIGHT MARGIN
RMIN	Minimum Right Margin	MINIMUM RIGHT MARGIN
BOTTOM	Bottom Margin	BOTTOM MARGIN
BMIN	Minimum Bottom Margin	MINIMUM BOTTOM MARGIN

Variable	SAS Log Label	Output Data Set Label
TOP	Top Margin	TOP MARGIN
TMIN	Minimum Top Margin	MINIMUM TOP MARGIN
HRES	XxY Resolution	HORIZONTAL PIXELS PER UNIT
VRES	XxY Resolution	VERTICAL PIXELS PER UNIT
COMPRESSION	Compression Enabled	COMPRESSION ENABLED
COMPMETHOD	Compression Method	COMPRESSION METHOD
EMBEDDING	Font Embedding	FONT EMBEDDING SUPPORT
ANIMATION	Animation	ANIMATION SUPPORT

¹ If the type of units is displayed with the value of a variable, such as 0 inches for the left margin, the Units label is not displayed in the output to the SAS log.

System Options That Affect the Value of Size Variables

For universal printers, the values of the HEIGHT, WIDTH, LEFT, RIGHT, BOTTOM, TOP, LMIN, RMIN, BMIN, and TMIN variables are affected by the settings of the PAPERSIZE, LEFTMARGIN, RIGHTMARGIN, BOTTOMMARGIN, and TOPMARGIN SAS system options. The default paper size is determined by the SAS locale (which affects the default size for universal printers). For SAS/GRAPH devices, these variable values are not affected by the system option settings.

For more information, see [SAS System Options: Reference](#).

Example: GENERAL Report

The following QDEVICE procedure creates a GENERAL report for the SVG universal printer:

```
proc qdevice;
  printer svg;
run;
```

Here is the GENERAL report in the SAS log:

```

78  proc qdevice; printer svg; run;
      Name: SVG
      Description: Scalable Vector Graphics 1.1
      Module: SASPDSVG
      Type: Universal Printer
      Registry: SASHELP
      Prototype: SVG 1.1
      Default Typeface: Cumberland AMT
      Typeface Alias: Courier
      Font Style: Regular
      Font Weight: Normal
      Font Height: 8 points
      Font Version: Version 1.03
      Maximum Colors: 16777216
      Visual Color: Direct Color
      Color Support: RGBA
      Destination: sasprt.svg
      I/O Type: DISK
      Data Format: SVG
      Height: 6.25 inches
      Width: 8.33 inches
      Ypixels: 600
      Xpixels: 800
      Rows(vpos): 50
      Columns(hpos): 114
      Left Margin: 0 inches
      Minimum Left Margin: 0 inches
      Right Margin: 0 inches
      Minimum Right Margin: 0 inches
      Bottom Margin: 0 inches
      Minimum Bottom Margin: 0 inches
      Top Margin: 0 inches
      Minimum Top Margin: 0 inches
      XxY Resolution: 96x96 pixels per inch
      Compression Enabled: Never
      Compression Method: Deflate
      Font Embedding: Option
      Animation: Enabled

```

Create a FONT Report

The FONT report produces a report of all system and device-resident fonts that are supported by the specified device or printer.

About FONT Report Variables

For a description of the variables, see [“FONT Report Variables” on page 1627](#).

The following table lists the variables that you can use in a FONT report as well as the labels for the variables that are used either in the SAS log or the output data set. If you do not specify the VAR statement, the variables appear in the order in which they appear in the table.

Variable	SAS Log Label	Output Data Set Label
NAME	Name	NAME OF DEVICE OR PRINTER
DESC	Description	DESCRIPTION OF DEVICE OR PRINTER
TYPE	Type	TYPE OF DEVICE OR PRINTER
LOCATION	"Device Catalog" for a device "Registry" for a printer	LOCATION OF DEVICE OR PRINTER DEFINITION
FONT	Depends on the font or type	FONT TYPEFACE
ALIAS	(alias: <i>alias</i>) to the right of the font name	FONT TYPEFACE ALIAS
FTYPE	Depends on the type of font	FONT TYPE
FSTYLE	None	FONT STYLE
FWEIGHT	None	FONT WEIGHT
FVERSION	None	FONT VERSION

Variable Labels in a FONT Report

When you specify the FONT, FTYPE, FSTYLE, or the FWEIGHT variables in a FONT report, the variable labels that appear in the SAS log vary. The variable labels in the output data set are always the same: FONT TYPEFACE DEFAULT, FONT TYPE, FONT STYLE DEFAULT, and FONT WEIGHT DEFAULT, respectively.

If the VAR statement specifies one or more of the variables FONT, FTYPE, FSTYLE, FWEIGHT, or FVERSION, the SAS log reports only the font type labels and the font family names. The font styles, weights, and versions are not reported to the SAS log. The font type label that appears is dependent on the font type. Some example labels are Supported Font Typefaces, Supported Resident Typefaces, Supported TrueType Typefaces, and Supported Type1 Typefaces.

Example: FONT Report

The following QDEVICE procedure creates a FONT report for the PDF graphics device:

```
proc qdevice report=font;
```

```
device activex;
run;
```

Here is a partial FONT report in the SAS log:

```

Name: PDF
Description: Portable Document Format Version 1.5
Type: Shortcut Device
Device Catalog: your-SAS-path/sashelp
Supported Resident Typefaces: Adobe Caslon
                             Adobe Caslon Oldstyle
                             Adobe Caslon Small Caps
                             Adobe Caslon Small Caps Oldstyle
                             Adobe Garamond
                             Adobe Garamond Oldstyle
                             Adobe Garamond Small Caps Oldstyle
                             Adobe Garamond Titling
                             Adobe Wood Type Ornaments One
                             Adobe Wood Type Ornaments Two
                             Albertus MT
                             Americana
                             Amigo
                             Antique Olive
                             Antique Olive Compact
                             Antique Olive Condensed
                             AvantGarde-Book
                             AvantGarde-BookOblique
                             AvantGarde-Demi
                             AvantGarde-DemiOblique
                             Barmeno
                             Bernhard Modern
                             Birch
                             Blackoak
                             Bodoni
                             Bodoni Poster
                             Bodoni Poster Compressed
                             Bookman-Demi
                             Bookman-DemiItalic
                             Bookman-Light
                             Bookman-LightItalic
                             Carta
                             Clarendon
```

Create a DEVOPTION Report

The DEVOPTION report produces a report of the hardware device options that are supported by the specified device. This report is unavailable for universal printers.

About DEVOPTION Report Variables

For a description of the variables, see [“DEVOPTION Report Variables” on page 1626](#).

The following table lists the variables that you can use in a DEVOPTION report as well as the labels for the variables that are used either in the SAS log or the output

data set. If you do not specify the VAR statement, the variables appear in the order in which they appear in the table.

Variable	SAS Log Label	Output Data Set Label
NAME	Name	NAME OF DEVICE OR PRINTER
DESC	Description	DESCRIPTION OF DEVICE OR PRINTER
TYPE	Type	TYPE OF DEVICE OR PRINTER
LOCATION	Device Catalog	LOCATION OF DEVICE OR PRINTER DEFINITION
BIT	Bit Position	DEVICE OPTION BIT POSITION
BITSTRING	Bit Pattern	DEVICE OPTION BIT PATTERN
OPTION	Device Option	DEVICE OPTION NAME
ODESC	Option Description	DEVICE OPTION DESCRIPTION
SUPPORT	Support	DEVICE OPTION SUPPORT

Example: DEVOPTION Report

The following QDEVICE procedure creates a DEVOPTION report for the SASMF graphics device, reporting supported options because the default for SUPPORT is YES:

```
proc qdevice report=devooption;
    device sasemf;
run;
```

Here is a partial listing of the report in the SAS log:

```

      Name: SASEMF
      Description: Version 9.3 Enhanced Metafile Format
      Type: Shortcut Device
      Device Catalog: your-SAS-path/sashelp
      Bit Position: 0
      Bit Pattern: 8000000000000000
      Device Option: GDCIRCLEARC
      Option Description: Hardware is capable of drawing circles
      Support: Yes
      Bit Position: 1
      Bit Pattern: 4000000000000000
      Device Option: GDPIEFILL
      Option Description: Device has hardware pie-fill capability
      Support: Yes
      Bit Position: 3
      Bit Pattern: 1000000000000000
      Device Option: GDCRT
      Option Description: Hardware is a CRT or the device acts like a CRT
      Support: Yes
      Bit Position: 5
      Bit Pattern: 0400000000000000
      Device Option: GDPOLYGONFILL
      Option Description: Device has polygonfill capability
      Support: Yes
      Bit Position: 7
      Bit Pattern: 0100000000000000
      Device Option: GDRGB
      Option Description: Hardware is capable of defining colors in one or more color
      spaces
      Support: Yes
      Bit Position: 8
      Bit Pattern: 0080000000000000
      Device Option: GDMBPOLY
      Option Description: Hardware can draw polygons with multiple boundaries
      Support: Yes

```

Create a LINSTYLE Report

The LINSTYLE report produces a report of the hardware (dashed) line styles that are supported by the specified device.

About LINSTYLE Report Variables

For a description of the variables, see [“LINSTYLE Report Variables” on page 1633](#).

The following table lists the variables that you can use in a LINSTYLE report as well as the labels for the variables that are used either in the SAS log or the output data set. If you do not specify the VAR statement, the variables appear in the order in which they appear in the table.

Variable	SAS Log Label	Output Data Set Label
NAME	Name	NAME OF DEVICE OR PRINTER

Variable	SAS Log Label	Output Data Set Label
DESC	Description	DESCRIPTION OF DEVICE OR PRINTER
TYPE	Type	TYPE OF DEVICE OR PRINTER
LOCATION	Device Catalog	LOCATION OF DEVICE OR PRINTER DEFINITION
LINE	Supported Line Styles Unsupported Line Styles	HARDWARE DASHED LINE NUMBER
SUPPORT	Supported Line Styles Unsupported Line Styles	HARDWARE DASHED LINE SUPPORT

Example: LINESTYLE Report

The following QDEVICE procedure creates a LINESTYLE report of the LJ5PS device, reporting the supported line styles because the default for SUPPORT is YES:

```
proc qdevice report=linestyle;
    device svg;
run;
```

Here is the LINESTYLE report in the SAS log:

```

      Name: SVG
    Description: Scalable Vector Graphics 1.1
      Type: Shortcut Device
  Device Catalog: your-SAS-path/sashelp
Supported Line Styles: 1-46
```

Create a RECTANGLE Report

A RECTANGLE report produces a report of the hardware fill types that are supported by the specified device.

About RECTANGLE Report Variables

For a description of the variables, see [“RECTANGLE Report Variables”](#) on page 1634.

The following table lists the variables that you can use in a RECTANGLE report as well as the labels for the variables that are used either in the SAS log or the output data set. If you do not specify the VAR statement, the variables appear in the order in which they appear in the table.

Variable	SAS Log Label	Output Data Set Label
NAME	Name	NAME OF DEVICE OR PRINTER
DESC	Description	DESCRIPTION OF DEVICE or PRINTER
TYPE	Type	TYPE OF DEVICE OR PRINTER
LOCATION	Device Catalog	LOCATION OF DEVICE OR PRINTER DEFINITION
FILL	Supported Hardware Fills	HARDWARE RECTANGLE FILL NAME
SUPPORT	Supported Hardware Fills Unsupported Hardware Fills	HARDWARE RECTANGLE FILL SUPPORT

Example: RECTANGLE Report

The following QDEVICE procedure creates a RECTANGLE report for the SASPRTG universal printer, reporting the supported hardware fills because the default for SUPPORT is YES.

```
proc qdevice report=rectangle;
    device sasprt;
run;
```

Here is the RECTANGLE report in the SAS log:

```

      Name: SASPRTG
    Description: POSTSCRIPT LEVEL 1
      Type: Printer Interface Device
    Device Catalog: your-sas-path\sashelp
Supported Hardware Fills: Empty,Solid
```

SASPRTG is a printer interface device. Because Universal Printing is active, SASPRTG interfaces with the default universal printer. For this reason, the report shows information about the PostScript Level 1 printer as a universal printer.

Create a SYMBOL Report

A SYMBOL report produces a report of the hardware symbols that are supported by the specified device.

About SYMBOL Report Variables

For a description of the variables, see [“SYMBOL Report Variables” on page 1635](#).

The following table lists the variables that you can use in a SYMBOL report as well as the labels for the variables that are used either in the SAS log or the output data set. If you do not specify the VAR statement, the variables appear in the order in which they appear in the table.

Variable	SAS Log Label	Output Data Set Label
NAME	Name	NAME OF DEVICE OR PRINTER
DESC	Description	DESCRIPTION OF DEVICE OR PRINTER
TYPE	Type	TYPE OF DEVICE OR PRINTER
LOCATION	Device Catalog	LOCATION OF DEVICE OR PRINTER DEFINTION
SYMBOL	Supported Hardware Symbols Unsupported Hardware Symbols	HARDWARE SYMBOL NAME
SUPPORT	Support	DEVICE OPTION SUPPORT

Example: SYMBOL Report

The following QDEVICE procedure creates a SYMBOL report for the CGM device, reporting the supported hardware symbols because the default for SUPPORT is YES:

```
proc qdevice report=symbol;
    device cgm;
run;
```

Here is the SYMBOL report in the SAS log:

```
Name: CGM
Description: CGM generator--binary output
Type: Graph Device
Device Catalog: your-sas-path\sashelp
Supported Hardware Symbols: Plus,X,Star
```

Examples: QDEVICE Procedure

Example 1: Generate a Report for the Default Display Device

Features: PROC QDEVICE

Details

The following example creates a General report for the default display device.

Program

```
proc qdevice;
run;
```

Log

If you do not specify the OUT= option, the QDEVICE procedure sends its output to the SAS log.

Example Code 47.1 The SAS Log After Running PROC QDEVICE

NOTE: No DEVICE or PRINTER statement was specified. The default "PDF" printer will be used.

```

      Name: PDF
      Description: Portable Document Format Version 1.5
      Module: SASPPDF
      Type: Universal Printer
      Registry: SASHELP
      Prototype: PDF Version 1.5
Default Typeface: Cumberland AMT
Typeface Alias: Courier
  Font Style: Regular
  Font Weight: Normal
  Font Height: 8 points
  Font Version: Version 1.03
Maximum Colors: 16777216
Visual Color: Direct Color
Color Support: RGBA
  Destination: sasprt.pdf
    I/O Type: DISK
  Data Format: PDF
    Height: 10.5 inches
    Width: 8 inches
    Ypixels: 3150
    Xpixels: 2400
    Rows(vpos): 85
    Columns(hpos): 120
    Left Margin: 0.25 inches
Minimum Left Margin: 0 inches
    Right Margin: 0.25 inches
Minimum Right Margin: 0 inches
    Bottom Margin: 0.25 inches
Minimum Bottom Margin: 0 inches
    Top Margin: 0.25 inches
Minimum Top Margin: 0 inches
  XxY Resolution: 300x300 pixels per inch
Compression Enabled: Option
Compression Method: Deflate
  Font Embedding: Option
  Animation: Unsupported

```

Example 2: Generate a General Report for All Devices

Features: PROC QDEVICE statement option: OUT=
DEVICE statement

Details

The following example creates a General report for all devices and writes the results to WORK.ALLDEVICES.

You can use the `_ALL_` keyword to generate a report for all devices.

Program

```
proc qdevice out=allDevices;
  device _all_;
run;
```

Output

The following image shows a portion of the report as it appears in the Viewtable window.

Output 47.1 The Output Data Set Report for All Devices

	NAME	DESC	MOD...	TYPE
1	BMP	BMP File Fmt--256 colors	SASGDIMG	Graph Device
2	CGM	CGM generator--binary output	SASGDC...	Graph Device
3	CGMC	CGM generator w/colors--binary output	SASGDC...	Graph Device
4	CGMOF93L	CGM for MS landscape w/o hardware li...	SASGDC...	Graph Device
5	CGMOF93P	CGM for MS portrait w/o hardware lines	SASGDC...	Graph Device
6	CGMOF97L	CGM for Office 97 landscape mode	SASGDC...	Graph Device
7	CGMOF97P	CGM for Microsoft Office 97 portrait	SASGDC...	Graph Device
8	CGMOFML	CGM for MS Office w/ Metric Scaling land	SASGDC...	Graph Device
9	CGMOFMP	CGM for MS Office w/ Metric Scaling port	SASGDC...	Graph Device
10	CLJ	HP Color LaserJet Printer--Letter Size	SASGDHPL	Graph Device
11	CLJPS	HP Color LaserJet--PostScript--Letter	SASGDPSL	Graph Device
12	CLJPSA4	HP Color LaserJet--PostScript--A4 size	SASGDPSL	Graph Device
13	DJT2C	DesignJet 2000 Series 24" color HPGL/2	SASGDHPL	Graph Device

Example 3: Generate a Report for SAS/GRAPH Device Drivers and Universal Printers

Features: PROC QDEVICE statement option: OUT=
DEVICE statement
PRINTER statement

Details

The following example creates a General report for all devices that end in EMF and the PDF and SVG? universal printers. The results are written to the WORK.MYREPORT data set. If you do not specify the REPORT= option, the QDEVICE procedure generates a General report.

Program

```
proc qdevice out=myreport;  
  device '*emf';  
  printer pdf 'svg?';  
run;
```

Output

The following image shows a portion of the report as it appears in the Viewtable window.

Output 47.2 The Output Data Set Report for the EMF Device, and PDF and SVG Printers

Obs	NAME	DESC	MODULE	TYPE
1	EMF	Enhanced Metafile Format Plus Extensions	SASGDDMX	Shortcut Device
2	SASEMF	Version 9.3 Enhanced Metafile Format	SASGDDMX	Shortcut Device
3	PDF	Portable Document Format Version 1.5	SASPDPDF	Universal Printer
4	SVGT	Transparent SVG	SASPDSVG	Universal Printer
5	SVGZ	Compressed Scalable Vector Graphics 1.1	SASPDSVG	Universal Printer

Example 4: Generate a Report for the Default Printer

Features: PROC QDEVICE statement
PRINTER statement

Details

By default, printing in SAS under Windows is done by the default Windows printer and not by Universal Printing. Therefore, the results that you see for the QDEVICE procedure when you use the `printer _PRINTER_` statement differ. Under Windows, where the NOUPRINT system option is the default, the report is based on the printer interface device that interfaces with the default Windows printer. Under UNIX, where the UPRINT system option is set, the report is based on the default SAS universal printer.

Because the `REPORT=` option is not specified, the QDEVICE procedure generates a General report. The `OUT=` option is not specified and the results are written to the SAS log. The `_PRINTER_` keyword determines the default printer to report on and generates a report for that printer.

For more information, see [“Setting Up and Managing Universal Printing” in SAS Viya Platform Universal Printing](#)

Program: Windows

```
proc qdevice;
  printer _PRINTER_;
run;
```

Log: Default Printer Report

Example Code 47.2 The SAS Log Output Report for the Default Printer

```

      Name: PostScript Level 1
      Description: Generic PostScript Level 1 Printer
      Module: SASPDPSL
      Type: Universal Printer
      Registry: SASHELP
      Prototype: PostScript Level 1 (Color)
Default Typeface: Cumberland AMT
Typeface Alias: Courier
  Font Style: Regular
  Font Weight: Normal
  Font Height: 8 points
  Font Version: Version 1.03
Maximum Colors: 16777216
Visual Color: Direct Color
Color Support: CMYK
  Destination: sasprt.ps
    I/O Type: DISK
  Data Format: PostScript
    Height: 10 inches
    Width: 7.5 inches
    Ypixels: 3000
    Xpixels: 2250
    Rows(vpos): 81
    Columns(hpos): 112
    Left Margin: 0.5 inches
Minimum Left Margin: 0 inches
    Right Margin: 0.5 inches
Minimum Right Margin: 0 inches
    Bottom Margin: 0.5 inches
Minimum Bottom Margin: 0 inches
    Top Margin: 0.5 inches
Minimum Top Margin: 0 inches
XxY Resolution: 300x300 pixels per inch
Compression Enabled: Never
  Font Embedding: Option
    Animation: Unsupported
```

Program: UNIX

```
proc qdevice;
  printer _PRINTER_;
run;
```

Log: Default Universal Printer Report under UNIX

Example Code 47.3 The SAS Log Output Report for the Default Universal Printer under UNIX

NOTE: The "PostScript Level 1" printer will be used by default with the ODS PRINTER destination.

```

      Name: PostScript Level 1
    Description: Generic PostScript Level 1 Printer
      Module: SASPDPSL
        Type: Universal Printer
      Registry: SASHELP
    Prototype: PostScript Level 1 (Color)
Default Typeface: Cumberland AMT
  Typeface Alias: Courier
    Font Style: Regular
    Font Weight: Normal
    Font Height: 8 points
    Font Version: Version 1.03
Maximum Colors: 16777216
  Visual Color: Direct Color
    Color Support: CMYK
      Destination: sasprt.ps
        I/O Type: DISK
      Data Format: PostScript
        Height: 10 inches
        Width: 7.5 inches
        Ypixels: 3000
        Xpixels: 2250
      Rows(vpos): 81
    Columns(hpos): 112
      Left Margin: 0.5 inches
Minimum Left Margin: 0 inches
      Right Margin: 0.5 inches
Minimum Right Margin: 0 inches
      Bottom Margin: 0.5 inches
Minimum Bottom Margin: 0 inches
      Top Margin: 0.5 inches
Minimum Top Margin: 0 inches
      XxY Resolution: 300x300 pixels per inch
Compression Enabled: Never
      Font Embedding: Option
      Animation: Unsupported
  
```

Example 5: Generate a Font Report

Features:

- PROC QDEVICE statement options
 - REPORT=
 - OUT=
- PRINTER statement

Details

The first example generates a report of all the printer-resident and system fonts available for the printer. The results are written to the Work.Myfonts data set.

The second example is a SAS program that uses a macro, the DATA step, and the PRINT procedure to create a list of fonts for devices.

Program

```
proc qdevice report=font out=myfonts;
  printer 'PDF';
run;
```

Output

The following output shows the first row of the report.

Output 47.3 The Output Data Set for a Font Report

Obs	NAME	DESC	TYPE	LOCATION	FONT	ALIAS	FTYPE	FSTY
1	PDF	Portable Document Format Version 1.5	Universal Printer	SASHELP	Adobe Caslon		Printer Resident	Italic

Program

```
/* Macro FONTLIST - Report fonts supported by a device */

%macro fontlist(type, name);
proc qdevice report=font out=fonts;
  &type &name;
  var font ftype fstyle fweight;
run;

data;
  set fonts;
  drop ftype;
  length type $16;
  if ftype = "System"
  then do;
    if substr(font,2,3) = "ttf" then type = "TrueType";
    else if substr(font,2,3) = "at1" then type = "Adobe Type1";
    else if substr(font,2,3) = "cff" then type = "Adobe CFF/
Type2";
```

```

        else if substr(font,2,3) = "pfr" then type = "Bitstream PFR";
        else type = "System";
    if type ^= "System" then font = substr(font,7,length(font)-6);
        else if substr(font,1,1) = "@"
            then font = substr(font, 2,length(font)-1);
        end;
    else type = "Printer Resident";
run;

proc sort;
    by font;
run;

title "Fonts Supported by the %upcase(&name) &type";

proc print label;
    label fstyle="Style" fweight="Weight" font="Font" type="Type";
run;

%mend fontlist;

%fontlist(device, pdf)

```

Program Description

Create the macro fontlist. The %macro statement begins the macro. The input to the macro is the type, whether it is a device or printer, and the name of the device or printer.

```

/* Macro FONTLIST - Report fonts supported by a device */

%macro fontlist(type, name);

```

Create a data set, fonts, for the device. The macro input variables, type and name, are used to create a Font report using the QDEVICE procedure. The output is written to the data set fonts.

```

proc qdevice report=font out=fonts;
    &type &name;
    var font ftype fstyle fweight;
run;

```

Categorize the font type. Fonts can be a type System, TrueType, Adobe Type1, Adobe CFF/Type2, Bitstream PFR, or Printer Resident.

```

data;
    set fonts;
    drop ftype;
    length type $16;
    if ftype = "System"
        then do;
        if substr(font,2,3) = "ttf" then type = "TrueType";
        else if substr(font,2,3) = "at1" then type = "Adobe Type1";
        else if substr(font,2,3) = "cff" then type = "Adobe CFF/
Type2";

```

```

        else if substr(font,2,3) = "pfr" then type = "Bitstream PFR";
        else type = "System";
    if type ^= "System" then font = substr(font,7,length(font)-6);
        else if substr(font,1,1) = "@"
            then font = substr(font, 2,length(font)-1);
        end;
    else type = "Printer Resident";
run;

```

Sort the font data set by the font name.

```

proc sort;
    by font;
run;

```

Print the fonts for a device or printer.

```

title "Fonts Supported by the %upcase(&name) &type";

proc print label;
    label fstyle="Style" fweight="Weight" font="Font" type="Type";
run;

```

End the macro.

```

%mend fontlist;

```

Use the macro &fontlist to create and output data set for the PCL5c device.

```

%fontlist(device, pdf)

```

Output

Output 47.4 A Partial View of the Fonts Supported by the PDF Device

Fonts Supported by the PDF device

Obs	Font	Style	Weight	Type
1	Adobe Caslon	Italic	Bold	Printer Resident
2	Adobe Caslon	Italic	Normal	Printer Resident
3	Adobe Caslon	Italic	Semi Bold	Printer Resident
4	Adobe Caslon	Roman	Bold	Printer Resident
5	Adobe Caslon	Roman	Normal	Printer Resident
6	Adobe Caslon	Roman	Semi Bold	Printer Resident
7	Adobe Caslon Oldstyle	Italic	Bold	Printer Resident
8	Adobe Caslon Oldstyle	Italic	Normal	Printer Resident
9	Adobe Caslon Oldstyle	Italic	Semi Bold	Printer Resident
10	Adobe Caslon Oldstyle	Roman	Bold	Printer Resident
11	Adobe Caslon Oldstyle	Roman	Normal	Printer Resident
12	Adobe Caslon Oldstyle	Roman	Semi Bold	Printer Resident
13	Adobe Caslon Small Caps	Roman	Semi Bold	Printer Resident
14	Adobe Caslon Small Caps Oldstyle	Roman	Normal	Printer Resident
15	Adobe Garamond	Italic	Bold	Printer Resident
16	Adobe Garamond	Italic	Normal	Printer Resident
17	Adobe Garamond	Italic	Semi Bold	Printer Resident
18	Adobe Garamond	Roman	Bold	Printer Resident
19	Adobe Garamond	Roman	Normal	Printer Resident
20	Adobe Garamond	Roman	Semi Bold	Printer Resident

Example 6: Generate a Device Option Report

Features: PROC QDEVICE statement options
REPORT=
SUPPORT=
OUT=
DEVICE statement

Details

The following example creates a Device Options (DEVOPTIONS) report for the PNG device. The report is written to an output data set.

Program

```
proc qdevice report=devooption support=all out=devop;  
    device pdf;  
run;  
  
proc print data=devop;  
    var bit bitstring option odesc;  
    where Support="Yes";  
    title "Supported PDF Device Options";  
run;
```

Program Description

Report the options for the PDF device. The REPORT=DEVOPTION option specifies to create a device options report. SUPPORT=ALL specifies to report all device features. The option OUT=DEVOP creates the data set WORK.DEVOP. The DEVICE PDF statement specifies to report on the PDF device.

```
proc qdevice report=devooption support=all out=devop;  
    device pdf;  
run;
```

Print the device options report. Printing the WORK.DEVOP data set, the printed report shows the BIT, BITSTRING, OPTION, and ODESC variables for the device options where Support="Yes".

```
proc print data=devop;  
    var bit bitstring option odesc;  
    where Support="Yes";  
    title "Supported PDF Device Options";  
run;
```

Output

Output 47.5 The Output Data Set for the PDF Device

Supported PDF Device Options

Obs	BIT	BITSTRING	OPTION	ODESC
1	0	8000000000000000	GDCIRCLEARC	Hardware is capable of drawing circles
2	1	4000000000000000	GDPIEFILL	Device has hardware pie-fill capability
4	3	1000000000000000	GDCRT	Hardware is a CRT or the device acts like a CRT
5	4	0800000000000000	GDTRANSPARENCY	Device supports transparency
6	5	0400000000000000	GDPOLYGONFILL	Device has polygonfill capability
8	7	0100000000000000	GDRGB	Hardware is capable of defining colors in one or more color spaces
9	8	0080000000000000	GDMBPOLY	Hardware can draw polygons with multiple boundaries
10	9	0040000000000000	GDOPACITY	Hardware is capable of supporting opacity
13	14	0002000000000000	GDHRDCHR	Hardware characters are supported by the device
14	15	0001000000000000	GDXLIMIT	There is no limit on max value allowed for x coordinate
15	16	0000800000000000	GDYLIMIT	There is no limit on max value allowed for y coordinate
17	18	0000200000000000	GDTXJUSTIFY	Hardware is capable of justifying proportional text
20	24	0000008000000000	GDUNICODE	Device supports the use of the Unicode font attribute
21	25	0000004000000000	GDPOLYLINE	Hardware is capable of supporting polylines
24	28	0000000800000000	GDTRUETYPE	Device supports the use of TrueType fonts
32	36	0000000080000000	GDIMAGE	Device is capable of drawing images
35	39	0000000001000000	GDIMGROTATE	Device is incapable of doing image rotation
36	40	0000000000800000	GDTRUECOLOR	Hardware is a 24-bit true color device
37	41	0000000000400000	GDFONTATTR	Device supports setting font attributes
38	42	0000000000200000	GDSCANLNFT	Device will use scan line font rendering
40	44	0000000000080000	GDTEXTCLIP	Hardware will clip text at the device limits
46	50	0000000000002000	GDAUTOSIZE	Autosize text to fit rows and columns
50	54	0000000000000200	GDPOLYOUTLINE	Device draws polygon outlines
51	55	0000000000000100	GDPRINTERPATH	Device temporarily sets printerpath to that of the device name
52	56	0000000000000080	GDOPTPASSTHRU	PAPERSIZE option sets default value of PAPERSIZE goption
53	57	0000000000000040	GDPIEOUTLINE	Driver draws pie slice outlines (empty pies)
54	60	0000000000000008	GDNOROTATE	Force the graphics sublib not to rotate the graph

Example 7: Specify a User Library and Catalog for a Report

Features: PROC QDEVICE statement options
 CATALOG=
 DEVLOC=
 PRINTER statement

Details

This example creates a general report for a GIF printer in a user-specified library and catalog using the DEVLOC= and CATALOG= options in the PROC QDEVICE statement. The results are written to the SAS log.

Program

Assign the device library and catalog

```
libname devlib 'file-path/em';  
proc qdevice report=general devloc=devlib cat=mydevices;  
    device gif;  
run;
```

Log

Example Code 47.4 The SAS Log Report for a Specific Device Library and Catalog

```

144 libname devlib 'file-path/em';
NOTE: Libref DEVLIB was successfully assigned as follows:
      Engine:          V9
      Physical Name:   file-path/em
145 proc qdevice report=general devloc=devlib cat=mydevices;
146     device gif;
147 run;

      Name: GIF
      Description: Graphics Interchange Format RGB Color/Alpha Blending
      Module: SASGDDMX
      Type: Shortcut Device
      Device Catalog: file-path
      Prototype: GIF
Default Typeface: Cumberland AMT
Typeface Alias: Courier
      Font Style: Regular
      Font Weight: Normal
      Font Height: 8 points
      Font Version: Version 1.03
Maximum Colors: 16777216
Visual Color: True Color
Color Support: RGBA
      Destination: sasprt.gif
      I/O Type: DISK
      Data Format: GIF
      Height: 6.25 inches
      Width: 8.33 inches
      Ypixels: 600
      Xpixels: 800
      Rows(vpos): 50
      Columns(hpos): 114
      Left Margin: 0 inches
Minimum Left Margin: 0 inches
      Right Margin: 0 inches
Minimum Right Margin: 0 inches
      Bottom Margin: 0 inches
Minimum Bottom Margin: 0 inches
      Top Margin: 0 inches
Minimum Top Margin: 0 inches
      XxY Resolution: 96x96 pixels per inch
Compression Enabled: Always
Compression Method: LZW
      Font Embedding: Never
      Animation: Enabled

```

R Procedure

Overview: R Procedure	1663
What Does the R Procedure Do?	1664
Concepts: R Procedure	1665
How PROC R Works	1665
Configuration for PROC R	1666
Submitting R Code Using the SUBMIT and ENDSUBMIT Statements	1666
Syntax: R Procedure	1667
PROC R Statement	1667
SUBMIT Statement	1669
ENDSUBMIT Statement	1669
Usage: R Procedure	1669
Submitting R Code within a SUBMIT Block	1669
Using PROC R Callback Methods	1670
Showing Debug Messages for PROC R Functions	1672
Troubleshooting: Removing All Variables from the Environment	1673
Examples: R Procedure	1674
Example 1: Specify Input from an External R Script	1674
Example 2: Observe the Impact of Restarting and Terminating the R Procedure	1676
Example 3: Submit SAS Code and Call a SAS Function within PROC R	1680
Example 4: Create a SAS Macro Variable in PROC R	1682
Example 5: Render R Plots	1684
Example 6: Display Data Frames	1695

Overview: R Procedure

What Does the R Procedure Do?

R is an interpreted, high-level programming language designed primarily for statistical computing and data analysis. The R procedure enables you to run statements from the R programming language within SAS code. You can submit R statements from an external R script or enter R statements directly.

PROC R enables you to perform these tasks:

- run R code within a SAS session
- call most SAS functions within R statements
- submit SAS code from R
- move data between SAS data sets or views and R data frames
- transfer values between SAS macro variables and R variables
- call SAS functions within R statements
- display graphics from R (Base R or ggplot2) in SAS Data and AI Studio results

You can also run R code in batch scripts. For more information, see [“Introduction” in Using the batch Plug-In for the SAS Viya Platform Command-Line Interface](#).

Note: You must install the following R packages to be able to use all functionality:

- R6
 - arrow
 - haven
 - plotly
 - svglite
 - gt
-

Concepts: R Procedure

How PROC R Works

PROC R creates a subprocess from a SAS Compute Server process. This subprocess uses the R deployment that you choose. PROC R enables you to submit R code as one entire program in a SUBMIT block. For more information, see the [SUBMIT statement on page 1669](#).

When you submit a block of R code within the SUBMIT and ENDSUBMIT statements, you see the following behaviors:

- If a statement results in an error or an exception, processing for the block of code stops at the point of the error.
- Compound statements in R are expressions that contain other statements—such as for loops, while loops, if/else blocks, or function definitions—are enclosed in braces {}. While indentation is not syntactically required in R, continuation lines within a compound statement are typically indented consistently for readability. A compound statement ends at its closing brace }, which marks the end of the block. No blank line is required between the end of a compound statement and the next R statement.

The R procedure enables you to use the same subprocess across procedure calls, so that objects or data are available in subsequent R procedure calls. You can choose to terminate an R subprocess and create a new subprocess when desired, either at the beginning or end of an R procedure call. For more information, see [“Example 2: Observe the Impact of Restarting and Terminating the R Procedure” on page 1676](#).

When an R subprocess is created, a module that is part of the R procedure is automatically imported. This module is called SAS, and it enables interaction between SAS and your R instance. The SAS module provides a number of callback methods that you can use to interact between the SAS Compute Server and your R instance. These callback methods enable you to share variables between R and SAS (symget or symput), move data between SAS data sets and R data frames (sd2df or df2sd), invoke SAS functions (sasfnc), and submit SAS code within your R statements (submit). With these callback methods, you can create a workflow that mixes SAS and R programming as you require. For more information, see [“Using PROC R Callback Methods” on page 1670](#).

Configuration for PROC R

There are some configuration requirements that enable PROC R to interact with R on your system. You or a SAS administrator must configure these two environment variables:

- PROC_RPATH points to your R executable. Your R deployment, which includes the R executable, is typically mounted to a file system that is accessible by the SAS Compute Server. The R executable creates the R subprocess that is triggered by the R procedure.
- PROC_M2PATH specifies the location of the SAS.R file that is part of your deployment. This environment variable initially points to the default location of this file. The SAS.R file contains the SAS module that is imported by the R deployment. This file provides the interface for the callback methods that interact with the Compute Server session.

For more information, see [“Configure SAS to Run External Languages Using SAS Configurator for Open Source”](#) in *SAS Viya Platform: Programming Run-Time Servers*.

Your SAS Viya offering is deployed in a locked-down state by default. As a result, PROC R is disabled. Use the ENABLE_AMS= option for R and SOCKET access methods in the LOCKDOWN statement to enable PROC R. For more information, see these topics:

- LOCKDOWN System Option and LOCKDOWN Statement in [“Security Features”](#) in *SAS Viya Platform: Programming Run-Time Servers*
- [“LOCKDOWN”](#) in *SAS Programmer's Guide: Essentials*

Submitting R Code Using the SUBMIT and ENDSUBMIT Statements

When you submit R code using the SUBMIT and ENDSUBMIT statements, your R statements are submitted as a block of code when the end of the procedure call is reached. The output appears in the SAS log after all of the R statements have been processed. If you include any SAS callback methods, then the output from those calls appears in the SAS log prior to the output from the R code.

See Also

[SAS Scripting Wrapper for Analytics Transfer \(SWAT\)](#)

Syntax: R Procedure

```
PROC R < INFILE='file-name' | fileref > < RESTART > < TERMINATE > <
TIMEOUT=n >;
    < SUBMIT; >
        R statements
    < ENDSUBMIT; >
RUN;
```

Statement	Task	Example
PROC R	Executes R statements within SAS code or specifies a file that contains R statements to execute.	Ex. 1, Ex. 2, Ex. 3, Ex. 4, Ex. 5, Ex. 6
SUBMIT	Identifies the beginning of a block of R statements.	Ex. 1, Ex. 2, Ex. 3, Ex. 4, Ex. 5, Ex. 6
ENDSUBMIT	Identifies the end of a block of R statements.	Ex. 1, Ex. 2, Ex. 3, Ex. 4, Ex. 5, Ex. 6

PROC R Statement

Runs R statements within a SAS session.

Restriction: When SAS is in a locked-down state, the R procedure is not available. Your server administrator can re-enable this procedure so that it is accessible in the locked-down state. For more information, see [“Enable or Disable Access Methods” in SAS Viya Platform: Programming Run-Time Servers](#). The RLANG option must be added to the VIYA_LOCKDOWN_USER_METHODS list.

Syntax

Form 1: SUBMIT block

```
PROC R < INFILE='file-name' | fileref > < RESTART > < TERMINATE > <
TIMEOUT=n >;
```

Optional Arguments

INFILE='file-name' | fileref

identifies a source file or fileref that contains R statements to run within a SAS session.

Requirements If you specify a file name, enclose it in single or double quotation marks.

If you use the INFILE= option, then you must specify the path to the R script.

Examples Use the R code in open_data.r:

```
proc r infile='/user/myscripts/open_data.r';
```

Define a fileref for a file that contains R code:

```
filename script 'my_script.R';

proc r infile=script;
```

RESTART

terminates an R subprocess if one is currently running, and creates a new subprocess. The R subprocess is then used for the current and subsequent calls to PROC R.

You specify RESTART at the beginning of a new block of R code.

Example

```
proc r restart;
  submit;
  <R statements>
endsubmit;
run;
```

TERMINATE

stops the R subprocess when the current R procedure call completes. The next call to the R procedure begins a new instance of the R subprocess.

Example

```
proc r terminate;
  submit;
  <R statements>
endsubmit;
run;
```

TIMEOUT=*n*

specifies the number of seconds that can elapse before PROC R terminates when attempting to connect to an R environment.

Default 60

Example

```
proc r timeout=30 <additional options>;
  submit;
  <R statements>
endsubmit;
run;
```

SUBMIT Statement

Identifies the beginning of a block of R code. Enter R statements between the SUBMIT and ENDSUBMIT statements.

Requirement: Each SUBMIT statement must have a corresponding ENDSUBMIT statement.

Syntax

SUBMIT;

ENDSUBMIT Statement

Specified on a line by itself, this statement identifies the end of a block of R statements.

Syntax

ENDSUBMIT;

Usage: R Procedure

Submitting R Code within a SUBMIT Block

You can submit R statements within the PROC R invocation, between SUBMIT and ENDSUBMIT statements. The following code executes a single R print statement:

```
proc r;  
  submit;  
  print("Hello from R")  
  endsubmit;  
run;
```

Output 48.1 *Output*

```
[1] "Hello from R"
NOTE: PROCEDURE R used (Total process time):
      real time          0.01 seconds
      cpu time           0.01 seconds
```

The following code assigns strings to macro variables. In this example, the macro variable `Mymacrovar` is defined using the `submit` callback method. The macro variable `Myvar` was defined using the `symput` callback method. Both macro variables can be accessed outside of the call to `PROC R`.

```
proc r restart;
  submit;
  submit("%let mymacrovar=Hi there;")
  symput('myvar', "This variable has global scope")
  txt <- symget("mymacrovar")
  print(txt)
  endsubmit;
run;

%put &=mymacrovar;
%put &=myvar;
```

Output 48.2 *Output with Macro Variables in PROC R*

```
82  submit("%let mymacrovar=Hi there;")
83  symput('myvar', "This variable has global scope")
84  txt <- symget("mymacrovar")
85  print(txt)
86  endsubmit;
87  run;
88  %let mymacrovar=Hi there;
[1] "Hi there"
NOTE: PROCEDURE R used (Total process time):
      real time          1.01 seconds
      cpu time           0.18 seconds

89
90  %put &=mymacrovar;
MYMACROVAR=Hi there
91  %put &=myvar;
MYVAR=This variable has global scope
```

Using PROC R Callback Methods

Callback Methods That Interact with SAS

`PROC R` provides a module called `SAS` that enables you to communicate between your SAS session and the R subprocess. You can submit SAS code or interact with

SAS variables within an R code block by using the following callback methods. Because R is a case-sensitive language, specify callback methods using the capitalization as it is shown in this list.

rplot(*r-plot-object*, <filename = "plot-name.extension">)

Prints a graph in the Results window.

Examples `rplot(p)`

```
rplot(quote(plot(mtcars$mpg, mtcars$wt,
                main = "MPG vs Weight",
                xlab = "Weight (1000 lbs)",
                ylab = "Miles per Gallon",
                col = "blue",
                pch = 19,
                cex = 1.5)),
      filename = "cars_scatter.png")
```

renderImage(*full-path-to-image-file*)

Renders an image file to the Results window. Valid file types include: 'png', 'jpg', 'jpeg', 'svg', 'gif', 'bmp', 'tiff'.

Example `full_path <- paste0(sas$workpath, "myplot.png")`
`renderImage(full_path)`

sasfnc(*"function-name"* <, *argument-1* <, *argument-2...>>)*

Calls a SAS function (remove the using %sysfunc part). If the function requires arguments, specify those in a comma-delimited list after the function name.

Examples Checks if the WORK libref exists
`sasfnc("exist", "work", "LIBREF")`

Calls the TODAY function in SAS
`sasfnc("today")`

Note: PROC R currently does not support invoking functions created with PROC FCMP.

show(*r-object*, <title="title">, <count=number>)

Displays R objects in the Results window.

Example `show(df, title = "Cars", count = 10)`

submit(*"sas-code"*)

Submits SAS code from within the R session.

Example `submit("%let year=2025;")`

symget(*"sas-macro-variable"*)

Retrieves the value of a SAS macro variable.

Example Retrieves the value of the SYSUSERID macro variable
`symget("SYSUSERID")`

symput("sas-macro-variable", "R-variable")

Sets the value of a SAS macro variable.

Example Sets the value of year to 2025

```
symput("year", "2025")
```

Callback Methods That Transfer Data within the R Procedure

The following callback methods can be used to transfer data between R data frames and SAS data sets.

sd2df("sas_dataset") | sd2df("filename", "library")

Exports a SAS data set to a local R data frame.

Alias sasdata2dataframe

Example Export sashelp.cars to an R data frame

```
df <- sd2df("sashelp.cars")
```

```
df <- sd2df("cars", "sashelp")
```

df2sd(df, "sas_dataset") | df2sd(df, "filename", "library")

Transfers an R data frame to a SAS data set.

Alias dataframe2sasdata

Example Converts the df dataframe into a data set in work.cars (work is the default library)

```
df2sd(df, "work.cars")
```

```
df2sd(df, "cars", "work")
```

Showing Debug Messages for PROC R Functions

PROC R provides an optional debug capability that you can use to display additional diagnostic messages as PROC R functions execute. This is intended to help you observe internal processing and troubleshoot issues when running R code in a SAS session.

Enabling Debug Messages

You can enable debug messages for PROC R functions by setting a debug flag before or during execution. When debug mode is enabled, PROC R functions display additional messages as they progress through their logic.

There are two supported ways to enable debug output:

- By setting an R option within the PROC R session
- By defining an environment variable in the SAS Compute service configuration

Note: When debug mode is enabled, all debug messages are written to the standard PROC R log output. Debug messages are not written to files or stored within the SAS Compute server pod.

Using an R Option

You can enable debug messages for the current PROC R session by setting the R.DEBUG option to TRUE within the submitted R code.

Example Code 48.1 Example

```
%let SYM=1;
proc r; submit;
    options(R.DEBUG = TRUE)
    sym <- symget("SYM")
    endsubmit;
run;
```

Using an Environment Variable

You can also enable debug messages by defining an environment variable for the SAS Compute service. This approach enables debug output at the container level. This configuration is applied by modifying the SAS Compute job PodTemplate in a Kubernetes environment.

```
template.spec.containers.env
- name: RLANG_DEBUG
  value: "TRUE"
```

Troubleshooting: Removing All Variables from the Environment

When you run R code in PROC R, the R session contains several internal variables that PROC R uses to communicate with the SAS Compute session. If you remove these variables, the Compute session can become unresponsive.

The following R code can be used to remove all variables from the environment:

```
rm(list=ls())
```

In PROC R, this command removes the following variables:

- ctor
- obj

- PortVal
- sas

If these variables are deleted, the Compute session can hang. Any of the following solutions can prevent this issue:

- Do not use `rm(list = ls())` (or any equivalent “remove everything” pattern) inside PROC R.
- Never delete `ctor`, `obj`, `portVal`, or `sas`.
- Exclude the PROC R internal variables from the removal list. See the example below.

Example Code 48.2 *Example: Remove all objects except PROC R internal variables*

```
proc r;
submit;
  x <- 10
  y <- 11
  ch <- "abc"

  # Remove all objects except PROC R internal variables
  rm(list = setdiff(ls(), c("ctor", "obj", "portVal", "sas")))
endsubmit;
run;
```

Examples: R Procedure

Example 1: Specify Input from an External R Script

Features: FILENAME statement
 PROC R statement, INFILE= option

Details

This example shows two ways to run an R script called `my_script.R`. The first program specifies the file name and location directly in the `INFILE=` option of the PROC R statement. The second program defines a fileref to the `my_script.R` file and uses that fileref in the `INFILE=` option.

Here are the contents of `my_script.R` :


```
print("Hello World")
y <- "Hi!"
my_function <- function() {
  print("Inside the my_script.R script")
}
```

Program: Specify an R Script Location with the INFILE= Option

Execute the PROC R statement and specify the R script location and name.

Upload your file to a location in SAS Data and AI Studio. Use the INFILE= option to provide the name and location of the R script. SAS executes the R statements in `my_script.R` when you invoke the INFILE= option. Variables and functions that are created by running the script are available inside the SUBMIT and ENDSUBMIT block in PROC R.

```
proc r infile='tmp/my_script.R';
submit;
my_function()
print(paste("y =", y))
endsubmit;
run;
```

Program: Specify an R Script as a Fileref

```
* Filename example for SAS Content;
filename script filesrv folderPath='/Users/myuser/My Folder/tmp/'
filename='my_script.R';

* Filename example for SAS Server;
filename script '/bigdisk/lax/myuser/my_script.R';

proc r infile=script;
submit;
my_function()
print(paste("y =", y))
endsubmit;
run;
```

Program Description

Define a fileref that specifies your R script and its location. Upload your R script into SAS Data and AI Studio. Use the FILENAME statement to create a fileref called `script` for your R script.

```
* Filename example for SAS Content;
filename script filesrv folderPath='/Users/myuser/My Folder/tmp/'
filename='my_script.R';
```

```
* Filename example for SAS Server;
filename script '/bigdisk/lax/myuser/my_script.R';
```

Execute the PROC R statement and specify the fileref that identifies the R script.

The script is invoked when you specify the INFILE= option in the PROC R statement. Variables and functions that are created by running the script are available inside the SUBMIT and ENDSUBMIT block in PROC R.

```
proc r infile=script;
submit;
my_function()
print(paste("y =", y))
endsubmit;
run;
```

Example Code 48.1 SAS Log After Invoking an External R Script

```
80  filename script filesrv folderPath='/Users/myuser/My Folder/tmp/'
filename='my_script.R';
81
82  proc r infile=script;
83  submit
NOTE: Resuming R state from previous PROC R invocation.
83  !      ;
84  my_function()
85  print(paste("y =", y))
86  endsubmit;
87  run;
[1] "Hello World"
[1] "Inside the my_script.R script"
[1] "y = Hi!"
NOTE: PROCEDURE R used (Total process time):
      real time          0.19 seconds
      cpu time           0.05 seconds
```

Example 2: Observe the Impact of Restarting and Terminating the R Procedure

Features: PROC R statement, RESTART option
PROC R statement, TERMINATE option

Details

This example demonstrates the effects of maintaining the status of the R procedure across multiple procedure calls. You can see that the status of the R procedure is reset when you use the RESTART or TERMINATE options.

Program

```

proc r;
submit;
y <- "Hi!"
my_function <- function() {
  print("Inside the function")
}
endsubmit;
run;

proc r;
submit;
my_function()
print(paste("y =", y))
endsubmit;
run;

proc r restart;
submit;
x <- "New variable x is created"
print(paste("x =", x))
#References to function, variable from previous PROC R call
my_function()
print(paste("y =", y))
endsubmit;
run;

proc r terminate;
submit;
#Reference to variable defined in previous PROC R call
print(paste("x =", x))
x <- "Updating the value for x"
print(paste("x =", x))
# Redefining a function called my_function
my_function <- function() {
  print("Inside the proc step")
}
endsubmit;
run;

proc r;
submit;
print(paste("x =", x))
my_function()
endsubmit;
run;

```

Program Description

Call PROC R and define a variable `y` and a function called `my_function`. Assign a string value to the variable `y`. Define a function called `my_function` that prints another string.

```
proc r;
```

```

submit;
y <- "Hi!"
my_function <- function() {
  print("Inside the function")
}
endsubmit;
run;

```

Refer to the previously defined function and variable in a subsequent R procedure call. Call the function `my_function` and print the value of the variable `y`. Because the status of the R procedure is maintained across subsequent procedure calls, the definition of these items is maintained.

```

proc r;
submit;
my_function()
print(paste("y =", y))
endsubmit;
run;

```

Use the RESTART option to reset the R procedure status. The status of the R procedure resets at the beginning of this procedure call. Define a new variable called `x`. Previously defined variables or functions are no longer available. References to these result in an error in the SAS log and the procedure call exits.

```

proc r restart;
submit;
x <- "New variable x is created"
print(paste("x =", x))
#References to function, variable from previous PROC R call
my_function()
print(paste("y =", y))
endsubmit;
run;

```

Use the TERMINATE option to stop maintaining the status of PROC R. When you use the TERMINATE option, the status of the R procedure resets at the end of the procedure call. The variable `x` is still defined from the previous call to PROC R. Specify a new definition for the function `my_function`. The previous definition of this function was lost when you specified the RESTART option in the previous call to PROC R.

```

proc r terminate;
submit;
#Reference to variable defined in previous PROC R call
print(paste("x =", x))
x <- "Updating the value for x"
print(paste("x =", x))
# Redefining a function called my_function
my_function <- function() {
  print("Inside the proc step")
}
endsubmit;
run;

```

Verify that the status of PROC R was reset at the end of the preceding call to PROC R. Because you specified the TERMINATE option in the previous call to PROC R, no variable or function definitions were maintained after that procedure

call ended. References to `x` result in an error in the SAS log and the procedure call exits.

```
proc r;
submit;
print(paste("x =", x))
my_function()
endsubmit;
run;
```

Output 48.3 *Output That Shows the Effect of the RESTART and TERMINATE Options*

```
80  proc r;
81  submit
NOTE: Resuming R state from previous PROC R invocation.
81 !      ;
82  y <- "Hi!"
83  my_function <- function() {
84    print("Inside the function")
85  }
86  endsubmit;
87  run;
NOTE: PROCEDURE R used (Total process time):
      real time          0.00 seconds
      cpu time           0.00 seconds
```

```
89  proc r;
90  submit
NOTE: Resuming R state from previous PROC R invocation.
90 !      ;
91  my_function()
92  print(paste("y =", y))
93  endsubmit;
94  run;
[1] "Inside the function"
[1] "y = Hi!"
NOTE: PROCEDURE R used (Total process time):
      real time          0.04 seconds
      cpu time           0.00 seconds
```

```
98  x <- "New variable x is created"
99  print(paste("x =", x))
100 #References to function, variable from previous PROC R call
101 my_function()
102 print(paste("y =", y))
103 endsubmit;
104 run;
ERROR: Unhandled R exception.
[1] "x = New variable x is created"
ERROR: [R] Error: could not find function "my_function"
NOTE: The SAS System stopped processing this step because of errors.
NOTE: PROCEDURE R used (Total process time):
      real time          0.89 seconds
      cpu time           0.18 seconds
```

```

106 proc r terminate;
107 submit
NOTE: Resuming R state from previous PROC R invocation.
107!      ;
108 #Reference to variable defined in previous PROC R call
109 print(paste("x =", x))
110 x <- "Updating the value for x"
111 print(paste("x =", x))
112 # Redefining a function called my_function
113 my_function <- function() {
114     print("Inside the proc step")
115 }
116 endsubmit;
117 run;
[1] "x = New variable x is created"
[1] "x = Updating the value for x"
NOTE: R terminated.
NOTE: PROCEDURE R used (Total process time):
      real time          0.04 seconds
      cpu time           0.02 seconds

```

```

119 proc r;
120 submit
NOTE: R initialized.
WARNING: ignoring environment value of R_HOME
120!      ;
121 print(paste("x =", x))
122 my_function()
123 endsubmit;
124 run;
ERROR: Unhandled R exception.
ERROR: [R] Error: object 'x' not found
NOTE: The SAS System stopped processing this step because of errors.
NOTE: PROCEDURE R used (Total process time):
      real time          0.92 seconds
      cpu time           0.17 seconds

```

Example 3: Submit SAS Code and Call a SAS Function within PROC R

Features: submit callback method
 sasfnc callback method

Details

This example demonstrates how the submit callback method can be used to submit SAS code within a block of R code. In this example, use the submit callback method to create a data set called Test. The Test data set is available outside of the call to PROC R. This example also shows how you use the sasfnc callback method to call

the SAS function SHA256HEX to display the SHA256 digest obtained for the value 'abc'.

Program

```
proc r;
submit;
var1 <- "'r'"
var2 <- 2

submit(
  paste(
    "data work.test; x=", var1, "; y=", var2, "; run;",
    sep = ""
  )
)

var3 <- sasfnc("sha256hex","abc")
print(paste("var3 = ", var3))

endsubmit;
run;

proc print data=test;
run;
```

Program Description

Begin your call to PROC R and use the SUBMIT statement to start a block of R code. Assign values to two R variables, `var1` and `var2`. The string value takes an additional set of quotation marks so that the value 'r' is sent to the DATA step.

```
proc r;
submit;
var1 <- "'r'"
var2 <- 2
```

Call the submit callback method and create a SAS data set Work.Test. Assign values to X and Y by using the R format function.

```
submit(
  paste(
    "data work.test; x=", var1, "; y=", var2, "; run;",
    sep = ""
  )
)
```

Call a SAS function using the SAS.sasfnc callback method. Call the [SHA256HEX function](#) for the string "abc" and assign the return value to the R variable `var3`. Call the R print function to display a string that shows the value of `var3`.

```
var3 <- sasfnc("sha256hex","abc")
print(paste("var3 = ", var3))
```

End the block of R code, and then print the generated data set `Work.Test`. Call the `ENDSUBMIT` statement and end the R code block. Follow the call to `PROC R` with a `PROC PRINT` call that refers to the `Work.Test` data set. Note that the data set is available outside of the `PROC R` call.

```
endsubmit;
run;

proc print data=test;
run;
```

Output 48.4 *Output from Submit and sasfnc Calls in PROC R*

Obs	x	y
1	r	2

Example Code 48.2 *Log from Submit and sasfnc Calls in PROC R*

```
NOTE: The data set WORK.TEST has 1 observations and 2 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds

[1] "var3 = BA7816BF8F01CFEA414140DE5DAE2223B00361A396177A9CB410FF61F20015AD"
NOTE: PROCEDURE R used (Total process time):
      real time           0.25 seconds
      cpu time            0.16 seconds

96
97  proc print data=test;
98  run;
NOTE: There were 1 observations read from the data set WORK.TEST.
NOTE: The PROCEDURE PRINT printed page 1.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.06 seconds
      cpu time            0.05 seconds
```

Example 4: Create a SAS Macro Variable in PROC R

Features:

- Defining SAS macro variables
- symget callback method
- symput callback method

Details

This example shows how you can create a SAS macro variable and assign the value of an R variable to it. The macro variable has global scope and can be referenced outside of the PROC R call.

Program

```
%let myvar = Jane Doe;

proc r;
submit;
x <- symget('myvar')
print(paste('macro variable myvar = ', x))

r_var <- 'Inside r'
symput('macrovar', r_var)
endsubmit;
run;

%put &macrovar;
%put &myvar;
```

Program Description

Assign a macro variable, Myvar, outside of a call to PROC R.

```
%let myvar = Jane Doe;
```

Call PROC R and assign the value of Myvar to an R variable. Assign the value of the SAS macro variable Myvar to the R variable x by using the symget callback method. Use the R print function to display the value of x.

```
proc r;
submit;
x <- symget('myvar')
print(paste('macro variable myvar = ', x))
```

Define an R variable and assign a string value. Define the R variable r_var and assign a string value to it. Then call symput to replace the value of r_var with the value of the Macrovar variable.

```
r_var <- 'Inside r'
symput('macrovar', r_var)
endsubmit;
run;
```

Refer to the global macro variable Macrovar outside of PROC R. Macro variables that you define with symput are global variables. Therefore, you can refer to them outside of the call to PROC R. The macro variable Myvar is also still available.

```
%put &=macrovar;
%put &=myvar;
```

Example Code 48.3 SAS Log from the Macro Variable Example

```
84 x <- symget('myvar')
85 print(paste('macro variable myvar = ', x))
86
87 r_var <- 'Inside r'
88 symput('macrovar', r_var)
89 endsubmit;
90 run;
[1] "macro variable myvar = Jane Doe"
NOTE: PROCEDURE R used (Total process time):
      real time          0.15 seconds
      cpu time           0.05 seconds

91
92 %put &=macrovar;
MACROVAR=Inside r
93 %put &=myvar;
MYVAR=Jane Doe
```

Example 5: Render R Plots

Features: rplot callback method
 renderImage callback method

Details

The following example programs demonstrate how to use `rplot()` to print graphs created with Base R and `ggplot2`, and how to render an existing image in the Results window with `renderImage()`.

Program: Render an R Plot with Base R

```
proc r;
submit;

x <- 1:100
y <- x^2 + rnorm(100, 0, 100)

rplot({
plot(x, y,
```

```

        main = "Base R Scatter Plot",
        xlab = "X Values",
        ylab = "Y Values",
        col = "blue",
        pch = 19)

    abline(lm(y ~ x), col = "red", lwd = 2)
  })

endsubmit;
run;

```

Program Description

Call PROC R. The SUBMIT statement identifies the beginning of a block of R code.

```

proc r;
submit;

```

Generate data for plotting. Create input values for the plot. Define an x sequence and compute corresponding y values from a simple formula plus random variation.

```

x <- 1:100
y <- x^2 + rnorm(100, 0, 100)

```

Render the plot to SAS output. Wrap the plotting commands in `rplot()` so that the graphic is captured and displayed in SAS results output.

```

rplot({

```

Specify the plot contents and basic appearance. Draw the scatter plot from the x and y vectors. Set a title, axis labels, and a few display options (point color and marker type).

```

plot(x, y,
     main = "Base R Scatter Plot",
     xlab = "X Values",
     ylab = "Y Values",
     col = "blue",
     pch = 19)

```

Add a fitted line to the plot. Fit a simple model and overlay the resulting line on the existing plot. Use styling options to make the line visually distinct.

```

    abline(lm(y ~ x), col = "red", lwd = 2)
  })

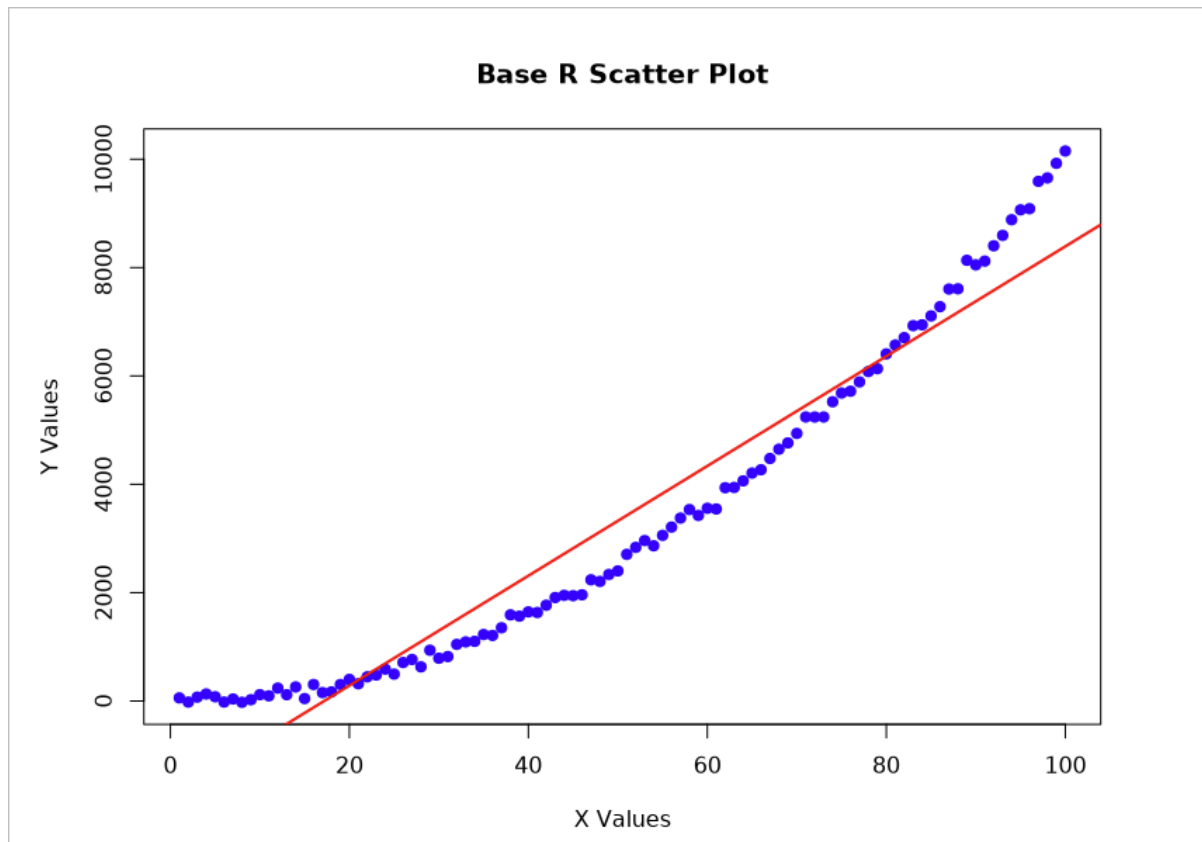
```

End the R block and finish PROC R. Close the submitted R code with ENDSUBMIT. Execute the procedure with RUN to complete the step and finalize output.

```

endsubmit;
run;

```

Output 48.5 Output from the Rplot Callback Method

Program: Render an R Plot with ggplot2

```
proc r;
submit;
library(ggplot2)

df <- data.frame(
  category = rep(c("A", "B", "C", "D"), each = 25),
  value = c(rnorm(25, 10, 2),
            rnorm(25, 15, 3),
            rnorm(25, 12, 2.5),
            rnorm(25, 18, 4))
)

p <- ggplot(df, aes(x = category, y = value, fill = category)) +
  geom_boxplot(alpha = 0.7) +
  theme_minimal() +
  labs(title = "Distribution by Category",
       x = "Category",
       y = "Value") +
  scale_fill_brewer(palette = "Set2")

rplot(p, filename = "boxplot_demo.png")

endsubmit;
run;
```

Program Description

Call PROC R and load required R packages. Start PROC R and open a SUBMIT block so that the following statements are executed as R code. Load ggplot2 so you can create plots using the grammar-of-graphics approach.

```
proc r;
submit;
library(ggplot2)
```

Generate sample data for the plot. Create a data frame with a category column and a numeric value column. The values are generated with `rnorm()` to produce example distributions that differ by category.

```
df <- data.frame(
  category = rep(c("A", "B", "C", "D"), each = 25),
  value = c(rnorm(25, 10, 2),
            rnorm(25, 15, 3),
            rnorm(25, 12, 2.5),
            rnorm(25, 18, 4))
)
```

Create the ggplot object and define its appearance. Build a plot object that maps category to the X axis and value to the Y axis, and uses fill to color groups. Add a box plot layer and apply common presentation options such as a theme, labels, and a color palette.

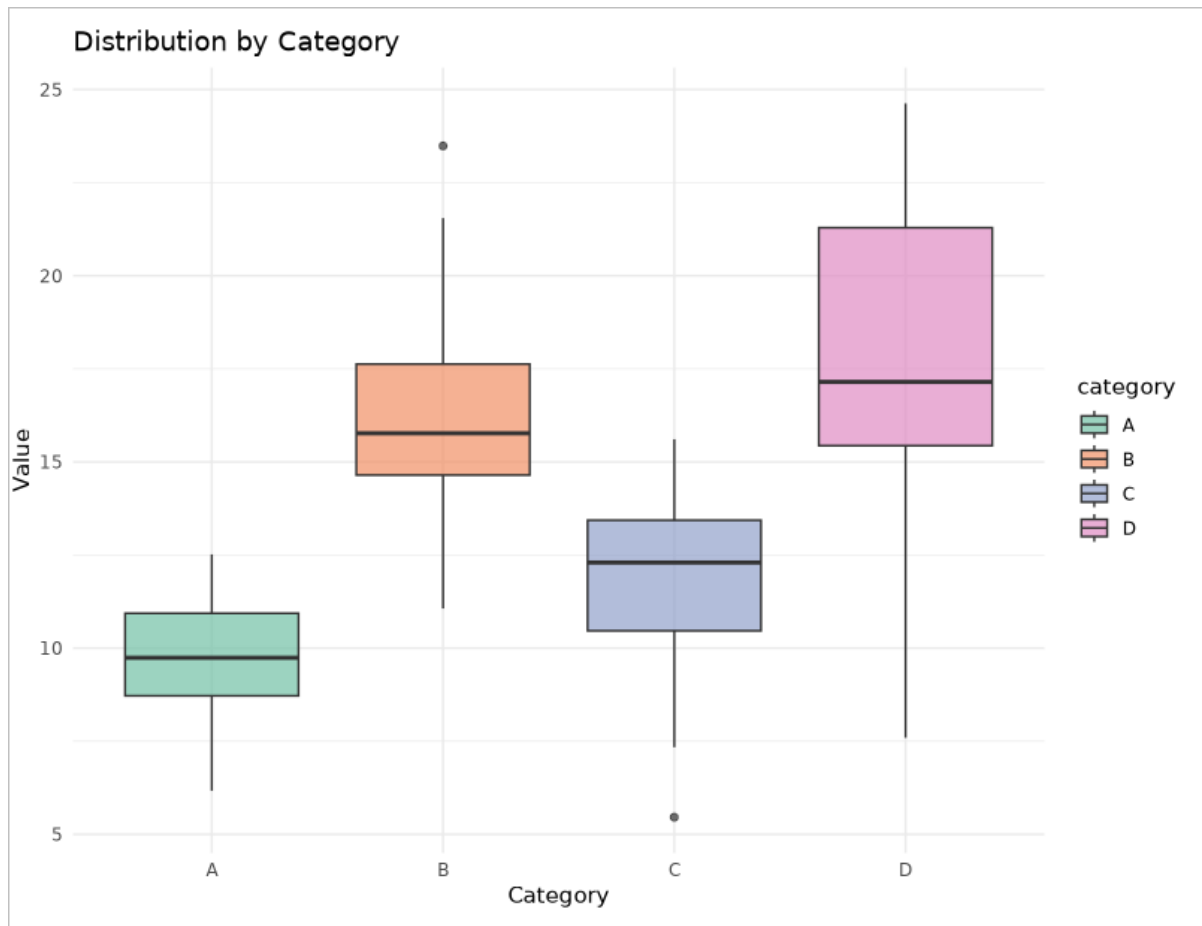
```
p <- ggplot(df, aes(x = category, y = value, fill = category)) +
  geom_boxplot(alpha = 0.7) +
  theme_minimal() +
  labs(title = "Distribution by Category",
       x = "Category",
       y = "Value") +
  scale_fill_brewer(palette = "Set2")
```

Render the plot in SAS output and write it to a file. Render the ggplot object so that it appears in SAS results output. Save the rendered image to a PNG file by specifying `filename=`.

```
rplot(p, filename = "boxplot_demo.png")
```

End the R block and finish PROC R. Close the submitted R code with ENDSUBMIT. Execute the procedure with RUN to complete the step and finalize output.

```
endsubmit;
run;
```

Output 48.6 Output from the Rplot Callback Method

Program: Render Multiple Plots in Sequence

```
proc r;
submit;

library(ggplot2)

cars_df <- sd2df("sashelp.cars")

p1 <- ggplot(cars_df, aes(x = Weight, y = MPG_City)) +
  geom_point(aes(color = Type), alpha = 0.6, size = 3) +
  geom_smooth(method = "lm", se = TRUE, color = "red") +
  theme_minimal() +
  labs(title = "Fuel Efficiency vs Weight",
       x = "Weight (lbs)",
       y = "City MPG")

rplot(p1, filename = "scatter_mpg.png")

type_summary <- aggregate(MSRP ~ Type, data = cars_df, FUN = mean)

p2 <- ggplot(type_summary, aes(x = reorder(Type, MSRP), y = MSRP)) +
  geom_col(fill = "steelblue", alpha = 0.8) +
  coord_flip() +
  theme_minimal() +
```

```

labs(title = "Average MSRP by Vehicle Type",
      x = "Vehicle Type",
      y = "Average MSRP ($)")

rplot(p2, filename = "bar_msrp.png")

p3 <- ggplot(cars_df, aes(x = Horsepower)) +
  geom_histogram(binwidth = 20, fill = "darkgreen",
                 alpha = 0.7, color = "black") +
  theme_minimal() +
  labs(title = "Distribution of Horsepower",
       x = "Horsepower",
       y = "Count")

rplot(p3, filename = "hist_hp.png")

endsubmit;
run;

```

Program Description

Call PROC R and begin an R submission block. Start PROC R and open a SUBMIT block so that the following statements are executed as R code.

```

proc r;
submit;

```

Load required R packages. Load ggplot2 so that you can create plots using its functions and layering syntax.

```
library(ggplot2)
```

Read a SAS data set into an R data frame. Import the SAS data set `sashelp.cars` into R as a data frame named `cars_df`. This provides the source data used by all three plots.

```
cars_df <- sd2df("sashelp.cars")
```

Specify Plot 1 (scatter plot with a trend line). Create a scatter plot that compares city fuel efficiency to vehicle weight, with points colored by vehicle type. Add a fitted line to summarize the overall relationship and apply labels and a simple theme.

```

p1 <- ggplot(cars_df, aes(x = Weight, y = MPG_City)) +
  geom_point(aes(color = Type), alpha = 0.6, size = 3) +
  geom_smooth(method = "lm", se = TRUE, color = "red") +
  theme_minimal() +
  labs(title = "Fuel Efficiency vs Weight",
       x = "Weight (lbs)",
       y = "City MPG")

```

Render Plot 1 in SAS output. Render the ggplot object so that it appears in SAS results output. Save the image to a PNG file by specifying `filename=`.

```
rplot(p1, filename = "scatter_mpg.png")
```

Create summary data for Plot 2 (average MSRP by type). Compute a grouped summary that calculates the mean MSRP for each vehicle type.

```
type_summary <- aggregate(MSRP ~ Type, data = cars_df, FUN = mean)
```

Specify Plot 2 (bar chart). Create a bar chart showing the average MSRP for each vehicle type, ordered by value. Flip the coordinates so that category labels are easier to read and apply basic labels and styling.

```
p2 <- ggplot(type_summary, aes(x = reorder(Type, MSRP), y = MSRP)) +
  geom_col(fill = "steelblue", alpha = 0.8) +
  coord_flip() +
  theme_minimal() +
  labs(title = "Average MSRP by Vehicle Type",
       x = "Vehicle Type",
       y = "Average MSRP ($)")
```

Render Plot 2 in SAS output. Render the second ggplot object to SAS results output. Save the bar chart to a PNG file using the specified filename.

```
rplot(p2, filename = "bar_msrp.png")
```

Specify Plot 3 (histogram). Create a histogram to show the distribution of horsepower values in the data. Control the bin width and add labels and a theme.

```
p3 <- ggplot(cars_df, aes(x = Horsepower)) +
  geom_histogram(binwidth = 20, fill = "darkgreen",
                alpha = 0.7, color = "black") +
  theme_minimal() +
  labs(title = "Distribution of Horsepower",
       x = "Horsepower",
       y = "Count")
```

Render Plot 3 in SAS output. Render the histogram to SAS results output. Save the rendered image to a PNG file by providing filename=.

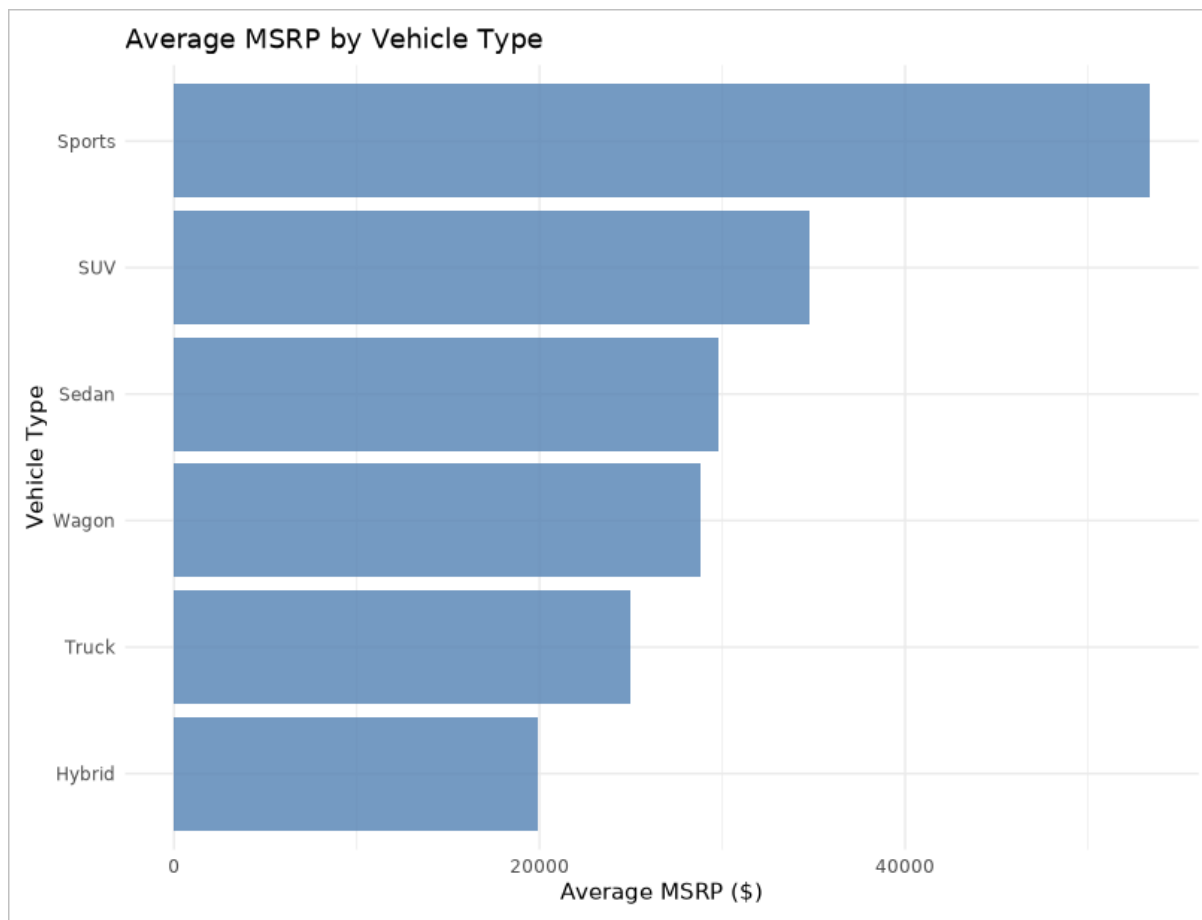
```
rplot(p3, filename = "hist_hp.png")
```

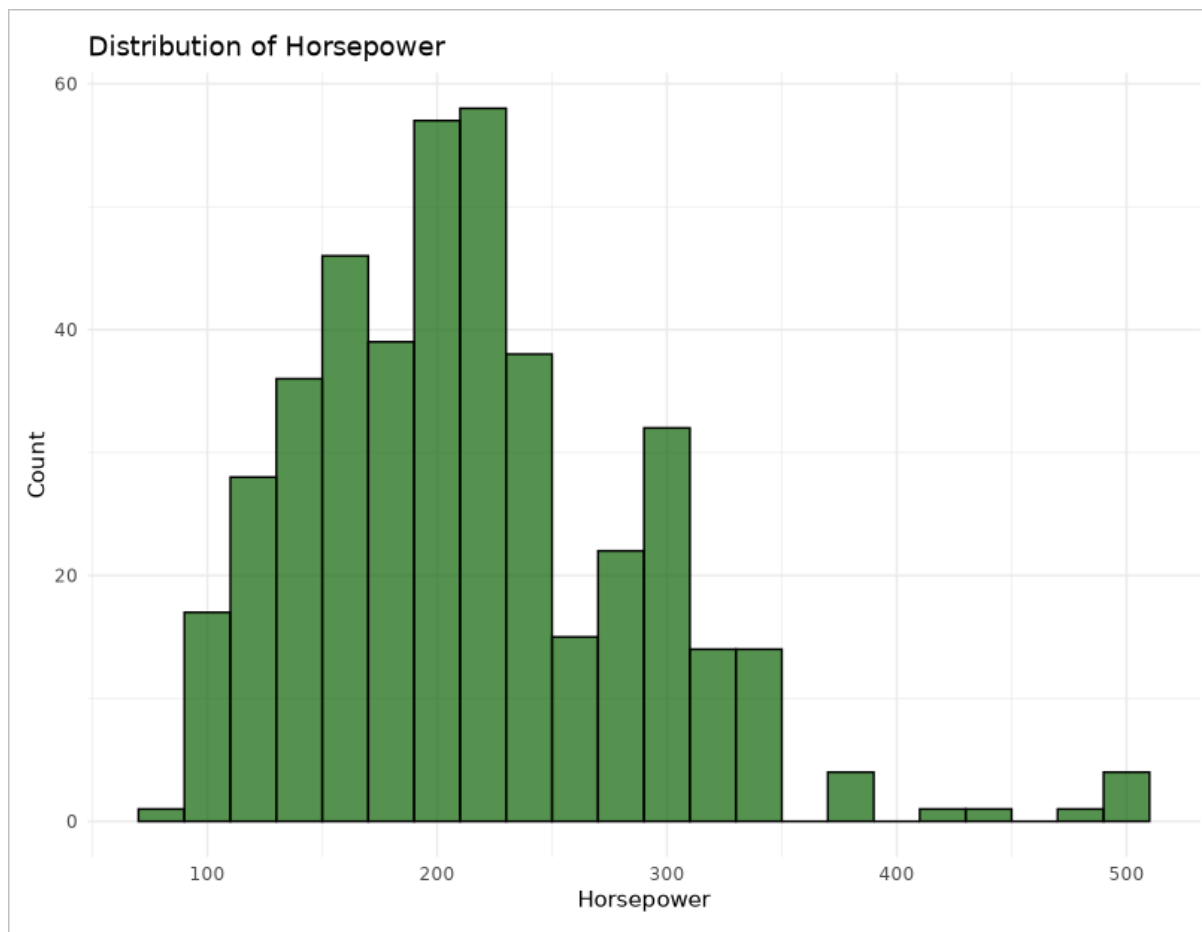
End the R block and finish PROC R. Close the submitted R code with ENDSUBMIT. Execute the procedure with RUN to complete processing and finalize the output.

```
endsubmit;
run;
```


Output 48.7 Output Scatter Plot from the Rplot Callback Method



Output 48.8 Output Bar Chart from the Rplot Callback Method

Output 48.9 Output Histogram from the Rplot Callback Method

Program: Display an Existing Image

```
proc r;
submit;

library(ggplot2)

data <- data.frame(
  x = seq(0, 10, 0.1),
  y = sin(seq(0, 10, 0.1))
)

p <- ggplot(data, aes(x = x, y = y)) +
  geom_line(color = "purple", size = 1.5) +
  geom_point(color = "orange", size = 2) +
  theme_classic() +
  labs(title = "Sine Wave Visualization",
       x = "X", y = "sin(X)")

plot_path <- file.path(sas$workpath, "sine_wave.png")
ggsave(plot_path, p, width = 8, height = 5, dpi = 150)

renderImage(plot_path)

endsubmit;
```

```
run;
```

Program Description

Call PROC R and begin an R submission block. Start PROC R and open a SUBMIT block so that the following statements are executed as R code within SAS.

```
proc r;
submit;
```

Load the plotting package. Load ggplot2 so that you can build a plot using its functions and layering syntax.

```
library(ggplot2)
```

Generate data for the plot. Create a small data frame with an x sequence and corresponding y values computed with `sin()`.

```
data <- data.frame(
  x = seq(0, 10, 0.1),
  y = sin(seq(0, 10, 0.1))
)
```

Create the ggplot object and set its appearance. Build a plot object that draws the sine wave using a line layer and highlights points along the curve. Apply a theme and labels.

```
p <- ggplot(data, aes(x = x, y = y)) +
  geom_line(color = "purple", size = 1.5) +
  geom_point(color = "orange", size = 2) +
  theme_classic() +
  labs(title = "Sine Wave Visualization",
       x = "X", y = "sin(X)")
```

Save the plot to an image file (without displaying it yet). Construct a file path in the SAS Work directory and use it as the output location for the image. Save the ggplot object to a PNG file with `ggsave()` so that it can be rendered later.

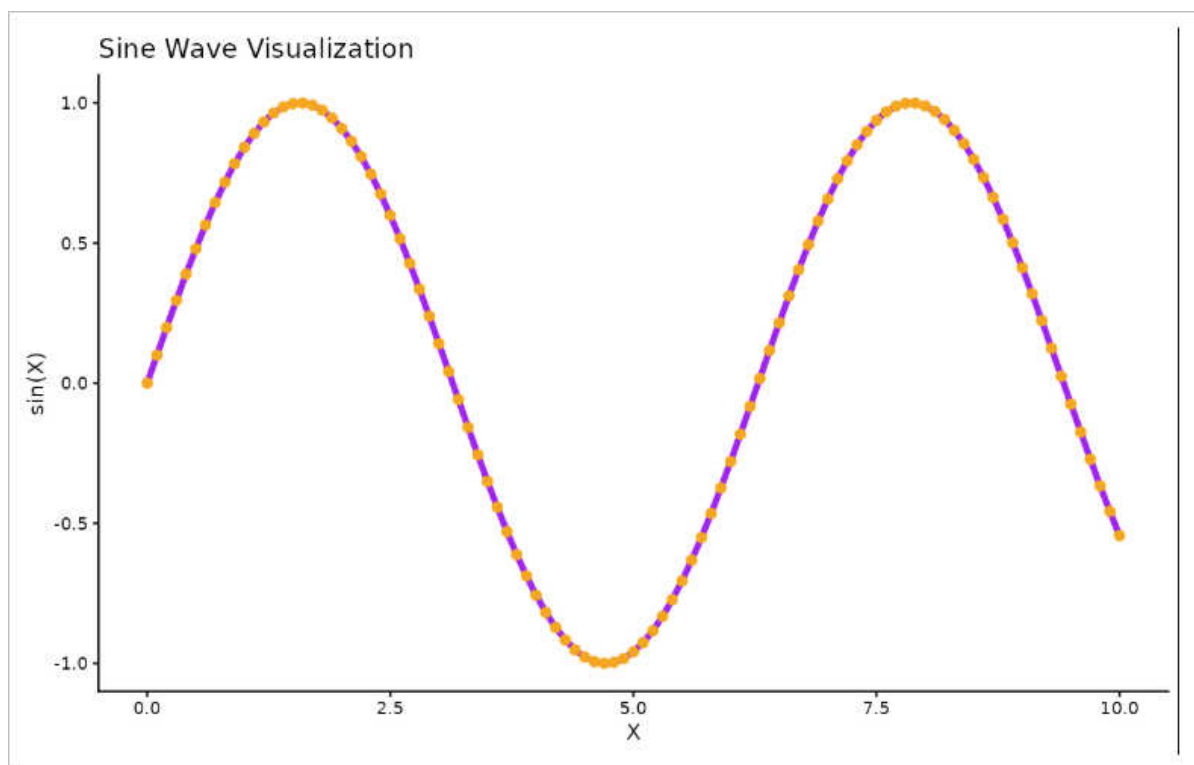
```
plot_path <- file.path(sas$workpath, "sine_wave.png")
ggsave(plot_path, p, width = 8, height = 5, dpi = 150)
```

Render the saved image in the Results window. Render the image file referenced by `plot_path` so that it appears in the Results window.

```
renderImage(plot_path)
```

End the R block and finish PROC R. Close the submitted R code with `ENDSUBMIT`. Execute the procedure with `RUN` to complete processing and finalize the output.

```
endsubmit;
run;
```

Output 48.10 *Output Image from the RenderImage Callback Method*

Example 6: Display Data Frames

Features:

- show callback method
- sd2df callback method
- df2sd callback method

Details

The following example programs demonstrate the sd2df, df2sd, and show functions to transfer between SAS Data Sets and R Data Frames and view output in a SAS Results window.

Program: Create a SAS Data Set and Import It into R as a Data Frame

```
data work.mydata;  
  length a 8;
```

```

length b_value 8;
length var_32 $32;
var_32 = "x";
a = 1;
b_value = 2;
output;
run;

proc r;
submit;

df <- sd2df("work.mydata");

show(df)

endsubmit;
run;

```

Program Description

Create a SAS data set to use as input. This DATA step builds `work.mydata` with three variables (`a`, `b_value`, and character `var_32`) and writes a single observation to the WORK library.

```

data work.mydata;
length a 8;
length b_value 8;
length var_32 $32;
var_32 = "x";
a = 1;
b_value = 2;
output;
run;

```

Start PROC R and open an R submission block. Start the R procedure and begin a SUBMIT block so that the following statements are interpreted and executed as R code within the SAS session.

```

proc r;
submit;

```

Read the SAS data set into an R data frame. Import the SAS data set `work.mydata` into R as a data frame named `df`.

```

df <- sd2df("work.mydata");

```

Display the R data frame in the SAS Results window. Render the R data frame output back to SAS so that you can view it in the Results window.

```

show(df)

```

End the R block and finish PROC R. Close the R submission block and execute PROC R to finalize processing and produce the displayed output.

```

endsubmit;
run;

```

Output 48.11 Output Data Frame from the Show Callback Method

Output			
Obs	a	b_value	var_32
1	1	2	x

Program: Import a SAS Data Set into R and Display Selected Rows and Summary Table

```

proc r;
submit;

cars_df <- sd2df("sashelp.cars")

show(cars_df, title = "SASHELP.CARS (First 10 Rows)", count = 10)

summary_stats <- data.frame(
  Statistic = c("Mean MPG", "Max Horsepower", "Min Weight",
               "Avg MSRP", "Total Cars"),
  Value = c(
    mean(cars_df$MPG_City, na.rm = TRUE),
    max(cars_df$Horsepower, na.rm = TRUE),
    min(cars_df$Weight, na.rm = TRUE),
    mean(cars_df$MSRP, na.rm = TRUE),
    nrow(cars_df)
  )
)

show(summary_stats, title = "Cars Dataset Summary Statistics")

endsubmit;
run;

```

Program Description

Call PROC R and begin an R submission block. Start PROC R and open a SUBMIT block so that the following statements are executed as R code within SAS.

```

proc r;
submit;

```

Read a SAS data set into an R data frame. Import the sashelp.cars data set into R as a data frame named cars_df.

```

cars_df <- sd2df("sashelp.cars")

```

Display a subset of the data frame in the Results window. Use show() to render a data frame in the SAS Results window. The title= and count= options provide a label and limit the output to the first rows.

```

show(cars_df, title = "SASHELP.CARS (First 10 Rows)", count = 10)

```

Create a summary table of key statistics. Compute basic measures from the imported data and store them in a simple two-column data frame.

```
summary_stats <- data.frame(
  Statistic = c("Mean MPG", "Max Horsepower", "Min Weight",
               "Avg MSRP", "Total Cars"),
  Value = c(
    mean(cars_df$MPG_City, na.rm = TRUE),
    max(cars_df$Horsepower, na.rm = TRUE),
    min(cars_df$Weight, na.rm = TRUE),
    mean(cars_df$MSRP, na.rm = TRUE),
    nrow(cars_df)
  )
)
```

Display the summary table in the Results window. Use `show()` again to render the summary data frame in the Results window.

```
show(summary_stats, title = "Cars Dataset Summary Statistics")
```

End the R block and finish PROC R. Close the submitted R code with `ENDSUBMIT`. Execute the procedure with `RUN` to complete processing and finalize the output.

```
endsubmit;
run;
```

Output 48.12 Output Data Frame from the First Show Callback Method

SASHELP.CARS (First 10 Rows)															
Obs	Make	Model	Type	Origin	DriveTrain	MSRP	Invoice	EngineSize	Cylinders	Horsepower	MPG_City	MPG_Highway	Weight	Wheelbase	Length
1	Acura	MDX	SUV	Asia	All	36945	33337	3.5	6	265	17	23	4451	106	189
2	Acura	RSX Type S 2dr	Sedan	Asia	Front	23820	21761	2.0	4	200	24	31	2778	101	172
3	Acura	TSX 4dr	Sedan	Asia	Front	26990	24647	2.4	4	200	22	29	3230	105	183
4	Acura	TL 4dr	Sedan	Asia	Front	33195	30299	3.2	6	270	20	28	3575	108	186
5	Acura	3.5 RL 4dr	Sedan	Asia	Front	43755	39014	3.5	6	225	18	24	3880	115	197
6	Acura	3.5 RL w/Navigation 4dr	Sedan	Asia	Front	46100	41100	3.5	6	225	18	24	3893	115	197
7	Acura	NSX coupe 2dr manual S	Sports	Asia	Rear	89765	79978	3.2	6	290	17	24	3153	100	174
8	Audi	A4 1.8T 4dr	Sedan	Europe	Front	25940	23508	1.8	4	170	22	31	3252	104	179
9	Audi	A41.8T convertible 2dr	Sedan	Europe	Front	35940	32506	1.8	4	170	23	30	3638	105	180
10	Audi	A4 3.0 4dr	Sedan	Europe	Front	31840	28846	3.0	6	220	20	28	3462	104	179

Output 48.13 Output Data Frame from the Second Show Callback Method

Cars Dataset Summary Statistics		
Obs	Statistic	Value
1	Mean MPG	20.06
2	Max Horsepower	500.00
3	Min Weight	1850.00
4	Avg MSRP	32774.86
5	Total Cars	428.00

Program: Create an R Data Frame and Transfer It to a SAS Data Set

```
proc r;
submit;

df <- data.frame(
  id      = 1:10,
  value   = c(1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7, 8.8, 9.9, 10.1),
  charcol = c("a","b","c","a","b","c","a","b","c","a")
)

df2sd(df, "work.mydata")

endsubmit;
run;

proc print data = mydata;
run;
```

Program Description

Start PROC R and open an R submission block. Start the R procedure and begin a SUBMIT block so that the following statements execute as R code within SAS.

```
proc r;
submit;
```

Create an R data frame in memory. Construct an R data frame named `df` with three columns: an integer ID sequence (1:10), a numeric column (`value`), and a character column (`charcol`).

```
df <- data.frame(
  id      = 1:10,
  value   = c(1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7, 8.8, 9.9, 10.1),
  charcol = c("a","b","c","a","b","c","a","b","c","a")
)
```

Transfer the R data frame to a SAS data set. Convert the R data frame `df` into a SAS data set named `work.mydata`, making it available to subsequent SAS steps.

```
df2sd(df, "work.mydata")
```

End the R block and finish PROC R. Close the submitted R code and execute PROC R to complete the transfer back into SAS.

```
endsubmit;
run;
```

View the newly created SAS data set. Print the contents of `mydata` (from the WORK library by default) so that you can confirm the rows and column values produced from the R data frame.

```
proc print data = mydata;
run;
```

Output 48.14 Log Output from PRINT Procedure

```
NOTE: There were 10 observations read from the data set WORK.MYDATA.  
NOTE: The PROCEDURE PRINT printed page 1.  
NOTE: PROCEDURE PRINT used (Total process time):  
      real time          0.13 seconds  
      cpu time           0.10 seconds
```

Output 48.15 Output SAS Data Set from the PRINT Procedure

Obs	id	value	charcol
1	1	1.1	a
2	2	2.2	b
3	3	3.3	c
4	4	4.4	a
5	5	5.5	b
6	6	6.6	c
7	7	7.7	a
8	8	8.8	b
9	9	9.9	c
10	10	10.1	a

RANK Procedure

Overview: RANK Procedure	1701
What Does the RANK Procedure Do?	1702
Ranking Data	1702
Concepts: RANK Procedure	1703
Computer Resources	1703
Statistical Applications	1704
Treatment of Tied Values	1704
In-Database Processing for PROC RANK	1706
Syntax: RANK Procedure	1708
PROC RANK Statement	1709
ATTRIB Statement	1713
BY Statement	1714
FORMAT Statement	1715
LABEL Statement	1716
WHERE Statement	1718
RANKS Statement	1720
VAR Statement	1721
Results: RANK Procedure	1722
Missing Values	1722
Output Data Set	1722
Numeric Precision	1722
Examples: RANK Procedure	1722
Example 1: Ranking Values of Multiple Variables	1722
Example 2: Ranking Values within BY Groups	1725
Example 3: Partitioning Observations into Groups Based on Ranks	1727
References	1731

Overview: RANK Procedure

What Does the RANK Procedure Do?

The RANK procedure computes ranks for one or more numeric variables across the observations of a SAS data set and writes the ranks to a new SAS data set. PROC RANK by itself produces no printed output.

Ranking Data

The following output shows the results of ranking the values of one variable with a simple PROC RANK step. In this example, the new ranking variable shows the order of finish of five golfers over a four-day competition. The player with the lowest number of strokes finishes in first place. The following statements produce the output:

```
proc rank data=golf out=rankings;
  var strokes;
  ranks Finish;
run;

proc print data=rankings;
run;
```

Output 49.1 Assignment of the Lowest Rank Value to the Lowest Variable Value

The SAS System				1
Obs	Player	Strokes	Finish	
1	Jack	279	2	
2	Jerry	283	3	
3	Mike	274	1	
4	Randy	296	4	
5	Tito	302	5	

In the following output, the candidates for city council are ranked by district according to the number of votes that they received in the election. They are also ranked according to the number of years that they have served in office.

This example shows how PROC RANK can do the following tasks:

- reverse the order of the rankings so that the highest value receives the rank of 1, the next highest value receives the rank of 2, and so on
- rank the observations separately by values of multiple variables
- rank the observations within BY groups
- handle tied values

For an explanation of the program that produces this report, see [“Example 2: Ranking Values within BY Groups” on page 1725](#).

Output 49.2 *Assignment of the Lowest Rank Value to the Highest Variable Value within Each BY Group*

Results of City Council Election						1
----- District=1 -----						
Obs	Candidate	Vote	Years	Vote Rank	Years Rank	
1	Cardella	1689	8	1	1	
2	Latham	1005	2	3	2	
3	Smith	1406	0	2	3	
4	Walker	846	0	4	3	
N = 4						
----- District=2 -----						
Obs	Candidate	Vote	Years	Vote Rank	Years Rank	
5	Hinkley	912	0	3	3	
6	Kreitemeyer	1198	0	2	3	
7	Lundell	2447	6	1	1	
8	Thrash	912	2	3	2	
N = 4						

Concepts: RANK Procedure

Computer Resources

For any variable that is being ranked, PROC RANK stores in memory the value of that variable for every observation.

Statistical Applications

Ranks are useful for investigating the distribution of values for a variable. The ranks divided by n or $n+1$ form values in the range 0 to 1, and these values estimate the cumulative distribution function. You can apply inverse cumulative distribution functions to these fractional ranks to obtain probability quantile scores. You can compare these scores to the original values to judge the fit to the distribution. For example, if a set of data has a normal distribution, the normal scores should be a linear function of the original values, and a plot of scores versus original values should be a straight line.

Many nonparametric methods are based on analyzing ranks of a variable:

- A two-sample t test applied to the ranks is equivalent to a Wilcoxon rank sum test using the t approximation for the significance level. If you apply the t test to the normal scores rather than to the ranks, the test is equivalent to the van der Waerden test. If you apply the t test to median scores (GROUPS=2), the test is equivalent to the median test.
- A one-way analysis of variance applied to ranks is equivalent to the Kruskal-Wallis k -sample test; the F test generated by the parametric procedure applied to the ranks is often better than the X^2 approximation used by Kruskal-Wallis. This test can be extended to other rank scores (Quade 1966).
- You can obtain a Friedman's two-way analysis for block designs by ranking within BY groups and then performing a main-effects analysis of variance on these ranks (Conover 1998).
- You can investigate regression relationships by using rank transformations with a method described by Iman and Conover (1979).

Treatment of Tied Values

When PROC RANK ranks values, if two or more values of an analysis variable that are within a BY group are equal, then tied values are present in the data. Because the values are indistinguishable and there is usually no further obvious information about which the ranks can reasonably be based, PROC RANK does not assign different ranks to the values. Tied values could be arbitrarily assigned different ranks. But in statistical applications such as nonparametric statistical tests using ranks, it is conventional to assign the same rank to tied values.

These statistical tests commonly assume that the data is from a continuous distribution, in which the probability of a tie is theoretically zero. In practice, whether because of inaccuracies in measurement, the finite accuracy of representation within a digital computer, or other reasons, tied values often occur. It is also conventional in these statistical tests to assign the average rank to a group of tied values. Assignment of the average rank is preferred because it

preserves the sum of the ranks and therefore does not distort the estimate of the cumulative distribution function.

For applications within and outside of statistics, the RANK procedure provides the `TIES=` option to control the treatment of tied values. The default value for this option depends on the specified ranking or scoring method, which you can specify with the options of the PROC RANK statement. For ranking and scoring methods, when `TIES=LOW`, `TIES=HIGH`, or `TIES=MEAN`, tied values are initially treated as if they are distinguishable. These methods all begin by sorting the values of the analysis variable within a BY group, and then assigning to each nonmissing value an ordinal number that indicates its position in the sequence.

Subsequently, for non-scoring methods, PROC RANK resolves tied values by selecting the minimum with `TIES=LOW`, selecting the maximum with `TIES=HIGH`, or calculating the average of the ordinals in a group of tied values with `TIES=MEAN`. PROC RANK then obtains the rank from this value through one or more further transformations such as scaling, translation, and truncation.

Scoring methods include normal and Savage scoring, which are requested by the `NORMAL=` and `SAVAGE` options. Non-scoring methods include ordinal ranking, the default, and those methods that are requested by the `FRACTION`, `NPLUS1`, `GROUPS=`, and `PERCENT` options. For the scoring methods `NORMAL=` and `SAVAGE`, PROC RANK obtains the probability quantile scores with the appropriate formulas as if no tied values were present within the data. PROC RANK then resolves tied values by selecting the minimum, selecting the maximum, or calculating the average of all scores within a tied group.

For all ranking and scoring methods, when `TIES=DENSE`, tied values are treated as indistinguishable, and each value within a tied group is assigned the same ordinal. As with the other `TIES=` resolution methods, all ranking and scoring methods begin by sorting the values of the analysis variable and then assigning ordinals. However, a group of tied values is treated as a single value. The ordinal assigned to the group differs by only +1 from the ordinal that is assigned to the value just prior to the group, if there is one. The ordinal differs by only -1 from the ordinal assigned to the value just after the group, if there is one. Therefore, the smallest ordinal within a BY group is 1, and the largest ordinal is the number of unique, nonmissing values in the BY group.

After the ordinals are assigned, PROC RANK calculates ranks and scores using the number of unique, nonmissing values instead of the number of nonmissing values for scaling. Because of its tendency to distort the cumulative distribution function estimate, dense ranking is not generally acceptable for use in nonparametric statistical tests.

Note that PROC RANK bases its computations on the internal numeric values of the analysis variables. The procedure does not format or round these values before analysis. When values differ in their internal representation, even slightly, PROC RANK does not treat them as tied values. If this is a concern for your data, then round the analysis variables by an appropriate amount before invoking PROC RANK. For information about the `ROUND` function, see [“ROUND Function” in SAS Functions and CALL Routines: Reference](#).

In-Database Processing for PROC RANK

In-database processing has several advantages over processing within SAS. These advantages include increased security, reduced network traffic, and the potential for faster processing. Increased security is possible because sensitive data does not have to be extracted from the DBMS.

Faster processing is possible for the following reasons:

- Data is manipulated locally, on the DBMS, using high-speed secondary storage devices instead of being transported across a relatively slow network connection.
- The DBMS might have more processing resources at its disposal.
- The DBMS might be capable of optimizing a query for execution in a highly parallel and scalable fashion.

In-database processing for PROC RANK supports the following database management systems:

- Amazon Redshift
- DB2
- Google BigQuery

Note: In-database processing for PROC RANK requires a VAR variable that is not of type FLOAT64.

- Greenplum
- Hadoop
- Impala
- Microsoft SQL Server
- MySQL(as of 2021.1.5)
- Netezza
- Oracle
- PostgreSQL
- SAP HANA
- Snowflake
- Spark
- Teradata
- Vertica
- Yellowbrick

The presence of table statistics might affect the performance of the RANK procedure's in-database processing. If your DBMS is not configured to automatically generate table statistics, then manual generation of table statistics might be necessary to achieve acceptable in-database performance.

Note: For DB2, generation of table statistics (either automatic or manual) is highly recommended for all but the smallest input tables.

If the RANK procedure's input data set is a table or view that resides within a database from which rows would normally be retrieved with the SAS/ACCESS interface to a supported DBMS, then PROC RANK can perform much or all of its work within the DBMS. There are several other factors that determine whether such in-database processing can occur. In-database processing will not occur in the following circumstances:

- if the RENAME= data set option is specified on the input data set.
- if a WHERE statement appears in the context of the RANK procedure or a WHERE= data set option is specified on the input data set, and the WHERE statement or option contains a reference to a SAS function that has no equivalent in the DBMS or a format that has not been installed for use by SAS within the DBMS.
- if any variable specified in a BY statement has an associated format. Formatted BY variables are not supported by PROC RANK for in-database processing.
- if a FORMAT statement appears within the procedure context and applies to a variable specified in a BY statement, then in-database processing cannot be performed. Formatted BY variables are not supported by RANK for in-database processing. With a DBMS, formats can be associated with variables only if a FORMAT or ATTRIB statement appears within the procedure context.
- The TIES=CONDENSE option is not supported for the RANK procedure's in-database processing in an Oracle DBMS. If you use this option, it will prevent SQL generation and execution of in-database processing.

When PROC RANK can process data within the DBMS, it generates an SQL query. The structure of the SQL query that is generated during an in-database invocation of PROC RANK depends on several factors, including these:

- the target DBMS
- the ranking methods that are used
- the number of variables that are ranked
- the inclusion of BY and WHERE statements
- the PROC RANK options that are used, such as TIES= and DESCENDING

The SQL query expresses the required calculations and is submitted to the DBMS. The results of this query will either remain as a new table within the DBMS if the output of the RANK procedure is directed there, or it will be returned to SAS. The settings for the MSGLEVEL option and the SQLGENERATION option determine whether messages will be printed to the SAS log, which indicates whether in-database processing was performed. Generated SQL can be examined by setting the SQL_IP_TRACE option or the SASTRACE= option. SQL_IP_TRACE shows the SQL that is generated by PROC RANK. For more information, see the SASTRACE=

option in *SAS/ACCESS for Relational Databases: Reference* or the SQL_IP_TRACE option in *SAS(R) Analytics Accelerator 1.3 for Teradata: Guide*.

For more information about the settings for system options, library options, data set options, and statement options that affect in-database performance for SAS procedures, see the SQLGENERATION= LIBNAME Option and the SQLGENERATION= option in *SAS/ACCESS for Relational Databases: Reference*.

Syntax: RANK Procedure

Tips:

You can use the ATTRIB, FORMAT, LABEL, and WHERE statements with the RANK procedure. For more information, see [“Statements with the Same Function in Multiple Procedures” on page 23](#).

You can also use any global statement. For a list, see [“Global Statements” on page 23](#) and [“Dictionary of SAS Global Statements” in SAS Global Statements: Reference](#).

For in-database processing to occur, your data must reside within a supported version of the DBMS that has been properly configured for SAS in-database processing. For more information, see [“In-Database Processing for PROC RANK” on page 1706](#).

PROC RANK <options>;

ATTRIB variable-list(s) attribute-list(s);

BY <DESCENDING> variable-1
<<DESCENDING> variable-2 ...>
<NOTSORTED>;

FORMAT variable-1 <...variable-n> <format> <DEFAULT=default-format>;

LABEL variable-1=label-1<...variable-n=label-n>;

VAR data-set-variables(s);

RANKS new-variables(s);

WHERE where-expression-1
< logical-operator where-expression-n>;

Statement	Task	Example
PROC RANK	Compute the ranks for one or more numeric variables in a SAS data set and writes the ranks to a new SAS data set	Ex. 1, Ex. 2, Ex. 3
BY	Calculate a separate set of ranks for each BY group	Ex. 2, Ex. 3
RANKS	Identify a variable to which the ranks are assigned	Ex. 1, Ex. 2
VAR	Specify the variables to rank	Ex. 1, Ex. 2, Ex. 3

PROC RANK Statement

Computes the ranks for one or more numeric variables.

Restriction: Only one ranking method can be specified in a single PROC RANK step.

Examples: [“Example 1: Ranking Values of Multiple Variables” on page 1722](#)
[“Example 2: Ranking Values within BY Groups” on page 1725](#)
[“Example 3: Partitioning Observations into Groups Based on Ranks” on page 1727](#)

Syntax

PROC RANK *<options>*;

Summary of Optional Arguments

Compute fractional ranks

NPLUS1

computes fractional ranks by dividing each rank by the denominator $n+1$.

Create an output data set

OUT=SAS-data-set

names the output data set.

Preserve values

PRESERVERAWBYVALUES

preserves raw values of all BY variables.

Reverse the order of the rankings

DESCENDING

reverses the direction of the ranks.

Specify how to rank tied values

TIES=HIGH | LOW | MEAN | DENSE

specifies how to compute normal scores or ranks for tied data values.

Specify the input data set

DATA=SAS-data-set

specifies the input SAS data set.

Specify the ranking method

FRACTION

computes fractional ranks by dividing each rank by the number of observations having nonmissing values of the ranking variable.

GROUPS=number-of-groups

assigns group values ranging from 0 to *number-of-groups* minus 1.

NORMAL=BLOM | TUKEY | VW

computes normal scores from the ranks.

PERCENT

calculates the percentage of observations with nonmissing values in the rank.

SAVAGE

computes Savage (or exponential) scores from the ranks.

Optional Arguments

DATA=SAS-data-set

specifies the input SAS data set.

Restrictions You cannot use PROC RANK with an engine that supports concurrent access if another user is updating the data set at the same time.

For in-database processing to occur, the data set specification must refer to a table that resides on a supported DBMS.

See [“Input Data Sets” on page 24](#)

DESCENDING

reverses the direction of the ranks. With DESCENDING, the largest value receives a rank of 1, the next largest value receives a rank of 2, and so on. Otherwise, values are ranked from smallest to largest.

See [“Example 1: Ranking Values of Multiple Variables” on page 1722](#)

[“Example 2: Ranking Values within BY Groups” on page 1725](#)

FRACTION

computes fractional ranks by dividing each rank by the number of observations having nonmissing values of the ranking variable.

Alias F

Interaction TIES=HIGH is the default with the FRACTION option. With TIES=HIGH, fractional ranks are considered values of a right-continuous, empirical cumulative distribution function.

See [NPLUS1 option](#)

GROUPS=number-of-groups

assigns group values ranging from 0 to *number-of-groups* minus 1. Common specifications are GROUPS=100 for percentiles, GROUPS=10 for deciles, and GROUPS=4 for quartiles. For example, GROUPS=4 partitions the original values into four groups. The smallest values receive, by default, a quartile value of 0 and the largest values receiving a quartile value of 3.

The formula for calculating group values is as follows:

$$\text{FLOOR}(\text{rank} * k / (n + 1))$$

FLOOR is the FLOOR function, *rank* is the value's order rank, *k* is the value of GROUPS=, and *n* is the number of observations having nonmissing values of the ranking variable for TIES=LOW, TIES=MEAN, and TIES=HIGH. For TIES=DENSE, *n* is the number of observations that have unique nonmissing values.

If the number of observations is evenly divisible by the number of groups, each group has the same number of observations, provided there are no tied values at the boundaries of the groups. Grouping observations by a variable that has many tied values can result in unbalanced groups because PROC RANK always assigns observations with the same value to the same group.

Tip Use DESCENDING to reverse the order of the group values.

See [“Example 3: Partitioning Observations into Groups Based on Ranks” on page 1727](#)

NORMAL=BLOM | TUKEY | VW

computes normal scores from the ranks. The resulting variables appear normally distributed. *n* is the number of observations that have nonmissing values of the ranking variable for TIES=LOW, TIES=MEAN, and TIES=HIGH. For TIES=DENSE, *n* is the number of observations that have unique nonmissing values. The formulas are as follows:

BLOM

$$y_i = \Phi^{-1}((r_i - 3/8) / (n + 1/4))$$

TUKEY

$$y_i = \Phi^{-1}((r_i - 1/3) / (n + 1/3))$$

VW

$$y_i = \Phi^{-1}(r_i / (n + 1))$$

In these formulas, Φ^{-1} is the inverse cumulative normal (PROBIT) function, r_i is the rank of the *i*th observation, and *n* is the number of nonmissing observations for the ranking variable.

VW stands for van der Waerden. With NORMAL=VW, you can use the scores for a nonparametric location test. All three normal scores are approximations to the exact expected order statistics for the normal distribution (also called *normal scores*). The BLOM version appears to fit slightly better than the others (Blom 1958; Tukey 1962).

Restriction Use of the NORMAL= option will prevent in-database processing.

Interaction If you specify the TIES= option, then PROC RANK computes the normal score from the ranks based on non-tied values and applies the TIES= specification to the resulting score.

NPLUS1

computes fractional ranks by dividing each rank by the denominator *n*+1, where *n* is the number of observations that have nonmissing values of the ranking variable for TIES=LOW, TIES=MEAN, and TIES=HIGH. For TIES=DENSE, *n* is the number of observations that have unique nonmissing values.

Aliases	FN1 N1
Interaction	TIES=HIGH is the default with the NPLUS1 option.
See	FRACTION option

OUT=SAS-data-set

names the output data set. If *SAS-data-set* does not exist, PROC RANK creates it. If you omit OUT=, the data set is named using the *DATA**n* naming convention.

Interaction When in-database processing is being performed and OUT= also refers to a supported DBMS table, and if both IN= and OUT= reference the same library, then all processing can occur on the DBMS with results directly populating the output table. In this case, no results will be returned to SAS.

PRESERVERAWBYVALUES

preserves raw values of all BY variables. when those variables are propagated to the output data set. If the PRESERVERAWBYVALUES option is not specified, and one BY variable is specified, then a representative value for each BY group is written to the output data set. If multiple BY variables are specified, then a representative set of values for each BY group is written to the output data set.

PERCENT

divides each rank by the number of observations that have nonmissing values of the variable and multiplies the result by 100 to get a percentage. *n* is the number of observations that have nonmissing values of the ranking variable for TIES=LOW, TIES=MEAN, and TIES=HIGH. For TIES=DENSE, *n* is the number of observations that have unique nonmissing values.

Alias	P
Interaction	TIES=HIGH is the default with the PERCENT option.
Tip	You can use PERCENT to calculate cumulative percentages, but you use GROUPS=100 to compute percentiles.

SAVAGE

computes Savage (or exponential) scores from the ranks by the following formula (Lehman 1998):

$$y_i = \left[\sum_{j=n-r_i+1}^n \left(\frac{1}{j} \right) \right] - 1$$

Interaction If you specify the TIES= option, then PROC RANK computes the Savage score from the ranks based on non-tied values and applies the TIES= specification to the resulting score.

TIES=HIGH | LOW | MEAN | DENSE

specifies how to compute normal scores or ranks for tied data values.

HIGH

assigns the largest of the corresponding ranks (or largest of the normal scores when NORMAL= is specified).

LOW

assigns the smallest of the corresponding ranks (or smallest of the normal scores when NORMAL= is specified).

MEAN

assigns the mean of the corresponding rank (or mean of the normal scores when NORMAL= is specified).

DENSE

computes scores and ranks by treating tied values as a single-order statistic. For the default method, ranks are consecutive integers that begin with the number one and end with the number of unique, nonmissing values of the variable that is being ranked. Tied values are assigned the same rank.

.....
Note: CONDENSE is an alias for DENSE.

Default	MEAN (unless the FRACTION option or PERCENT option is in effect).
Interaction	If you specify the NORMAL= option, then the TIES= specification applies to the normal score, not to the rank that is used to compute the normal score.
See	“Treatment of Tied Values” on page 1704 “Example 1: Ranking Values of Multiple Variables” on page 1722 “Example 2: Ranking Values within BY Groups” on page 1725

ATTRIB Statement

Associates a format, informat, label, and length with one or more variables.

Syntax

ATTRIB *variable-list(s) attribute-list(s);*

Arguments

variable-list(s)

names the variables that you want to associate with the attributes.

TIP List the variables in any form that SAS allows.

attribute-list(s)

specifies one or more attributes to assign to *variable-list*. Specify one or more of these attributes in the ATTRIB statement:

FORMAT=*format*

associates a format with variables in *variable-list*.

Tip The format can be either a standard SAS format or a format that is defined with the FORMAT procedure.

INFORMAT=*informat*

associates an informat with variables in *variable-list*.

Tip The informat can be either a standard SAS informat or an informat that is defined with the FORMAT procedure.

LABEL=*'label'*

associates a label with variables in *variable-list*.

See Also

[“ATTRIB Statement” in SAS DATA Step Statements: Reference.](#)

BY Statement

Produces a separate set of ranks for each BY group.

- Interactions:** If the NOTSORTED option is specified in a BY statement, then in-database processing cannot be performed.
- Application of a format to any BY variable of the input data set, using a FORMAT statement for example, will prevent in-database processing.
- See:** [“Example 3: Partitioning Observations into Groups Based on Ranks” on page 1727](#)
- Examples:** [“Example 2: Ranking Values within BY Groups” on page 1725](#)
[“Example 3: Partitioning Observations into Groups Based on Ranks” on page 1727](#)

Syntax

```
BY <DESCENDING> variable-1
   <<DESCENDING> variable-2 ...>
   <NOTSORTED>;
```


Required Argument

variable

specifies the variable that the procedure uses to form BY groups. You can specify more than one variable. If you do not use the NOTSORTED option in the BY statement, then the observations in the data set must either be sorted by all the variables that you specify or be indexed appropriately. Variables in a BY statement are called *BY variables*.

Optional Arguments

DESCENDING

specifies that the observations are sorted in descending order by the variable that immediately follows the word DESCENDING in the BY statement.

NOTSORTED

specifies that observations are not necessarily sorted in alphabetic or numeric order. The observations are grouped in another way, such as chronological order.

The requirement for ordering or indexing observations according to the values of BY variables is suspended for BY-group processing when you use the NOTSORTED option. In fact, the procedure does not use an index if you specify NOTSORTED. The procedure defines a BY group as a set of contiguous observations that have the same values for all BY variables. If observations with the same values for the BY variables are not contiguous, the procedure treats each contiguous set as a separate BY group.

If you are using a SAS/ACCESS engine, and you specify a BY statement, then the data is always returned in sorted order. If you specify the NOTSORTED option, then it is ignored and in-database processing is performed.

FORMAT Statement

Associates formats with variables.

Syntax

FORMAT *variable-1* <...*variable-n*> *format variable-1* <...*variable-n*> *format*;

Arguments

variable

names one or more variables for SAS to associate with a format. You must specify at least one *variable*.

Tip To disassociate a format from a variable, use the variable in a FORMAT statement without specifying a format in a DATA step or in PROC DATASETS. In a DATA step, place this FORMAT statement after the SET

statement. See [“Removing a Format” in SAS DATA Step Statements: Reference](#). You can also use PROC DATASETS.

format

specifies the format that is listed for writing the values of the variables.

Tip Formats that are associated with variables by using a FORMAT statement behave like formats that are used with a colon modifier in a subsequent PUT statement. For more information about using a colon modifier, see [“PUT Statement: List” in SAS DATA Step Statements: Reference](#).

See [SAS Formats and Informats: Reference](#)

LABEL Statement

Assigns descriptive labels to variables.

Syntax

LABEL *variable-1* = '*text-string*' <...*variable-n* = '*text-string*'>;

LABEL *variable-1* = '' <...*variable-n*=''>; | *variable-1* = <...*variable-n*=>;

Arguments

variable

specifies the variable that you want to label. You can specify labels for multiple variables in a single LABEL statement:

```
data test;
  L = 5;
  W = 10;
  label L = 'Length' W = 'Width';
run;
```

text-string

specifies a label of up to 256 bytes.

```
data test;
  L = 5;
  label L = 'Length';
run;
```

Restrictions If the label includes a semicolon (;) or an equal sign (=), you must enclose the label in either single or double quotation marks.

```
label N = 'Number = ';
```

If the label includes single quotation marks ('), you must enclose the label in double quotation marks.

```
label Name = "Owner's Last Name";
```

There is partial support for variable name substitution with the JAVAIMG device. When the label text for the variable exceeds the space allowed, the variable name is used instead by procedures that are labeling an axis or a legend title. With the exception of the SAS/GRAPH GPLOT procedure, the variable name is not used when the JAVAIMG device is specified.

Notes

Quotation marks are not required unless the label includes a semicolon, quotation marks, or an equal sign.

The LABEL statement expects an equal sign (=) after the first variable. If any other character is found, an error occurs. The LABEL statement considers everything after the equal sign, following the first variable, as part of the label until the statement comes to another variable followed by an equal sign, regardless of quotation marks. In this example, the label for x is **this is x y 4 = this is y** because the next variable that is followed by an equal sign is z.

```
data test;
  x = 1;
  y = 1;
  z = 1;
  label x = 'this is x' y 4 = 'this is y' z = 'This is z';
run;
```

In this LABEL statement, the x variable has the same label:

```
label x = this is x y 4 = this is y z = This is z;
```

See

[“Character Constants and Character Variables” in SAS Programmer’s Guide: Essentials.](#)

”

removes a label from a variable. Enclose a single blank space in quotation marks to remove an existing label. You can remove labels from multiple variables in a single LABEL statement:

```
data test;
  L=5; W=10;
  label L = ' ' W = ' ';
run;
```

Alternatively, you can remove a label by specifying nothing after the equal sign:

```
data test;
  L=5; W=10;
  label L = W = ;
run;
```

WHERE Statement

Subsets the input data set by specifying certain conditions that each observation must meet before it is available for processing.

Syntax

WHERE *where-expression-1*
< *logical-operator where-expression-n*>;

Arguments

where-expression

is an arithmetic or logical expression that generally consists of a sequence of operands and operators.

Tips The operands and operators described in the next several sections are also valid for the WHERE= data set option.

You can specify multiple where-expressions.

logical-operator

can be AND, AND NOT, OR, or OR NOT.

Example

Procedures That Support the WHERE Statement

You can use the WHERE statement with any of the following Base SAS procedures that read a SAS data set:

COMPARE	REPORT
CORR	SORT
DATASETS (APPEND statement)	SQL
FREQ	STANDARD

MEANS or SUMMARY	TABULATE
PRINT	TRANSPOSE
RANK	UNIVARIATE

Details

- The COMPARE procedure and the APPEND statement in PROC DATASETS accept more than one input data set. For more information, see the documentation for the specific procedure.
- To subset the output data set, use the WHERE= data set option:

```
proc report data=debate nowd
              out=onlyfr(where=(year='1')) ;
run;
```

For more information about WHERE=, see [SAS DATA Step Statements: Reference](#).

Temporarily Dissociating a Format from a Variable

In this example, PROC PRINT prints only those observations that meet the condition of the WHERE expression.

```
options nodate pageno=1 linesize=64
              pagesize=40;

proc print data=debate noobs;
  where gpa>3.5;
  title 'Team Members with a GPA';
  title2 'Greater than 3.5';
run;
```

Output

**Team Members with a GPA
Greater than 3.5**

Name	Gender	Year	GPA
Capiccio	m	Freshman	3.598
Tucker	m	Freshman	3.901
Bagwell	f	Sophomore	3.722
Gold	f	Junior	3.609
Syme	f	Junior	3.883
Baglione	f	Senior	4.000
Carr	m	Senior	3.750
Hall	m	Senior	3.574

RANKS Statement

Creates new variables for the rank values.

Default: If you omit the RANKS statement, the rank values replace the original variable values in the output data set.

Requirement: If you use the RANKS statement, you must also use the VAR statement.

Examples: [“Example 1: Ranking Values of Multiple Variables” on page 1722](#)
[“Example 2: Ranking Values within BY Groups” on page 1725](#)

Syntax

RANKS *new-variables(s)*;

Required Argument

new-variable(s)

specifies one or more new variables that contain the ranks for the variable(s) listed in the VAR statement. The first variable listed in the RANKS statement contains the ranks for the first variable listed in the VAR statement. The second variable listed in the RANKS statement contains the ranks for the second variable listed in the VAR statement, and so on.

VAR Statement

Specifies the input variables.

Alias: VARIABLES

Default: If you omit the VAR statement, PROC RANK computes ranks for all numeric variables in the input data set.

Examples: [“Example 1: Ranking Values of Multiple Variables” on page 1722](#)
[“Example 2: Ranking Values within BY Groups” on page 1725](#)
[“Example 3: Partitioning Observations into Groups Based on Ranks” on page 1727](#)

Syntax

VAR *data-set-variables(s)*;

Required Argument

data-set-variable(s)

specifies one or more variables for which ranks are computed.

Details

Using the VAR Statement with the RANKS Statement

The VAR statement is required when you use the RANKS statement. Using these statements together creates the ranking variables named in the RANKS statement that corresponds to the input variables specified in the VAR statement. If you omit the RANKS statement, the rank values replace the original values in the output data set.

Results: RANK Procedure

Missing Values

Missing values are not ranked and are left missing when ranks or rank scores replace the original values in the output data set.

Output Data Set

The RANK procedure creates a SAS data set containing the ranks or rank scores but does not create any printed output. You can use PROC PRINT, PROC REPORT, or another SAS reporting tool to print the output data set.

The output data set contains all the variables from the input data set plus the variables named in the RANKS statement. If you omit the RANKS statement, the rank values replace the original variable values in the output data set.

Numeric Precision

For in-database processing, the mathematical operations expressed by the RANK procedure in SQL, and the order in which they are performed, are essentially the same as those performed within SAS. However, in-database processing might result in small numerical differences when compared to results produced directly by SAS.

Examples: RANK Procedure

Example 1: Ranking Values of Multiple Variables

Features: PROC RANK statement options

DESCENDING
 TIES=
 RANKS statement
 VAR statement
 PRINT procedure

Details

This example performs the following actions:

- reverses the order of the ranks so that the highest value receives the rank of 1
- assigns the best possible rank to tied values
- creates ranking variables and prints them with the original variables

Program

```
options nodate pageno=1 linesize=80 pagesize=60;

data cake;
  input Name $ 1-10 Present 12-13 Taste 15-16;
  datalines;
Davis      77 84
Orlando    93 80
Ramey      68 72
Roe        68 75
Sanders    56 79
Simms      68 77
Strickland 82 79
;

proc rank data=cake out=order descending ties=low;

  var present taste;
  ranks PresentRank TasteRank;
run;

proc print data=order;
  title "Rankings of Participants' Scores";
run;
```

Program Description

Set the SAS system options. The NODATE option specifies to omit the date and time at which the SAS job begins. The PAGENO= option specifies the page number for the next page of output that SAS produces. The LINESIZE= option specifies the line size. The PAGESIZE= option specifies the number of lines for a page of SAS output.

```
options nodate pageno=1 linesize=80 pagesize=60;
```

Create the CAKE data set. This data set contains each participant's last name, score for presentation, and score for taste in a cake-baking contest.

```
data cake;
  input Name $ 1-10 Present 12-13 Taste 15-16;
  datalines;
Davis      77 84
Orlando    93 80
Ramey      68 72
Roe        68 75
Sanders    56 79
Simms      68 77
Strickland 82 79
;
```

Generate the ranks for the numeric variables in descending order and create the Order output data set. DESCENDING reverses the order of the ranks so that the high score receives the rank of 1. TIES=LOW gives tied values the best possible rank. OUT= creates the Order output data set.

```
proc rank data=cake out=order descending ties=low;
```

Create two new variables that contain ranks. The VAR statement specifies the variables to rank. The RANKS statement creates two new variables, PresentRank and TasteRank, that contain the ranks for the variables Present and Taste, respectively.

```
var present taste;
ranks PresentRank TasteRank;
run;
```

Print the data set. PROC PRINT prints the Order data set. The TITLE statement specifies a title.

```
proc print data=order;
  title "Rankings of Participants' Scores";
run;
```

Output: Listing

Output 49.3 *Rankings of Participants' Scores*

Rankings of Participants' Scores						1
Obs	Name	Present	Taste	Present Rank	Taste Rank	
1	Davis	77	84	3	1	
2	Orlando	93	80	1	2	
3	Ramey	68	72	4	7	
4	Roe	68	75	4	6	
5	Sanders	56	79	7	3	
6	Simms	68	77	4	5	
7	Strickland	82	79	2	3	

Example 2: Ranking Values within BY Groups

Features:

- PROC RANK statement options
 - DESCENDING
 - TIES=
- BY statement
- RANKS statement
- VAR statement
- PRINT procedure

Details

This example performs the following actions:

- ranks observations separately within BY groups
- reverses the order of the ranks so that the highest value receives the rank of 1
- assigns the best possible rank to tied values
- creates ranking variables and prints them with the original variables

Program

```
options nodate pageno=1 linesize=80 pagesize=60;

data elect;
    input Candidate $ 1-11 District 13 Vote 15-18 Years 20;
    datalines;
Cardella      1 1689 8
Latham        1 1005 2
Smith         1 1406 0
Walker        1  846 0
Hinkley       2  912 0
Kreitemeyer   2 1198 0
Lundell       2 2447 6
Thrash        2  912 2
;

proc rank data=elect out=results ties=low descending;
    by district;

    var vote years;
    ranks VoteRank YearsRank;
run;

proc print data=results n;
```

```

    by district;
    title 'Results of City Council Election';
run;

```

Program Description

Set the SAS system options. The NODATE option specifies to omit the date and time at which the SAS job begins. The PAGENO= option specifies the page number for the next page of output that SAS produces. The LINESIZE= option specifies the line size. The PAGESIZE= option specifies the number of lines for a page of SAS output.

```
options nodate pageno=1 linesize=80 pagesize=60;
```

Create the Elect data set. This data set contains each candidate's last name, district number, vote total, and number of years' experience on the city council.

```

data elect;
    input Candidate $ 1-11 District 13 Vote 15-18 Years 20;
    datalines;
Cardella      1 1689 8
Latham        1 1005 2
Smith         1 1406 0
Walker        1  846 0
Hinkley       2  912 0
Kreitemeyer  2 1198 0
Lundell       2 2447 6
Thrash        2  912 2
;

```

Generate the ranks for the numeric variables in descending order and create the Results output data set. DESCENDING reverses the order of the ranks so that the highest vote total receives the rank of 1. TIES=LOW gives tied values the best possible rank. OUT= creates the Results output data set.

```
proc rank data=elect out=results ties=low descending;
```

Create a separate set of ranks for each BY group. The BY statement separates the rankings by values of District.

```
    by district;
```

Create two new variables that contain ranks. The VAR statement specifies the variables to rank. The RANKS statement creates the new variables, VoteRank and YearsRank, that contain the ranks for the variables Vote and Years, respectively.

```

    var vote years;
    ranks VoteRank YearsRank;
run;

```

Print the data set. PROC PRINT prints the Results data set. The N option prints the number of observations in each BY group. The TITLE statement specifies a title.

```

proc print data=results n;
    by district;
    title 'Results of City Council Election';
run;

```

Output: Listing

In the following output, Hinkley and Thrash tied with 912 votes in the second district. They both receive a rank of 3 because TIES=LOW.

Output 49.4 Results of City Council Election

Results of City Council Election						1
----- District=1 -----						
Obs	Candidate	Vote	Years	Vote Rank	Years Rank	
1	Cardella	1689	8	1	1	
2	Latham	1005	2	3	2	
3	Smith	1406	0	2	3	
4	Walker	846	0	4	3	
N = 4						
----- District=2 -----						
Obs	Candidate	Vote	Years	Vote Rank	Years Rank	
5	Hinkley	912	0	3	3	
6	Kreitemeyer	1198	0	2	3	
7	Lundell	2447	6	1	1	
8	Thrash	912	2	3	2	
N = 4						

Example 3: Partitioning Observations into Groups Based on Ranks

Features:

- PROC RANK statement option GROUPS=
- BY statement
- VAR statement
- PRINT procedure
- SORT procedure

Details

This example performs the following actions:

- partitions observations into groups on the basis of values of two input variables
- groups observations separately within BY groups
- replaces the original variable values with the group values

Program

```
options nodate pageno=1 linesize=80 pagesize=60;

data swim;
    input Name $ 1-7 Gender $ 9 Back 11-14 Free 16-19;
    datalines;
Andrea  F 28.6 30.3
Carole  F 32.9 24.0
Clayton M 27.0 21.9
Curtis  M 29.0 22.6
Doug    M 27.3 22.4
Ellen   F 27.8 27.0
Jan     F 31.3 31.2
Jimmy   M 26.3 22.5
Karin   F 34.6 26.2
Mick    M 29.0 25.4
Richard M 29.7 30.2
Sam     M 27.2 24.1
Susan   F 35.1 36.1
;

proc sort data=swim out=pairs;
    by gender;
run;

proc rank data=pairs out=rankpair groups=3;
    by gender;
    var back free;
run;

proc print data=rankpair n;
    by gender;
    title 'Pairings of Swimmers for Backstroke and Freestyle';
run;
```

Program Description

Set the SAS system options. The NODATE option specifies to omit the date and time at which the SAS job began. The PAGENO= option specifies the page number for the next page of output that SAS produces. The LINESIZE= option specifies the line size. The PAGESIZE= option specifies the number of lines for a page of SAS output.

```
options nodate pageno=1 linesize=80 pagesize=60;
```

Create the Swim data set. This data set contains swimmers' first names and their times, in seconds, for the backstroke and the freestyle. This example groups the swimmers into pairs, within male and female classes, based on times for both strokes so that every swimmer is paired with someone who has a similar time for each stroke.

```
data swim;
  input Name $ 1-7 Gender $ 9 Back 11-14 Free 16-19;
  datalines;
Andrea F 28.6 30.3
Carole F 32.9 24.0
Clayton M 27.0 21.9
Curtis M 29.0 22.6
Doug M 27.3 22.4
Ellen F 27.8 27.0
Jan F 31.3 31.2
Jimmy M 26.3 22.5
Karin F 34.6 26.2
Mick M 29.0 25.4
Richard M 29.7 30.2
Sam M 27.2 24.1
Susan F 35.1 36.1
;
```

Sort the Swim data set and create the Pairs output data set. PROC SORT sorts the data set by Gender. This action is required to obtain a separate set of ranks for each group. OUT= creates the Pairs output data set.

```
proc sort data=swim out=pairs;
  by gender;
run;
```

Generate the ranks that are partitioned into three groups and create an output data set. GROUPS=3 assigns one of three possible group values (0,1,2) to each swimmer for each stroke. OUT= creates the Rankpair output data set.

```
proc rank data=pairs out=rankpair groups=3;
```

Create a separate set of ranks for each BY group. The BY statement separates the rankings by Gender.

```
  by gender;
```

Replace the original values of the variables with the rank values. The VAR statement specifies that Back and Free are the variables to rank. With no RANKS statement, PROC RANK replaces the original variable values with the group values in the output data set.

```
  var back free;
```

```
run;
```

Print the data set. PROC PRINT prints the Rankpair data set. The N option prints the number of observations in each BY group. The TITLE statement specifies a title.

```
proc print data=rankpair n;
  by gender;
  title 'Pairings of Swimmers for Backstroke and Freestyle';
run;
```

Output: Listing

In the following output, the group values pair swimmers with similar times to work on each stroke. For example, Andrea and Ellen work together on the backstroke because they have the fastest times in the female class. The groups of male swimmers are unbalanced because there are seven male swimmers; for each stroke, one group has three swimmers.

Output 49.5 *Pairings of Swimmers for Backstroke and Freestyle*

Pairings of Swimmers for Backstroke and Freestyle				1
----- Gender=F -----				
Obs	Name	Back	Free	
1	Andrea	0	1	
2	Carole	1	0	
3	Ellen	0	1	
4	Jan	1	2	
5	Karin	2	0	
6	Susan	2	2	
N = 6				
----- Gender=M -----				
Obs	Name	Back	Free	
7	Clayton	0	0	
8	Curtis	2	1	
9	Doug	1	0	
10	Jimmy	0	1	
11	Mick	2	2	
12	Richard	2	2	
13	Sam	1	1	
N = 7				

References

- Blom, G. 1958. *Statistical Estimates and Transformed Beta Variables*. New York, New York: John Wiley & Sons, Inc.
- Conover, W.J. 1998. *Practical Nonparametric Statistics, Third Edition*. New York, New York: John Wiley & Sons, Inc.
- Conover, W.J. and R.L. Iman. 1976. "On Some Alternative Procedures Using Ranks for the Analysis of Experimental Designs." *Communications in Statistics* A5 (14): 1348–1368.
- Conover, W.J. and R.L. Iman. 1981. "Rank Transformations as a Bridge between Parametric and Nonparametric Statistics." *The American Statistician* 35: 124–129.
- Iman, R.L. and W.J. Conover. 1979. "The Use of the Rank Transform in Regression." *Technometrics* 21: 499–509.
- Lehman, E.L. 1998. *Nonparametrics: Statistical Methods Based on Ranks*. Upper Saddle River, New Jersey: Prentice Hall.
- Quade, D. 1966. "On Analysis of Variance for the K-Sample Problem." *Annals of Mathematical Statistics* 37: 1747–1758.
- Tukey, John W. 1962. "The Future of Data Analysis." *Annals of Mathematical Statistics* 33: 22.

REGISTRY Procedure

Overview: REGISTRY Procedure	1733
What Does the REGISTRY Procedure Do?	1733
Syntax: REGISTRY Procedure	1734
PROC REGISTRY Statement	1734
Usage: REGISTRY Procedure	1740
Structure of a Registry File	1740
Specifying Key Names	1741
Specifying Values for Keys	1741
Sample Registry Entries	1742
Examples: REGISTRY Procedure	1743
Example 1: Importing a File to the SAS Registry	1743
Example 2: Listing and Exporting the Registry File	1744
Example 3: Comparing the Registry to an External File	1745
Example 4: Comparing Registry Files	1747
Example 5: Specifying an Entire Key Sequence with the STARTAT= Option	1748
Example 6: Displaying a List of Fonts	1749

Overview: REGISTRY Procedure

What Does the REGISTRY Procedure Do?

The REGISTRY procedure maintains the SAS registry. The registry consists of two parts. One part is stored in the Sashelp library, and the other part is stored in the Sasuser library.

The REGISTRY procedure enables you to do the following:

- Import registry files to populate the Sashelp and Sasuser registries.

- Export all or part of the registry to another file.
- List the contents of the registry in the SAS log.
- Compare the contents of the registry to a file.
- Uninstall a registry file.
- Deliver detailed status information when a key or value will be overwritten or uninstalled.
- Clear out entries in the Sasuser registry.
- Validate that the registry exists.
- List diagnostic information.

For more information, see [The SAS Registry](#).

Syntax: REGISTRY Procedure

PROC REGISTRY *<options>;*

Statement	Task	Example
PROC REGISTRY	Manage registry files	Ex. 1, Ex. 2, Ex. 3, Ex. 4, Ex. 5, Ex. 6

PROC REGISTRY Statement

Maintains the SAS registry.

Examples:

[“Example 1: Importing a File to the SAS Registry” on page 1743](#)

[“Example 3: Comparing the Registry to an External File” on page 1745](#)

[“Example 2: Listing and Exporting the Registry File” on page 1744](#)

[“Example 4: Comparing Registry Files” on page 1747](#)

[“Example 5: Specifying an Entire Key Sequence with the STARTAT= Option” on page 1748](#)

[“Example 6: Displaying a List of Fonts” on page 1749](#)

Syntax

PROC REGISTRY *<options>;*

Summary of Optional Arguments

CLEARsasUSER

erases from the Sasuser registry the keys that were added by a user.

COMPAREREG1=*'libname.registry-name-1'*

compares two registry files.

COMPAREREG2=*'libname.registry-name-2'*

compares two registry files.

COMPARETO=*file-specification*

compares the contents of a registry to a file.

DEBUGOFF

disables registry debugging.

DEBUGON

enables registry debugging.

EXPORT=*file-specification*

writes the contents of a registry to the specified file.

FOLLOWLINKS

follows links that are found when processing the LIST command.

FULLSTATUS

provides additional information in the SAS log about the results of the IMPORT= and UNINSTALL= options.

IMPORT=*file-specification*

imports the specified file to a registry.

KEYONLY

limits the LIST, LISTUSER, LISTHELP, and LISTREG options output to display keys only.

LEVELS=*n*

limits the number of levels to display for the LIST, LISTUSER, LISTHELP, and LISTREG options.

LIST

writes the contents of the registry to the SAS log. This option is used with the STARTAT= option to list specific keys.

LISTHELP

writes the contents of the Sashelp portion of the registry to the SAS log.

LISTREG=*'libname.registry-name'*

sends the contents of a registry to the log.

LISTUSER

writes the contents of the Sasuser portion of the registry to the SAS log.

STARTAT=*'key-name'*

starts exporting or writing or comparing the contents of a registry at the specified key.

UNINSTALL=*file-specification*

deletes from the specified registry all the keys and values that are in the specified file.

UPCASE

uses uppercase for all incoming key names.

UPCASEALL

uses uppercase for all keys, names, and item values when you import a file.

USESASHELP

performs the specified operation on the Sashelp portion of the SAS registry.

Optional Arguments

CLEARASUSER

erases from the Sasuser portion of the SAS registry the keys that were added by a user.

COMPAREREG1=*'libname.registry-name-1'*

specifies one of two registries to compare. The results appear in the SAS log.

libname

is the name of the library in which the registry file resides.

registry-name-1

is the name of the first registry.

Requirement COMPAREREG1 must be used with COMPAREREG2.

Interaction To specify a single key and all of its subkeys, specify the STARTAT= option.

Example [“Example 4: Comparing Registry Files” on page 1747](#)

COMPAREREG2=*'libname.registry-name-2'*

specifies the second of two registries to compare. The results appear in the SAS log.

libname

is the name of the library in which the registry file resides.

registry-name-2

is the name of the second registry.

Requirement COMPAREREG2 must be used with COMPAREREG1.

Example [“Example 4: Comparing Registry Files” on page 1747](#)

COMPARETO=*file-specification*

compares the contents of a file that contains registry information to a registry. It returns information about keys and values that it finds in the file that are not in the registry. It reports the following items as differences:

- keys that are defined in the external file but not in the registry
- value names for a given key that are in the external file but not in the registry
- differences in the content of like-named values in like-named keys

COMPARETO= does not report as differences any keys and values that are in the registry but not in the file because the registry could easily be composed of pieces from many different files.

File-specification is one of the following values:

'external-file'

is the path and name of an external file that contains the registry information.

fileref

is a fileref that has been assigned to an external file.

Requirement You must have previously associated the fileref with an external file in a FILENAME statement, a FILENAME function, the Explorer window, or an appropriate operating environment command.

Interaction By default, PROC REGISTRY compares *file-specification* to the Sasuser portion of the registry. To compare *file-specification* to the Sashelp portion of the registry, specify the option USESASHELP.

See [Usage: REGISTRY Procedure on page 1740](#) for information about how to structure a file that contains registry information

Example [“Example 3: Comparing the Registry to an External File” on page 1745](#)

DEBUGON

enables registry debugging by providing more descriptive log entries.

DEBUGOFF

disables registry debugging.

EXPORT=file-specification

writes the contents of a registry to the specified file, where *file-specification* is one of the following values:

'external-file'

is the name of an external file that contains the registry information.

fileref

is a fileref that has been assigned to an external file.

Requirement You must have previously associated the fileref with an external file in a FILENAME statement, a FILENAME function, the Explorer window, or an appropriate operating environment command.

If *file-specification* already exists, then PROC REGISTRY overwrites it. Otherwise, PROC REGISTRY creates the file.

Interactions By default, EXPORT= writes the Sasuser portion of the registry to the specified file. To write the Sashelp portion of the registry, specify the USESASHELP option. You must have Write permission to the Sashelp library to use USESASHELP.

To export a single key and all of its subkeys, specify the STARTAT= option.

Example [“Example 2: Listing and Exporting the Registry File” on page 1744](#)

FOLLOWLINKS

follows links that are found when processing the LIST option.

Normally the LIST option displays the values of the link items. If you use the FOLLOWLINKS option, the links are treated as keys, and items contained in the links are displayed.

FULLSTATUS

lists the keys, subkeys, and values that were added or deleted as a result of running the IMPORT= and UNINSTALL options.

IMPORT=*file-specification*

specifies the file to import into the SAS registry. PROC REGISTRY does not overwrite the existing registry. Instead, it updates the existing registry with the contents of the specified file.

Note: The .sasxreg file extension is not required.

File-specification is one of the following values:

'external-file'

is the path and name of an external file that contains the registry information.

fileref

is a fileref that has been assigned to an external file.

Requirement You must have previously associated the fileref with an external file in a FILENAME statement, a FILENAME function, the Explorer window, or an appropriate operating environment command.

Interactions By default, IMPORT= imports the file to the Sasuser portion of the SAS registry. To import the file to the Sashelp portion of the registry, specify the USESASHELP option. You must have Write permission to Sashelp to use USESASHELP.

To obtain additional information in the SAS log as you import a file, use FULLSTATUS.

See [Usage: REGISTRY Procedure on page 1740](#) for information about how to structure a file that contains registry information

Example [“Example 1: Importing a File to the SAS Registry” on page 1743](#)

KEYSONLY

limits the LIST, LISTUSER, LISTHELP, and LISTREG options output to display keys only.

LEVELS=*n*

limits the number of levels to display for the LIST, LISTUSER, LISTHELP, and LISTREG options.

Requirement LEVEL ≥ 1. LEVELS=0 behaves as if LEVELS was not specified.

LIST

writes the contents of the entire SAS registry to the SAS log.

Interaction To write a single key and all of its subkeys, use the STARTAT= option.

LISTHELP

writes the contents of the Sashelp portion of the registry to the SAS log.

Interaction To write a single key and all of its subkeys, use the STARTAT= option.

LISTREG='libname.registry-name'

lists the contents of the specified registry in the log.

libname

is the name of the library in which the registry file resides.

registry-name

is the name of the registry.

Here is an example:

```
proc registry listreg='sashelp.registry';
run;
```

Interaction To list a single key and all of its subkeys, use the STARTAT= option.

LISTUSER

writes the contents of the Sasuser portion of the registry to the SAS log.

Interaction To write a single key and all of its subkeys, use the STARTAT= option.

Example [“Example 2: Listing and Exporting the Registry File” on page 1744](#)

STARTAT='key-name'

exports or writes the contents of a single key and all of its subkeys.

You must specify an entire key sequence if you want to start listing at any subkey under the root key.

Interaction Use STARTAT= with the EXPORT=, LIST, LISTHELP, LISTUSER, COMPARE1=, COMPARE2= and the LISTREG options.

Example [“Example 4: Comparing Registry Files” on page 1747](#)

UNINSTALL=file-specification

deletes from the specified registry all the keys and values that are in the specified file.

File-specification is one of the following values:

'external-file'

is the name of an external file that contains the keys and values to delete.

fileref

is a fileref that has been assigned to an external file. To assign a fileref, you can do the following:

- use the Explorer Window
- use the [“FILENAME Statement” in SAS Global Statements: Reference](#)

Interactions By default, UNINSTALL deletes the keys and values from the Sasuser portion of the SAS registry. To delete the keys and values from the Sashelp portion of the registry, specify the USESASHELP option. You must have Write permission to Sashelp to use this option.

Use FULLSTATUS to obtain additional information in the SAS log as you uninstall a registry.

See [Usage: REGISTRY Procedure on page 1740](#) for information about how to structure a file that contains registry information

UPCASE

uses uppercase for all incoming key names.

UPCASEALL

uses uppercase for all keys, names, and item values when you import a file.

USESASHELP

performs the specified operation on the Sashelp portion of the SAS registry.

Interaction Use USESASHELP with the IMPORT=, EXPORT=, COMPARETO, or UNINSTALL option. To use USESASHELP with IMPORT= or UNINSTALL, you must have Write permission to Sashelp.

Usage: REGISTRY Procedure

Structure of a Registry File

You can create registry files with the SAS Registry Editor or with any text editor.

A registry file must have a particular structure. Each entry in the registry file consists of a key name, followed on the next line by one or more values. The key name identifies the key or subkey that you are defining. Any values that follow specify the names or data to associate with the key.

Specifying Key Names

Key names are entered on a single line between square brackets ([and]). To specify a subkey, enter multiple key names between the brackets, starting with the root key. Separate the names in a sequence of key names with a backslash (\). The length of a single key name or a sequence of key names cannot exceed 255 characters (including the square brackets and the backslashes). Key names can contain any character except the backslash.

Examples of valid key name sequences follow. These sequences are typical of the SAS registry:

- [CORE\EXPLORER\MENUS\ENTRIES\CLASS]
- [CORE\EXPLORER\NEWMEMBER\CATALOG]
- [CORE\EXPLORER\NEWENTRY\CLASS]
- [CORE\EXPLORER\CONS\ENTRIES\LOG]

Specifying Values for Keys

Enter each value on the line that follows the key name that it is associated with. You can specify multiple values for each key, but each value must be on a separate line.

The general form of a value is:

value-name=value-content

A *value-name* can be an at sign (@), which indicates the default value name, or it can be any text string in double quotation marks. If the text string contains an ampersand (&), then the character (either uppercase or lowercase) that follows the ampersand is a shortcut for the value name. For more information, see [“Sample Registry Entries” on page 1742](#).

The entire text string cannot contain more than 255 characters (including quotation marks and ampersands). It can contain any character except a backslash (\).

Value-content can be any of the following:

- the string **double**: followed by a numeric value.
- a string. You can put any character inside the quotation marks, including nothing ("").

Note: To include a backslash in the string that is enclosed in quotation marks, use two adjacent backslashes. To include a double quotation mark, use two adjacent double quotation marks.

- the string **hex:** followed by any number of hexadecimal characters, up to the 255-character limit, separated by commas. If you extend the hexadecimal characters beyond a single line, then end the line with a backslash to indicate that the data continues on the next line. Hexadecimal values can also be referred to as "binary values" in the Registry Editor.
- the string **dword:** followed by an unsigned long hexadecimal value.
- the string **int:** followed by a signed long integer value.
- the string **uint:** followed by an unsigned long integer value.

The following list contains a sample of valid registry values:

- a double value=double:2.4E-44
- a string="my data"
- binary data=hexadecimal: 01,00,76,63,62,6B
- dword=dword:00010203
- signed integer value=int:-123
- unsigned integer value (decimal)=dword:0001E240

Sample Registry Entries

Registry entries can vary in content and appearance, depending on their purpose.

To see what the actual registry text file looks like, you can use PROC REGISTRY to write the contents of the registry key to the SAS log, using the LISTUSER and STARTAT= options.

The following example shows the syntax for sending a Sasuser registry entry to the log:

```
proc registry
  listuser
  startat='sasuser-registry-key-name';
run;
```

The following example shows a value for the STARTAT= option:

```
proc registry
  listuser
  startat='HKEY_SYSTEM_ROOT\CORE\PRINTING\PRINTERS\PostScript
\DEFAULT
SETTINGS';
run;
```

In the following example, the list of subkeys begins at the CORE\PRINTING\PRINTERS\PostScript\DEFAULT SETTINGS key.

Example Code 50.1 A Registry Entry for a PostScript Printer

NOTE: Contents of SASUSER REGISTRY starting at subkey [CORE\PRINTING\PRINTERS\PostScript\DEFAULT SETTINGS key]

```
Font Character Set="Western"
Font Size=double:12
Font Style="Regular"
Font Typeface="Courier"
Font Weight="Normal"
Margin Bottom=double:0.5
Margin Left=double:0.5
Margin Right=double:0.5
Margin Top=double:0.5
Margin Units="IN"
Paper Destination=""
Paper Size="Letter"
Paper Source=""
Paper Type=""
Resolution="300 DPI"
```

Examples: REGISTRY Procedure

Example 1: Importing a File to the SAS Registry

Features: IMPORT=
 FILENAME statement

Details

This example imports a file into the Sasuser portion of the SAS registry. The following source file contains examples of valid key name sequences in a registry file:

```
[HKEY_USER_ROOT\AllGoodPeopleComeToTheAidOfTheirCountry]
@="This is a string value"
"Value2"=""
"Value3"="C:\\This\\Is\\Another\\String\\Value"
```

Program

```
filename source 'external-file';  
proc registry import=source;  
run;
```

Program Description

Assign a fileref to a file that contains valid text for the registry. The FILENAME statement assigns the fileref SOURCE to the external file that contains the text to read into the registry.

```
filename source 'external-file';
```

Invoke PROC REGISTRY to import the file that contains input for the registry. PROC REGISTRY reads the input file that is identified by the fileref SOURCE. IMPORT= writes to the Sasuser portion of the SAS registry by default.

```
proc registry import=source;  
run;
```

Log

Example Code 50.2 Results from Importing a File to the SAS Registry

```
Parsing REG file and loading the registry please wait....  
Registry IMPORT is now complete.
```

Example 2: Listing and Exporting the Registry File

Features: EXPORT=
 LISTUSER

Details

The registry file is usually very large. To export a portion of the registry, use the STARTAT= option.

This example lists the Sasuser portion of the SAS registry and exports it to an external file.

Program

```
proc registry
  listuser

  export='external-file';
run;
```

Program Description

Write the contents of the Sasuser portion of the registry to the SAS log. The LISTUSER option causes PROC REGISTRY to write the entire Sasuser portion of the registry to the log.

```
proc registry
  listuser
```

Export the registry to the specified file. The EXPORT= option writes a copy of the Sasuser portion of the SAS registry to the external file.

```
  export='external-file';
run;
```

Log

Example Code 50.3 Results from Listing and Exporting a SAS Registry File

```
Starting to write out the registry file, please wait...
The export to file external-file is now complete.
Contents of SASUSER REGISTRY.
[ HKEY_USER_ROOT]
[   CORE]
[     EXPLORER]
[       CONFIGURATION]
        Initialized= "True"
[     FOLDERS]
[       UNXHOST1]
        Closed= "658"
        Icon= "658"
        Name= "Home Directory"
        Open= "658"
        Path= "~"
```

Example 3: Comparing the Registry to an External File

Features: COMPARETO= option

FILENAME statement

Details

This example compares the Sasuser portion of the SAS registry to an external file. Comparisons such as this one are useful if you want to know the difference between a backup file that was saved with a .txt file extension and the current registry file.

To compare the Sashelp portion of the registry with an external file, specify the USESASHELP option.

This SAS log shows two differences between the Sasuser portion of the registry and the specified external file. In the registry, the value of "Initialized" is "True"; in the external file, it is "False". In the registry, the value of "Icon" is "658"; in the external file it is "343".

Program

```
filename testreg 'external-file';

proc registry
    compareto=testreg;
run;
```

Program Description

Assign a fileref to the external file that contains the text to compare to the registry. The FILENAME statement assigns the fileref TESTREG to the external file.

```
filename testreg 'external-file';
```

Compare the specified file to the Sasuser portion of the SAS registry. The COMPARETO option compares the contents of a file to a registry. It returns information about keys and values that it finds in the file that are not in the registry.

```
proc registry
    compareto=testreg;
run;
```


Log

Example Code 50.4 Results from Comparing the Registry to an External File

```
Parsing REG file and comparing the registry please wait....
COMPARE DIFF: Value "Initialized" in
[HKEY_USER_ROOT\CORE\EXPLORER\CONFIGURATION]: REGISTRY TYPE=STRING, CURRENT
VALUE="True"
COMPARE DIFF: Value "Initialized" in
[HKEY_USER_ROOT\CORE\EXPLORER\CONFIGURATION]: FILE TYPE=STRING, FILE
VALUE="False"
COMPARE DIFF: Value "Icon" in
[HKEY_USER_ROOT\CORE\EXPLORER\FOLDERS\UNXHOST1]: REGISTRY TYPE=STRING,
CURRENT VALUE="658"
COMPARE DIFF: Value "Icon" in
[HKEY_USER_ROOT\CORE\EXPLORER\FOLDERS\UNXHOST1]: FILE TYPE=STRING, FILE
VALUE="343"
Registry COMPARE is now complete.
COMPARE: There were differences between the registry and the file.
```

Example 4: Comparing Registry Files

Features: COMPAREREG1= and COMPAREREG2= options
 STARTAT= option

Details

This example uses the REGISTRY procedure options COMPAREREG1= and COMPAREREG2= to specify two registry files for comparison.

Program

```
libname proclib 'SAS-library';
proc registry comparereg1='sasuser.registry'
    startat='CORE\EXPLORER'
    comparereg2='proclib.registry';
run;
```

Program Description

Declare the PROCLIB library. The PROCLIB library contains a registry file.

```
libname proclib 'SAS-library';
```

Start PROC REGISTRY and specify the first registry file to be used in the comparison.

```
proc registry comparereg1='sasuser.regstry'
```

Limit the comparison to the registry keys including and following the specified registry key. The STARTAT= option limits the scope of the comparison to the EXPLORER subkey under the CORE key. By default the comparison includes the entire contents of both registries.

```
startat='CORE\EXPLORER'
```

Specify the second registry file to be used in the comparison.

```
comparereg2='proclib.regstry';  
run;
```

Log

Example Code 50.5 Results from Comparing Two Registry Files

```
NOTE: Comparing registry SASUSER.REGSTRY to registry PROCLIB.REGSTRY
NOTE: Diff in Key (CORE\EXPLORER\MENUS\FILES\SAS) Item (1;&Open)
SASUSER.REGSTRY Type: String len 17 data PGM;INCLUDE '%s';
PROCLIB.REGSTRY Type: String len 15 data WHOSTEDIT '%s';

NOTE: Diff in Key (CORE\EXPLORER\MENUS\FILES\SAS) Item (3;&Submit)
SASUSER.REGSTRY Type: String len 23 data PGM;INCLUDE '%s';SUBMIT
PROCLIB.REGSTRY Type: String len 21 data WHOSTEDIT '%s';SUBMIT

NOTE: Diff in Key (CORE\EXPLORER\MENUS\FILES\SAS) Item (4;&Remote Submit)
SASUSER.REGSTRY Type: String len 35 data SIGNCHECK;PGM;INCLUDE '%s';RSUBMIT;
PROCLIB.REGSTRY Type: String len 33 data SIGNCHECK;WHOSTEDIT '%s';RSUBMIT;

NOTE: Diff in Key (CORE\EXPLORER\MENUS\FILES\SAS) Item (@)
SASUSER.REGSTRY Type: String len 17 data PGM;INCLUDE '%s';
PROCLIB.REGSTRY Type: String len 15 data WHOSTEDIT '%s';

NOTE: Item (2;Open with &Program Editor) in key
(CORE\EXPLORER\MENUS\FILES\TXT) not found in registry PROCLIB.REGSTRY
NOTE: Diff in Key (CORE\EXPLORER\MENUS\FILES\TXT) Item (4;&Submit)
SASUSER.REGSTRY Type: String len 24 data PGM;INCLUDE '%s';SUBMIT;
PROCLIB.REGSTRY Type: String len 22 data WHOSTEDIT '%s';SUBMIT;

NOTE: Diff in Key (CORE\EXPLORER\MENUS\FILES\TXT) Item (5;&Remote Submit)
SASUSER.REGSTRY Type: String len 35 data SIGNCHECK;PGM;INCLUDE '%s';RSUBMIT;
PROCLIB.REGSTRY Type: String len 33 data SIGNCHECK;WHOSTEDIT '%s';RSUBMIT;
```

Example 5: Specifying an Entire Key Sequence with the STARTAT= Option

Features: EXPORT option

STARTAT= option

Details

The following example shows how to use the STARTAT= option. You must specify an entire key sequence if you want to start listing any subkey under the root key. The root key is optional.

Program

```
proc registry export = my-fileref
startat='core\explorer\icons';
run;
```

Example 6: Displaying a List of Fonts

Features: LISTHELP option
 STARTAT option

Details

The following example writes a list of ODS fonts to the SAS log.

Program

```
proc registry clearsasuser; run;
proc registry listhelp startat='ods\fonts'; run;
proc registry clearsasuser; run;
```

Log

Example Code 50.6 Results from Displaying a List of Fonts from the SAS Registry

```
NOTE: Contents of SASHELP REGISTRY starting at subkey [ods\fonts]
[  ods\fonts]
  dings="Wingdings"
  monospace="Courier New"
  MTdings="Monotype Sorts"
  MTmonospace="Cumberland AMT"
  MTsans-serif="Albany AMT"
  MTsans-serif-unicode="Monotype Sans WT J"
  MTserif="Thorndale AMT"
  MTserif-unicode="Thorndale Duospace WT J"
  MTsymbol="Symbol MT"
  sans-serif="Arial"
  serif="Times New Roman"
  symbol="Symbol"
[    ja_JP]
  dings="Wingdings"
  monospace="MS Gothic"
  MTdings="Wingdings"
  MTmonospace="MS Gothic"
  MTsans-serif="MS PGothic"
  MTsans-serif-unicode="MS PGothic"
  MTserif="MS PMincho"
  MTserif-unicode="MS PMincho"
  MTsymbol="Symbol"
  sans-serif="MS PGothic"
  serif="MS PMincho"
  symbol="Symbol"
[    ko_KR]
  dings="Wingdings"
  monospace="GulimChe"
  MTdings="Wingdings"
  MTmonospace="GulimChe"
  MTsans-serif="Batang"
  MTsans-serif-unicode="Batang"
  MTserif="Gulim"
  MTserif-unicode="Gulim"
  MTsymbol="Symbol"
  sans-serif="Batang"
  serif="Gulim"
  symbol="Symbol"
[    th_TH]
  dings="Wingdings"
  monospace="Thorndale Duospace WT J"
  MTdings="Monotype Sorts"
  MTmonospace="Cumberland AMT"
  MTsans-serif="Monotype Sans WT J"
  MTsans-serif-unicode="Thorndale Duospace WT J"
  MTserif="Thorndale Duospace WT J"
  MTserif-unicode="Monotype Sans WT J"
  MTsymbol="Symbol MT"
  sans-serif="Angsana New"
  serif="Thorndale Duospace WT J"
  symbol="Symbol"
```

REPORT Procedure

Overview: REPORT Procedure	1751
What Does the REPORT Procedure Do?	1752
What Types of Reports Can PROC REPORT Produce?	1753
What Do the Various Types of Reports Look Like?	1753
Concepts: REPORT Procedure	1758
Laying Out a Report	1758
Using Compute Blocks	1764
Using Break Lines	1770
Using Compound Names	1771
ODS Destinations Supported by PROC REPORT	1773
Threaded Processing of Input Data Sets	1773
Syntax: REPORT Procedure	1774
PROC REPORT Statement	1776
ATTRIB Statement	1790
BREAK Statement	1791
BY Statement	1796
CALL DEFINE Statement	1800
COLUMN Statement	1803
COMPUTE Statement	1806
DEFINE Statement	1810
ENDCOMP Statement	1822
FORMAT Statement	1822
FREQ Statement	1823
LABEL Statement	1824
LINE Statement	1825
RBREAK Statement	1827
WEIGHT Statement	1829
WHERE Statement	1830
Usage: REPORT Procedure	1833
Statistics That Are Available in PROC REPORT	1833
Using ODS Styles with PROC REPORT	1835
Storing and Reusing a Report Definition	1846
In-Database Processing for PROC REPORT	1846
CAS Processing for PROC REPORT	1847
Substituting BY Line Values in a Text String	1849
Results: REPORT Procedure	1850
How PROC REPORT Builds a Report	1850

Examples: REPORT Procedure	1862
Example 1: Selecting Variables and Creating a Summary Line for a Report	1862
Example 2: Ordering the Rows in a Report	1866
Example 3: Using Aliases to Obtain Multiple Statistics for the Same Variable ..	1869
Example 4: Displaying Multiple Statistics for One Variable	1872
Example 5: Consolidating Multiple Observations into One Row of a Report	1875
Example 6: Creating a Column for Each Value of a Variable	1878
Example 7: Writing a Customized Summary on Each Page	1882
Example 8: Displaying a Calculated Percentage Column in a Report	1887
Example 9: How PROC REPORT Handles Missing Values	1891
Example 10: Creating an Output Data Set and Storing Computed Variables	1895
Example 11: Using a Format to Create Groups	1899
Example 12: Using Multilabel Formats	1902
Example 13: Specifying Style Elements for ODS Output in Multiple Statements	1905
Example 14: Using the CELLWIDTH= Style Attribute with PROC REPORT	1913
Example 15: Using STYLE/MERGE in PROC REPORT CALL DEFINE Statement .	1916
Example 16: Using STYLE/REPLACE in PROC REPORT CALL DEFINE Statement	1918

Overview: REPORT Procedure

What Does the REPORT Procedure Do?

The REPORT procedure combines features of the PRINT, MEANS, and TABULATE procedures with features of the DATA step in a single report-writing tool that can produce a variety of reports. You can use PROC REPORT in the following ways:

- You submit a series of statements with the PROC REPORT statement, just as you do in other SAS procedures. You submit these statements from the Program Editor in the PROC REPORT statement.
- Create accessible output tables. When the ACCESSIBLETABLE system option is specified, changes are made to the layout of some tables to make them accessible. You can use the ACCESSIBLECHECK system option to check if your tables are accessible. When the ACCESSIBLETABLE system option is used and the CAPTION= option is used with PROC REPORT, captions are visible in the output.

For more information about creating accessible tables with PROC REPORT, see [Creating Accessible SAS Viya Platform Output Using ODS and ODS Graphics](#).

What Types of Reports Can PROC REPORT Produce?

A **detail report** contains one row for every observation selected for the report. Each of these rows is a report row, a **detail report row**. A **summary report** consolidates data so that each row represents multiple observations. Each of these rows is also called a detail row or a **summary report row**.

Both detail and summary reports can contain **summary report lines** (break lines) as well as report rows. A summary line summarizes numerical data for a set of detail rows or for all detail rows. PROC REPORT provides both default and customized summaries. See [“Using Break Lines” on page 1770](#).

This overview illustrates the types of reports that PROC REPORT can produce. The statements that create the data sets and formats used in these reports are in [“Example 1: Selecting Variables and Creating a Summary Line for a Report” on page 1862](#). The formats are stored in a permanent SAS library. See the REPORT procedure examples for more reports and for the statements that create them.

What Do the Various Types of Reports Look Like?

The data set that these reports use contains one day's sales figures for eight stores in a chain of grocery stores.

A simple PROC REPORT step produces a report similar to one produced by a simple PROC PRINT step. [Figure 51.33 on page 1754](#) illustrates the simplest type of report that you can produce with PROC REPORT. The statements that produce the report follow. The data set and formats that the program uses are created in [“Example 1: Selecting Variables and Creating a Summary Line for a Report” on page 1862](#). Although the WHERE and FORMAT statements are not essential, here they limit the amount of output and make the values easier to understand.

```
libname proclib 'SAS-library';

options nodate pageno=1 linesize=64 pagesize=60
      fmtsearch=(proclib);

proc report data=grocery;
  where sector='se';
  format sector $sctrfmt. manager $mgrfmt.
         dept $deptfmt. sales dollar10.2;
run;
```

Figure 51.1 Simple Detail Report with a Detail Row for Each Observation

The SAS System				Detail row
Sector	Manager	Department	Sales	1
Southeast	Smith	Paper	\$50.00	←
Southeast	Smith	Meat/Dairy	\$100.00	
Southeast	Smith	Canned	\$120.00	
Southeast	Smith	Produce	\$80.00	
Southeast	Jones	Paper	\$40.00	
Southeast	Jones	Meat/Dairy	\$300.00	
Southeast	Jones	Canned	\$220.00	
Southeast	Jones	Produce	\$70.00	

The report in the following figure uses the same observations as the above figure. However, the statements that produce this report

- order the rows by the values of Manager and Department.
- create a default summary line for each value of Manager.
- create a customized summary line for the whole report. A customized summary lets you control the content and appearance of the summary information, but you must write additional PROC REPORT statements to create one.

For an explanation of the program that produces this report, see [“Example 2: Ordering the Rows in a Report” on page 1866](#).

Figure 51.2 Ordered Detail Report with Default and Customized Summaries

Sales for the Southeast Sector			Detail row
Manager	Department	Sales	1

Jones	Paper	\$40.00	←
	Canned	\$220.00	
	Meat/Dairy	\$300.00	
	Produce	\$70.00	

Jones		\$630.00	←

Smith	Paper	\$50.00	
	Canned	\$120.00	
	Meat/Dairy	\$100.00	
	Produce	\$80.00	

Smith		\$350.00	

Total sales for these stores were: \$980.00			

Customized summary line for the whole report Default summary line for Manager

The summary report in the following figure contains one row for each store in the northern sector. Each detail row represents four observations in the input data set, one observation for each department. Information about individual departments does not appear in this report. Instead, the value of Sales in each detail row is the sum of the values of Sales in all four departments. In addition to consolidating multiple observations into one row of the report, the statements that create this report

- customize the text of the column headings
- create default summary lines that total the sales for each sector of the city
- create a customized summary line that totals the sales for both sectors

For an explanation of the program that produces this report, see [“Example 5: Consolidating Multiple Observations into One Row of a Report”](#) on page 1875.

Figure 51.3 Summary Report with Default and Customized Summaries

Sales Figures for Northern Sectors			Default summary line for Sector
Sector	Manager	Sales	
Northeast	Alomar	786.00	1
	Andrews	1,045.00	
		\$1,831.00	
Northwest	Brown	598.00	
	Pelfrey	746.00	
	Reveiz	1,110.00	
		\$2,454.00	
Combined sales for the northern sectors were \$4,285.00.			Customized summary line for the whole report

The summary report in the following figure is similar to the above figure. The major difference is that it also includes information for individual departments. Each selected value of Department forms a column in the report. In addition, the statements that create this report compute and display a variable that is not in the input data set.

For an explanation of the program that produces this report, see [“Example 6: Creating a Column for Each Value of a Variable”](#) on page 1878.

Figure 51.4 Summary Report with a Column for Each Value of a Variable

Sales Figures for Perishables in Northern Sectors				Computed variable
Sector	Manager	Department		1
		Meat/Dairy	Produce	Perishable Total
Northeast	Alomar	\$190.00	\$86.00	\$276.00
	Andrews	\$300.00	\$125.00	\$425.00
Northwest	Brown	\$250.00	\$73.00	\$323.00
	Pelfrey	\$205.00	\$76.00	\$281.00
	Reveiz	\$600.00	\$30.00	\$630.00
Combined sales for meat and dairy : \$1,545.00				
Combined sales for produce : \$390.00				
Combined sales for all perishables: \$1,935.00				

Customized summary lines for the whole report

The customized report in the following figure shows each manager's store on a separate page. Only the first two pages appear here. The statements that create this report create

- a customized heading for each page of the report
- a computed variable (Profit) that is not in the input data set
- a customized summary with text that is dependent on the total sales for that manager's store

For an explanation of the program that produces this report, see [“Example 7: Writing a Customized Summary on Each Page”](#) on page 1882.

Figure 51.5 Customized Summary Report

Detail row		Computed variable	
Sales for Individual Stores			1
Northeast Sector Store managed by Alomar			
Department	Sales	Profit	

Canned	\$420.00	\$168.00	
Meat/Dairy	\$190.00	\$47.50	
Paper	\$90.00	\$36.00	
Produce	\$86.00	\$21.50	

\$786.00		\$196.50	
Sales are in the target region.			
Customized summary line for Manager		Default summary line for Manager	

Detail row		Computed variable	
Sales for Individual Stores			2
Northeast Sector Store managed by Andrews			
Department	Sales	Profit	

Canned	\$420.00	\$168.00	
Meat/Dairy	\$300.00	\$75.00	
Paper	\$200.00	\$80.00	
Produce	\$125.00	\$31.25	

\$1,045.00		\$261.25	
Sales exceeded goal!!			
Customized summary line for Manager		Default summary line for Manager	

The report in the following figure uses customized style elements to control things like font faces, font sizes, and justification, as well as the width of the border of the table and the width of the spacing between cells. This report was created by using the HTML destination of the Output Delivery System (ODS) and the STYLE= option in several statements in the procedure.

For an explanation of the program that produces this report, see [“Example 13: Specifying Style Elements for ODS Output in Multiple Statements”](#) on page 1905. For information about ODS, see [“Output Delivery System”](#) on page 71.

Figure 51.6 HTML Output

Sales for the Southeast Sector		
Manager	Department	Sales
Jones	Paper	40
	Canned	220
	Meat/Dairy	300
	Produce	70
Jones		630
Subtotal for Jones is \$630.00.		
Smith	Paper	50
	Canned	120
	Meat/Dairy	100
	Produce	80
Smith		350
Subtotal for Smith is \$350.00.		
Total for all departments is: \$980.00.		

Concepts: REPORT Procedure

Laying Out a Report

Planning the Layout

Report writing is simplified if you approach it with a clear understanding of what you want the report to look like. The most important thing to determine is the

layout of the report. To design the layout, ask yourself the following types of questions:

- What do I want to display in each column of the report?
- In what order do I want the columns to appear?
- Do I want to display a column for each value of a particular variable?
- Do I want a row for every observation in the report, or do I want to consolidate information for multiple observations into one row?
- In what order do I want the rows to appear?

When you understand the layout of the report, use the COLUMN and DEFINE statements in PROC REPORT to construct the layout.

The COLUMN statement lists the items that appear in the columns of the report, describes the arrangement of the columns, and defines headings that span multiple columns. A report item can be

- a data set variable
- a statistic calculated by the procedure
- a variable that you compute from other items in the report

Omit the COLUMN statement if you want to include all variables in the input data set in the same order as they occur in the data set.

The DEFINE statement defines the characteristics of an item in the report. These characteristics include how PROC REPORT uses the item in the report, the text of the column heading, and the format to use to display values.

Usage of Variables in a Report

Much of a report's layout is determined by the usages that you specify for variables in the DEFINE statements. For data set variables, these usages are

DISPLAY ORDER ACROSS GROUP ANALYSIS

A report can contain variables that are not in the input data set. These variables must have a usage of COMPUTED.

Display Variables

A report that contains one or more display variables has a row for every observation in the input data set. Display variables do not affect the order of the rows in the report. If no order variables appear to the left of a display variable, then the order of the rows in the report reflects the order of the observations in the data set. By default, PROC REPORT treats all character variables as display variables. For an example, see [“Example 1: Selecting Variables and Creating a Summary Line for a Report” on page 1862](#).

Order Variables

A report that contains one or more order variables has a row for every observation in the input data set. If no display variable appears to the left of an order variable, then PROC REPORT orders the detail rows according to the ascending, formatted values of the order variable. You can change the default order with ORDER= and DESCENDING in the DEFINE statement.

If the report contains multiple order variables, then PROC REPORT establishes the order of the detail rows by sorting these variables from left to right in the report. PROC REPORT does not repeat the value of an order variable from one row to the next if the value does not change, unless an order variable to its left changes values. For an example, see [“Example 2: Ordering the Rows in a Report” on page 1866](#).

The order of observations is not inherently defined for DBMS tables. If you specify the ORDER=DATA option for input data in a DBMS table, the order of rows written to a database table from PROC REPORT is not likely to be preserved.

Group Variables

If a report contains one or more group variables, then PROC REPORT tries to consolidate into one row all observations from the data set that have a unique combination of formatted values for all group variables.

When PROC REPORT creates groups, it orders the detail rows by the ascending, formatted values of the group variable. You can change the default order with ORDER= and DESCENDING in the DEFINE statement.

If the report contains multiple group variables, then the REPORT procedure establishes the order of the detail rows by sorting these variables from left to right in the report. PROC REPORT does not repeat the values of a group variable from one row to the next if the value does not change, unless a group variable to its left changes values.

If you are familiar with procedures that use class variables, then you see that group variables are class variables that are used in the row dimension in PROC TABULATE.

Note: You cannot always create groups. PROC REPORT cannot consolidate observations into groups if the report contains any order variables or any display variables that do not have one or more statistics associated with them. See [the COLUMN statement on page 1803](#). A message in the SAS log explains why groups could not be created. Instead, PROC REPORT creates a detail report that displays group variables the same way as it displays order variables. Even when PROC REPORT creates a detail report, the variables that you define as group variables retain that usage in their definitions.

See [“Example 5: Consolidating Multiple Observations into One Row of a Report” on page 1875](#).

Analysis Variables

An analysis variable is a numeric variable that is used to calculate a statistic for all the observations represented by a cell of the report. (Across variables, in combination with group variables or order variables, determine which observations a cell represents.) You associate a statistic with an analysis variable in the variable's definition or in the COLUMN statement. By default, PROC REPORT uses numeric variables as analysis variables that are used to calculate the Sum statistic.

The value of an analysis variable depends on where it appears in the report:

- In a detail report, the value of an analysis variable in a detail row is the value of the statistic associated with that variable calculated for a single observation. Calculating a statistic for a single observation is not practical. However, using the variable as an analysis variable enables you to create summary lines for sets of observations or for all observations.
- In a summary report, the value displayed for an analysis variable is the value of the statistic that you specify calculated for the set of observations represented by that cell of the report.
- In a summary line for any report, the value of an analysis variable is the value of the statistic that you specify calculated for all observations represented by that cell of the summary line.

For more information, see the [“BREAK Statement” on page 1791](#) and [“RBREAK Statement” on page 1827](#) statements.

For examples, refer to [“Example 2: Ordering the Rows in a Report” on page 1866](#), [“Example 3: Using Aliases to Obtain Multiple Statistics for the Same Variable” on page 1869](#), [“Example 5: Consolidating Multiple Observations into One Row of a Report” on page 1875](#), and [“Example 6: Creating a Column for Each Value of a Variable” on page 1878](#).

Note: Be careful when you use SAS dates in reports that contain summary lines. SAS dates are numeric variables. Unless you explicitly define dates as some other type of variable (ORDER, GROUP, or DISPLAY), PROC REPORT summarizes them.

Across Variables

PROC REPORT creates a column for each value of an across variable. PROC REPORT orders the columns by the ascending, formatted values of the across variable. You can change the default order with ORDER= and DESCENDING in the DEFINE statement. If no other variable helps define the column, then PROC REPORT displays the N statistic (the number of observations in the input data set that belong to that cell of the report.) See [the COLUMN statement on page 1803](#).

Note: When a display variable and an across variable share a column, the report must also contain another variable that is not in the same column. When referring to columns created by an across variable, you must use the `_cn_` syntax.

If you are familiar with procedures that use class variables, then you see that across variables are like class variables that are used in the column dimension with PROC TABULATE. Generally, you use Across variables in conjunction with order or group variables. For an example, see [“Example 6: Creating a Column for Each Value of a Variable” on page 1878](#).

Computed Variables

Computed variables are variables that you define for the report. They are not in the input data set, and PROC REPORT does not add them to the input data set. However, computed variables are included in an output data set if you create one.

You add a computed variable by

- including the computed variable in the COLUMN statement
- defining the variable's usage as COMPUTED in the DEFINE statement
- computing the value of the variable in a compute block associated with the variable

For examples, refer to [“Example 6: Creating a Column for Each Value of a Variable” on page 1878](#), [“Example 8: Displaying a Calculated Percentage Column in a Report” on page 1887](#), and [“Example 10: Creating an Output Data Set and Storing Computed Variables” on page 1895](#).

Interactions of Position and Usage

The position and usage of each variable in the report determine the report's structure and content. PROC REPORT orders the rows of the report according to the formatted values of order and group variables, considered from left to right as specified in the COLUMN statement. Similarly, PROC REPORT orders columns for an across variable from left to right, according to the values of the variable.

Several items can collectively define the contents of a column in a report. For example, in the following figure, the values that appear in the third and fourth columns are collectively determined by Sales, an analysis variable, and by Department, an across variable. This arrangement is called stacking items in the report because each item generates a heading, and the headings are stacked one above the other.

```
options nodate pageno=1 fmtsearch=(proclib);
proc report data=grocery split='*';
  column sector manager department,sales perish;
  define sector / group format=$sctrfmt. 'Sector' '';
  define manager / group format=$mgrfmt. 'Manager* ';
  define department/ across format=$deptfmt. '_Department_';
```



```

define sales / analysis sum format=dollar11.2 ' ';
define perish / computed format=dollar11.2 'Perishable Total';
break after manager / skip;
compute perish;
    perish=_c3+_c4_;
endcomp;
title "Sales Figures for Perishables in Northern Sectors";
where sector contains 'n' and (department='p1' or department='p2');
run;
title;

```

Figure 51.7 Stacking Department and Sales

Sales Figures for Perishables in Northern Sectors				
Sector	Manager	Department		Perishable Total
		Meat/Dairy	Produce	
Northeast	Aloma r	\$190.00	\$86.00	\$276.00
	Andrews	\$300.00	\$125.00	\$425.00
Northwest	Brown	\$250.00	\$73.00	\$323.00
	Pelfrey	\$205.00	\$76.00	\$281.00
	Reveiz	\$600.00	\$30.00	\$630.00

When you use multiple items to define the contents of a column, at most one of the following can be in a column:

- a display variable with or without a statistic above or below it
- an analysis variable with or without a statistic above or below it
- an order variable
- a group variable
- a computed variable

More than one of these items in a column creates a conflict for PROC REPORT about which values to display.

The following table shows which report items can share a column.

Note: You cannot stack order variables with other report items.

Table 51.1 Report Items That Can Share Columns

	Display	Analysis	Order	Group	Computed	Across	Statistic
Display						X*	X
Analysis						X	X
Order							

	Display	Analysis	Order	Group	Computed	Across	Statistic
Group						X	
Computed variable						X	
Across	X*	X			X	X	X
Statistic	X	X				X	

*When a display variable and an across variable share a column, the report must also contain another variable that is not in the same column.

When a column is defined by stacked report items, PROC REPORT formats the values in the column by using the format that is specified for the lowest report item in the stack that does not have an ACROSS usage.

The following items can stand alone in a column:

- display variable
- analysis variable
- order variable
- group variable
- computed variable
- across variable
- N statistic

Note: The values in a column that is occupied only by an across variable are frequency counts.

Using Compute Blocks

What Is a Compute Block?

A **compute block** is one or more programming statements that appear between a COMPUTE statement and an ENDCOMP statement. Between these two statements, you can use other SAS statements (assignment, CALL DEFINE, and LINE statements) that customize your output. PROC REPORT processes a data set by reading the variables in the order in which they appear from left to right in the

COLUMN statement. The procedure builds the report one column and one row at a time, and COMPUTE statements are executed as the report is built.

You create a compute block with the COMPUTE statement. A COMPUTE statement can contain a number of arguments of the following types:

report-item

A report item can be a data set variable, a statistic, or a computed variable.

location

Specifies at what point in the process the compute block executes in relation to the value for *target*. The location can be at the top or bottom of the report; before or after a set of observations.

target

Controls when the compute block executes. You can specify a *target* if you specify a location (BEFORE or AFTER) for the COMPUTE statement. A target can be a group or order variable or a `_PAGE_` variable.

Note: When you use the COMPUTE statement, you do not have to use a corresponding BREAK or RBREAK statement. See [“Using Break Lines” on page 1770](#). Also see [“Example 2: Ordering the Rows in a Report” on page 1866](#), which uses COMPUTE AFTER but does not use the RBREAK statement. Use these statements only when you want to implement one or more BREAK statement or RBREAK statement options. See [“Example 7: Writing a Customized Summary on Each Page” on page 1882](#), which uses both COMPUTE AFTER MANAGER and BREAK AFTER MANAGER.)

For an in-depth look at Using a Compute Block, refer to [The REPORT Procedure: A Primer for the Compute Block](#).

The Purpose of Compute Blocks

A compute block that is associated with a report item can perform the following tasks:

- define a variable that appears in a column of the report but is not in the input data set.
- define display attributes for a report item. See [“CALL DEFINE Statement” on page 1800](#).
- define or change the value for a report item, such as showing the word “Total” on a summary line.

A compute block that is associated with a location can write a customized summary.

In addition, all compute blocks can use most SAS language elements to perform calculations. See [“The Contents of Compute Blocks” on page 1766](#). A PROC REPORT step can contain multiple compute blocks, but they cannot be nested.

The Contents of Compute Blocks

A compute block begins with a COMPUTE statement and ends with an ENDCOMP statement. Within a compute block, you can use these SAS language elements:

- %INCLUDE statement
- these DATA step statements:

ARRAY	END
array-reference	IF-THEN/ELSE
assignment	LENGTH
CALL	RETURN
CONTINUE	sum
DO (all forms)	END
- comments
- null statements
- macro variables and macro invocations
- all DATA step functions

Within a compute block, you can also use these PROC REPORT features:

- Compute blocks for a customized summary can contain one or more LINE statements, which place customized text and formatted values in the summary. See [“LINE Statement” on page 1825](#).
- Compute blocks for a report item can contain one or more CALL DEFINE statements, which set attributes like color and format each time a value for the item is placed in the report. See the [“CALL DEFINE Statement” on page 1800](#).
- Any compute block can reference the automatic variable _BREAK_. See [“The Automatic Variable _BREAK_” on page 1771](#).

Four Ways to Reference Report Items in a Compute Block

A compute block can reference any report item that forms a column in the report (whether the column is visible). You reference report items in a compute block in one of four ways:

- by name.
- by a compound name that identifies both the variable and the name of the statistic that you calculate with it. A compound name has this form

variable-name.statistic

- by an alias that you create in the COLUMN statement or in the DEFINITION window.
- by column number, in the form

'_Cn_'

where n is the number of the column (from left to right) in the report.

Note: The only time a column number is necessary is when a COMPUTED variable is sharing a column with an ACROSS variable.

Note: Even though the columns that you define with NOPRINT and NOZERO do not appear in the report, you must count them when you are referencing columns by number. See the discussion of “NOPRINT” on page 1817 and “NOZERO” on page 1818.

Note: Referencing variables that have missing values leads to missing values. If a compute block references a variable that has a missing value, then PROC REPORT displays that variable as a blank (for character variables) or as a period (for numeric variables).

The following table shows how to use each type of reference in a compute block.

Variable Type	Referenced By	Example
Group	Name*	Department
Order	Name*	Department
Computed	Name*	Department
Display	Name*	Department
Display sharing a column with a statistic	A compound name*	Sales.sum
Analysis	A compound name*	Sales.mean
Any type sharing a column with an across variable	Column number **	'_c3_'

*If the variable has an alias, then you must reference it with the alias.

**Even if the variable has an alias, you must reference it by column number.

Refer to [“Example 3: Using Aliases to Obtain Multiple Statistics for the Same Variable” on page 1869](#), which references analysis variables by their aliases;

“[Example 6: Creating a Column for Each Value of a Variable](#)” on page 1878, which references variables by column number; and “[Example 8: Displaying a Calculated Percentage Column in a Report](#)” on page 1887, which references group variables and computed variables by name.

Compute Block Processing

In general, compute blocks are executed in the following order:

- 1 COMPUTE report-item;
- 2 COMPUTE BEFORE;
- 3 COMPUTE BEFORE target;
- 4 COMPUTE BEFORE _PAGE_;
- 5 COMPUTE AFTER;
- 6 COMPUTE AFTER target;
- 7 COMPUTE AFTER _PAGE_;

The COMPUTE statement’s behavior is dependent on the arguments that you include. The following syntax and behavior explains the actions of the compute block based on specific arguments in the COMPUTE statement.

Note: PROC REPORT assigns values to the columns in a row of a report from left to right. Consequently, you cannot base the calculation of a computed variable on any variable that appears to its right in the report. For information about how PROC REPORT (in general) builds a report, see [Results: REPORT Procedure on page 1850](#).

COMPUTE *report-item*;

When you include the *report-item* argument, the compute block executes on every observation when that particular column is processed. In general, this statement is used for a specific *report-item* column so that you can calculate a value, change a value, change a format, apply style attributes, or create a temporary variable.

COMPUTE BEFORE;

With this syntax, the compute block is executed before the first detail row. The block is executed only once. Overall summary values for the analysis variables are available in the compute block. In addition, values of temporary variables that are created in this block are available to other compute blocks. Values for group or order variables are not available in this block. Text that is generated by LINE statements appears below the headers and above the first detail row in the report. CALL DEFINE statements set attributes on rows that are created by an RBREAK BEFORE statement.

COMPUTE BEFORE *target*;

When you use this syntax, the compute block is executed when the value of the target variable changes. In this syntax:

- *target* specifies either a group variable or an order variable.
- BEFORE is a value for *location*. This value specifies that the compute block is executed at the top (or beginning) of the section for a specific value of *target*.
- The value of the target variable for this specific section of the report is available to the compute block.
- The values of temporary variables created in this block are available to other compute blocks.
- Summary values for this specific *target* value are available to the compute block.
- CALL DEFINE statements set attributes on rows that are created by a BREAK BEFORE *target* statement.

COMPUTE BEFORE _PAGE_;

The compute block is executed once for each page. The page break can be generated either by the destination to which you send the output or by using the following BREAK statement syntax:

```
BREAK <location> <break-var> /PAGE;
```

Typically, this block outputs text by using a LINE statement. The text appears at the top of the page above the headers. However, it is still part of the table. Any variable that is listed in the LINE statement takes its value from the first detail row in the report.

COMPUTE AFTER;

The compute block is executed after the last detail row, and the block is executed only once (for each report). Overall summary values for the analysis variables are also available in the compute block. Values for group or order variables are not available in this block. Text that is generated by LINE statements appears after the last detail row in the report. This block is executed after each BY value if the PROC REPORT code contains a BY statement. CALL DEFINE statements set attributes on rows created by an RBREAK AFTER statement.

COMPUTE AFTER *target*;

With this syntax, the compute block is executed when the value of the target (break) variable changes. In this syntax:

- *target* specifies a variable that is defined as either a group variable or an order variable.
- AFTER is a value for *location*. This value specifies that the compute block is executed at the bottom (or end) of the section for a specific value of *target*.
- The value of the target variable for this specific section of the report is available to the compute block.
- Summary values for this specific *target* value are available to the compute block.
- CALL DEFINE statements set attributes on rows created by a BREAK AFTER *target* statement.

COMPUTE AFTER _PAGE_;

The compute block is executed once for each page. The page break can be generated either by the destination to which you send the output or by using the following BREAK statement syntax:

```
BREAK <location> <break-var> /PAGE;
```

Any variable that is listed in the LINE statement takes its value from the last detail row in the report. The text is placed at the end of the table, but it is still part of the table.

Note: The exact location of the text might change for each page, based on the amount of information that is output to each page. The text is placed on the last page of a table if it spans multiple pages.

Using Break Lines

What Are Break Lines?

Break lines are lines of text (including blanks) that appear at particular locations, called **breaks**, in a report. A report can contain multiple breaks. Generally, break lines are used to visually separate parts of a report, to summarize information, or both. They can occur in the following locations:

- at the beginning or end of a report
- at the top or bottom of each page
- between sets of observations (whenever the value of a group or order variable changes)

Break lines can contain the following items:

- text
- values calculated for either a set of rows or for the whole report

Creating Break Lines

There are two ways to create break lines. The first way is simpler. It produces a default summary. The second way is more flexible. It produces a customized summary and provides a way to slightly modify a default summary. Default summaries and customized summaries can appear at the same location in a report.

Default summaries are produced with the BREAK statement, the RBREAK statement. You can use default summaries to visually separate parts of the report, to summarize information for numeric variables, or both. Options provide some

control over the appearance of the break lines, but if you choose to summarize numeric variables, then you have no control over the content and the placement of the summary information. (A break line that summarizes information is a summary line.)

Customized summaries are produced in a compute block. You can control both the appearance and content of a customized summary, but you must write the code to do so.

Order of Break Lines

You control the order of the lines in a customized summary. However, PROC REPORT controls the order of lines in a default summary and the placement of a customized summary relative to a default summary. When a default summary contains multiple break lines, the order in which the break lines appear is as follows:

- 1 summary line
- 2 page break

The Automatic Variable `_BREAK_`

PROC REPORT automatically creates a variable called `_BREAK_`. This variable contains the following:

- a blank if the current line is not part of a break
- the value of the break variable if the current line is part of a break between sets of observations
- the value `_RBREAK_` if the current line is part of a break at the beginning or end of the report
- the value `_PAGE_` if the current line is part of a break at the beginning or end of a page

Using Compound Names

When you use a statistic in a report, you generally refer to it in compute blocks by a compound name like `Sales.sum`. However, in different parts of the report, that same name has different meanings. Consider the report in the following output. The statements that create the output follow. The user-defined formats that are used are created by a [PROC FORMAT step on page 1864](#).

```
libname proclib
  'SAS-library';
```

```

options nodate pageno=1 fmtsearch=(proclib);
proc report data=grocery;
  column sector manager sales;
  define sector / group format=$sctrfmt.;
  define sales / analysis sum
              format=dollar9.2;
  define manager / group format=$mgrfmt.;
  break after sector / summarize;
  rbreak after / summarize;
  compute after;
    sector='Total: ';
  endcomp;
run;

```

Example Code 51.1 Three Different Meanings of Sales.sum

The SAS System			
1			
	Sector	Manager	Sales
	Northeast	Alomar	\$786.00
		Andrews	\$1,045.00
	-----		-----
	Northeast		\$1,831.00
			2
	Northwest	Brown	\$598.00
		Pelfrey	\$746.00
		Reveiz	\$1,110.00
	-----		-----
	Northwest		\$2,454.00
	Southeast	Jones	\$630.00
		Smith	\$350.00
	-----		-----
	Southeast		\$980.00
	Southwest	Adams	\$695.00
		Taylor	\$353.00
	-----		-----
	Southwest		\$1,048.00
	=====		=====
	Total:		\$6,313.00
	=====		=====
			3

Here Sales.sum has three different meanings:

- 1 In detail rows, the value is the sales for one manager's store in a sector of the city. For example, the first detail row of the report shows that the sales for the store that Alomar manages were \$786.00.
- 2 In the group summary lines, the value is the sales for all the stores in one sector. For example, the first group summary line shows that sales for the Northeast sector were \$1,831.00.
- 3 In the report summary line, the value \$6,313.00 is the sales for all stores in the city.

Note: When you refer in a compute block to a statistic that has an alias, do not use a compound name. Generally, you must use the alias. However, if the statistic shares a column with an across variable, then you must reference it by column

number. See [“Four Ways to Reference Report Items in a Compute Block”](#) on page 1766.

ODS Destinations Supported by PROC REPORT

PROC REPORT supports all ODS destinations. For more information, see [“Understanding ODS Destinations”](#) in *SAS Output Delivery System: User’s Guide*.

The PROC REPORT STYLE= option supports all ODS destinations except OUTPUT. For more information, see [“Using ODS Styles with PROC REPORT”](#) on page 1835.

PROC REPORT supports the ODS DOCUMENT procedure. The DOCUMENT destination enables you to restructure, navigate, and replay your data in different ways and to different destinations without rerunning your analysis or repeating your database query. The DOCUMENT destination makes your entire output stream available in “raw” form and accessible to you to customize. The output is kept in the original internal representation as a data component plus a table definition. For more information, see [“DOCUMENT Procedure”](#) in *SAS Output Delivery System: Procedures Guide*.

PROC REPORT supports the OUTPUT destination for SAS output data sets. Because ODS already knows the logical structure of the data and its native form, ODS can output a SAS data set that represents exactly the same resulting data set that the procedure worked with internally. For more information, see [“ODS OUTPUT Statement”](#) in *SAS Output Delivery System: User’s Guide*.

Threaded Processing of Input Data Sets

The THREAD option enables or disables parallel processing of the input data set. Threaded processing achieves a degree of parallelism in the processing operations. This parallelism is intended to reduce the real time to completion for a given operation and therefore limit the cost of additional CPU resources.

The value of the SAS system option CPUCOUNT= affects the performance of the threaded sort. CPUCOUNT= suggests how many system CPUs are available for use by the threaded procedures.

For more information, see the [“THREADS System Option”](#) in *SAS System Options: Reference* and the [“CPUCOUNT= System Option”](#) in *SAS System Options: Reference*.

Calculated statistics can vary slightly, depending on the order in which observations are processed. Such variations are due to numerical errors that are introduced by floating-point arithmetic, the results of which should be considered approximate and not exact. The order of observation processing can be affected by nondeterministic effects of multithreaded or parallel processing. The order of processing can also be affected by inconsistent or nondeterministic ordering of observations that are produced by a data source, such as a DBMS that delivers

query results through an ACCESS engine. For more information, see [“Threading in Base SAS” in SAS Programmer’s Guide: Essentials](#)

Syntax: REPORT Procedure

Restriction:	PROC REPORT does not support VARCHAR data types.
Accessibility note:	You can use the ACCESSIBLECHECK and ACCESSIBLETABLE system options. ACCESSIBLETABLE changes the layout of some tables to make them accessible and adds visual captions to tables. ACCESSIBLECHECK checks to see if the table is accessible and outputs information to the SAS log. For more information, see “Creating Accessible Tables” in Creating Accessible SAS Viya Platform Output Using ODS and ODS Graphics. You can use the ACCESSIBLETABLE system option along with the CAPTION= option and the CONTENTS= option to see visible captions for tables and visible changes in the Table of Contents using By directives.
Accessibility note:	You can use the ACCESSIBLETABLE system option along with the CAPTION= option and the CONTENTS= option to see visible captions for tables and visible changes in the Table of Contents using By directives.
Tips:	Supports the Output Delivery System. For details, see “Output Delivery System: Basic Concepts” in SAS Output Delivery System: User’s Guide. You can use the ATTRIB, FORMAT, LABEL, and WHERE statements with the PROC SORT procedure. For in-database processing to occur, your data must reside within a supported version of a DBMS that has been properly configured for SAS in-database processing. For more information, see “In-Database Processing for PROC REPORT” on page 1846.

```

PROC REPORT<options>;
    ATTRIB variable-list(s) attribute-list(s);
    BREAK location break-variable < / options >
    BY < DESCENDING > variable-1 << DESCENDING > variable-2 ...> <
        NOTSORTED >;
    COLUMN column-specification(s);
    COMPUTE location <target>
        < / STYLE=<style-override(s)> >;
        LINE specification(s);
        ... select SAS language elements ...
    ENDCOMP;
    COMPUTE report-item < / type-specification >;
    CALL DEFINE (column-id', <' attribute-name', value >
        | _ROW_ <' attribute-name', value >);
        ... select SAS language elements ...

```

```

ENDCOMP;
DEFINE report-item / <options>;
FORMAT variable-1 <...variable-n> <format> < DEFAULT=default-format >;
FREQ variable;
LABEL variable-1=label-1<...variable-n=label-n>;
RBREAK location < / options >;
WEIGHT variable;
WHERE where-expression-1
  < logical-operator where-expression-n>;

```

Statement	Task	Example
PROC REPORT	Produce a summary or detail report	Ex. 1, Ex. 2, Ex. 6, Ex. 4, Ex. 9, Ex. 10, Ex. 13
BREAK	Produce a default summary at a change in the value of a group or order variable	Ex. 2, Ex. 5, Ex. 6, Ex. 7
BY	Create a separate report for each BY group	
CALL DEFINE	Set the value of an attribute for a particular column in the current row	Ex. 5, Ex. 13
COLUMN	Describe the arrangement of all columns and of headings that span more than one column	Ex. 1, Ex. 3, Ex. 6, Ex. 4, Ex. 8, Ex. 9
COMPUTE	Specify one or more programming statements that PROC REPORT executes as it builds the report	Ex. 2, Ex. 3, Ex. 5, Ex. 6, Ex. 7, Ex. 8, Ex. 10, Ex. 13
ENDCOMP	Specify one or more programming statements that PROC REPORT executes as it builds the report	Ex. 2
DEFINE	Describe how to use and display a report item	Ex. 2, Ex. 3, Ex. 5, Ex. 6, Ex. 4, Ex. 7, Ex. 8, Ex. 10, Ex. 10, Ex. 11, Ex. 13
FREQ	Treat observations as if they appear multiple times in the input data set	
LINE	Provide a subset of features of the PUT statement for writing customized summaries	Ex. 2, Ex. 3, Ex. 5, Ex. 6, Ex. 7

Statement	Task	Example
RBREAK	Produce a default summary at the beginning or end of a report or at the beginning and end of each BY group	Ex. 1 , Ex. 8
WEIGHT	Specify weights for analysis variables in the statistical calculations	

PROC REPORT Statement

Combines features of the PRINT, MEANS, and TABULATE procedures with features of the DATA step in a single report-writing tool that can produce a variety of reports.

Supports: Threaded processing

Examples: [“Example 1: Selecting Variables and Creating a Summary Line for a Report” on page 1862](#)
[“Example 2: Ordering the Rows in a Report” on page 1866](#)
[“Example 6: Creating a Column for Each Value of a Variable” on page 1878](#)
[“Example 4: Displaying Multiple Statistics for One Variable” on page 1872](#)
[“Example 9: How PROC REPORT Handles Missing Values” on page 1891](#)
[“Example 10: Creating an Output Data Set and Storing Computed Variables” on page 1895](#)
[“Example 13: Specifying Style Elements for ODS Output in Multiple Statements” on page 1905](#)
[“Example 14: Using the CELLWIDTH= Style Attribute with PROC REPORT” on page 1913](#)

Syntax

PROC REPORT *<options>*;

Summary of Optional Arguments

[CAPTION=](#)*text* [< #BYLINE >](#) [< #BYVAL >](#) [< #BYVAR >](#)

[DATA=](#)*SAS-data-set*

specifies the input data set.

[NOALIAS](#)

uses a report that was created before compute blocks required aliases.

[OUT=](#)*SAS-data-set*

specifies the output data set.

[THREADS](#)

NOTHEADS

overrides the SAS system option THREADS | NOTHEADS.

Control classification levels**COMPLETECOLS****NOCOMPLETECOLS**

creates all possible combinations of the across variable values.

COMPLETEROWS**NOCOMPLETEROWS**

creates all possible combinations of the group variable values.

Control ODS output**CONTENTS='link-text' < #BYLINE > < #BYVAL > < #BYVAR >**

specifies text for the table of contents entry for the output.

SPANROWS

specifies that a single cell occupies the column in all the rows for which the value is the same.

STYLE<(location(s))>=style-override(s)

specifies one or more style overrides to use for different parts of the report.

Control the layout of the report**BYPAGENO=number**

resets the page number between BY groups.

CENTER**NOCENTER**

specifies whether to center or left-justify the report and summary text.

MISSING

considers missing values as valid values for group, order, or across variables.

SHOWALL

overrides options in the DEFINE statement that suppress the display of a column.

Control the statistical analysis**EXCLNPWGT**

excludes observations with nonpositive weight values from the analysis.

QMARKERS=number

specifies the sample size to use for the P^2 quantile estimation method.

QMETHOD=OS | P2

specifies the quantile estimation method.

QNTLDEF=1 | 2 | 3 | 4 | 5

specifies the mathematical definition to calculate quantiles.

VARDEF=divisor

specifies the divisor to use in the calculation of variances.

Customize column headings**NAMED**

writes name= in front of each value in the report, where name= is the column heading for the value.

NOHEADER

suppresses column headings.

SPLIT='character'

specifies the split character.

Store and retrieve report definitions, PROC REPORT statements, and your report profile

LIST

writes to the SAS log the PROC REPORT code that creates the current report.

NOEXEC

suppresses the building of the report.

OUTREPT=libref.catalog.entry

stores in the specified catalog the report definition that is defined by the PROC REPORT step that you submit.

REPORT=libref.catalog.entry

specifies the report definition to use.

Optional Arguments

BYPAGENO=number

If a BY statement is present, specifies the page number at the start of each BY group.

Range any positive integer greater than 0.

Restriction This option has no effect if a BY statement is not present.

Interaction The BYPAGENO= option also affects the ODS ESCAPECHAR THISPAGE function.

CAPTION=text < #BYLINE > < #BYVAL > < #BYVAR >

specifies a caption text string to add before each table. If no string is specified, the caption defaults to the text specified by the CONTENTS= option.

This option makes table captions both visual and accessible if the ACCESSIBLETABLE system option is specified.

If the NOACCESSIBLETABLE system option is specified, then no caption is displayed, however the caption text is still accessible.

These options are available if a BY statement is in effect:

#BYLINE

substitutes the entire BY line without leading or trailing blanks for #BYLINE in the text string. The BY line uses the format *variable-name=value*.

#BYVALn

#BYVAL(BY-variable-name)

substitutes the current value of the specified BY variable for #BYVAL in the text string.

Specify the variable with one of the following:

n

specifies a variable by its position in the BY statement. For example, #BYVAL2 specifies the second variable in the BY statement.

BY-variable-name

specifies a variable from the BY statement by its name. For example, #BYVAL(YEAR) specifies the BY variable, YEAR. *Variable-name* is not case sensitive.

#BYVARn

#BYVAR(*BY-variable-name*)

substitutes the name of the BY-variable or the label associated with the variable (whatever the BY line would normally display) for #BYVAR in the text string.

Specify the variable with one of the following:

n

specifies a variable by its position in the BY statement. For example, #BYVAR2 specifies the second variable in the BY statement.

BY-variable-name

specifies a variable from the BY statement by its name. For example, #BYVAR(SITES) specifies the BY variable, SITES. *Variable-name* is not case sensitive.

Restriction The ACCESSIBLETABLE system option must be specified to enable captions.

Tip You can use the PROC DOCUMENT OBBNOTE option to display or edit the caption.

See [“Creating Accessible Tables” in *Creating Accessible SAS Viya Platform Output Using ODS and ODS Graphics*](#)

CENTER | NOCENTER

specifies whether to center or left-justify the report and summary text (customized break lines).

PROC REPORT honors the first of these centering specifications that it finds:

- the CENTER or NOCENTER option in the PROC REPORT statement or the CENTER toggle in the ROPTIONS window
- the CENTER or NOCENTER option stored in the report definition that is loaded with REPORT= in the PROC REPORT statement
- the SAS system option CENTER or NOCENTER

Interaction When CENTER is in effect, PROC REPORT ignores spacing that precedes the leftmost variable in the report.

COMPLETECOLS | NOCOMPLETECOLS

creates all possible combinations for the values of the across variables even if one or more of the combinations do not occur within the input data set.

Consequently, the column headings are the same for all logical pages of the report within a single BY group.

Default COMPLETECOLS

Interaction The PRELOADFMT option in the DEFINE statement ensures that PROC REPORT uses all user-defined format ranges for the combinations of across variables, even when a frequency is zero.

COMPLETEROWS | NOCOMPLETEROWS

displays all possible combinations of the values of the group variables, even if one or more of the combinations do not occur in the input data set.

Consequently, the row headings are the same for all logical pages of the report within a single BY group.

Default NOCOMPLETEROWS

Interaction The PRELOADFMT option in the DEFINE statement ensures that PROC REPORT uses all user-defined format ranges for the combinations of group variables, even when a frequency is zero.

CONTENTS='link-text' < #BYLINE > < #BYVAL > < #BYVAR >

specifies the text for the entries in the table of contents created by default or by options settings in ODS destinations that support the STYLE= option.

If the ACCESSIBLETABLE system option is specified, you can use the By directives to create visual output in the Table of Contents.

These options are available if a BY statement is in effect:

#BYLINE

substitutes the entire BY line without leading or trailing blanks for #BYLINE in the text string. The BY line uses the format *variable-name=value*.

#BYVAL_{*n*}

#BYVAL(*BY-variable-name*)

substitutes the current value of the specified BY variable for #BYVAL in the text string.

Specify the variable with one of the following:

n

specifies a variable by its position in the BY statement. For example, #BYVAL2 specifies the second variable in the BY statement.

BY-variable-name

specifies a variable from the BY statement by its name. For example, #BYVAL(YEAR) specifies the BY variable, YEAR. *Variable-name* is not case sensitive.

#BYVAR_{*n*}

#BYVAR(*BY-variable-name*)

substitutes the name of the BY-variable or the label associated with the variable (whatever the BY line would normally display) for #BYVAR in the text string.

Specify the variable with one of the following:

n

specifies a variable by its position in the BY statement. For example, #BYVAR2 specifies the second variable in the BY statement.

BY-variable-name

specifies a variable from the BY statement by its name. For example, #BYVAR(SITES) specifies the BY variable, SITES. *Variable-name* is not case sensitive.

Restriction The ACCESSIBLETABLE system option must be specified to use the By Directives to create visible output text in the Table of Contents. You can use CONTENTS='link-text' without the ACCESSIBLETABLE option.

Tip All ODS destinations except OUTPUT support the STYLE= option.

See Creating Accessible Tables with the REPORT Procedure in [Creating Accessible SAS Viya Platform Output Using ODS and ODS Graphics](#).

DATA=SAS-data-set

specifies the input data set.

See [“Input Data Sets” on page 24](#)

EXCLNPWGT

excludes observations with nonpositive weight values (zero or negative) from the analysis. By default, PROC REPORT treats observations with negative weights like observations with zero weights and counts them in the total number of observations.

Alias EXCLNPWGTS

Requirement You must use a WEIGHT statement.

See [“WEIGHT Statement” on page 1829](#)

LIST

writes to the SAS log the PROC REPORT code that creates the current report.

This LIST might differ in these ways from the statements that you submit:

- It shows some defaults that you might not have specified.
- It omits some statements that are not specific to the REPORT procedure, whether you submit them with the PROC REPORT step or had previously submitted them. These statements include

BY FOOTNOTE FREQ TITLE WEIGHT WHERE

- It omits these PROC REPORT statement options:

LIST

OUT=

OUTREPT=

REPORT=

- It includes these style(<location>)= options:

CENTER RIGHT
 HEADER USAGE
 LEFT

- It omits SAS system options.
- It resolves automatic macro variables.

Restriction The LIST option does not support styles by columns if you specify style(column)= in the DEFINE statement.

MISSING

considers missing values as valid values for group, order, or across variables. Special missing values used to represent numeric values (the letters A through Z and the underscore (_) character) are each considered as a different value. A group for each missing value appears in the report. If you omit the MISSING option, then PROC REPORT does not include observations with a missing value for any group, order, or across variables in the report.

See The “Missing Values” in *SAS Programmer’s Guide: Essentials*.

Example “Example 9: How PROC REPORT Handles Missing Values” on page 1891

NAMED

writes *name*= in front of each value in the report, where *name* is the column heading for the value.

Interaction When you use the NAMED option, PROC REPORT automatically uses the NOHEADER option.

Tip Use NAMED in conjunction with the WRAP option to produce a report that wraps all columns for a single row of the report onto consecutive lines rather than placing columns of a wide report on separate pages.

NOALIAS

lets you use a report that was created before compute blocks required aliases. If you use NOALIAS, then you cannot use aliases in compute blocks.

NOEXEC

suppresses the building of the report. Use NOEXEC with OUTREPT= to store a report definition in a catalog entry. Use NOEXEC with LIST and REPORT= to display a listing of the specified report definition.

Alias NOEXECUTE

NOHEADER

suppresses column headings, including headings that span multiple columns.

OUT=SAS-data-set

names the output data set. If this data set does not exist, then PROC REPORT creates it. The data set contains one observation for each report row and one observation for each unique summary line. If you use both customized and default summaries at the same place in the report, then the output data set

contains only one observation because the two summaries differ only in how they present the data. Information about customization (underlining, color, text, and so on) is not data and is not saved in the output data set.

The output data set contains one variable for each column of the report. PROC REPORT tries to use the name of the report item as the name of the corresponding variable in the output data set. However, it cannot perform this substitution if a data set variable is under or over an across variable or if a data set variable appears multiple times in the COLUMN statement without aliases. In these cases, the name of the variable is based on the column number (_C1_, _C2_, and so on).

Output data set variables that are derived from input data set variables retain the formats of their counterparts in the input data set. PROC REPORT derives labels for these variables from the corresponding column headings in the report unless the only item defining the column is an across variable. In that case, the variables have no label. If multiple items are stacked in a column, then the labels of the corresponding output data set variables come from the analysis variable in the column.

The output data set also contains a character variable named `_BREAK_`. If an observation in the output data set derives from a detail row in the report, then the value of `_BREAK_` is missing or blank. If the observation derives from a summary line, then the value of `_BREAK_` is the name of the break variable that is associated with the summary line, or `_RBREAK_`. If the observation derives from a COMPUTE BEFORE `_PAGE_` or COMPUTE AFTER `_PAGE_` statement, then the value of `_BREAK_` is `_PAGE_`. Note, however, that for COMPUTE BEFORE `_PAGE_` and COMPUTE AFTER `_PAGE_`, the `_PAGE_` value is written only to the output data set; it is not available as a value of the automatic variable `_BREAK_` during execution of the procedure.

Tip An output data set can be created by using an ODS OUTPUT statement. The data set created by ODS OUTPUT is the same as the one created by the OUT= option. Refer to the “[ODS OUTPUT Statement](#)” in *SAS Output Delivery System: User's Guide*.

Examples “[Example 10: Creating an Output Data Set and Storing Computed Variables](#)” on page 1895

“[Example 10: Creating an Output Data Set and Storing Computed Variables](#)” on page 1895

OUTREPT=libref.catalog.entry

stores in the specified catalog entry the REPORT definition that is defined by the PROC REPORT step that you submit. PROC REPORT assigns the entry a type of REPT.

The stored report definition might differ in these ways from the statements that you submit:

- It omits some statements that are not specific to the REPORT procedure, whether you submit them with the PROC REPORT step or whether they are already in effect when you submit the step. These statements include

BY TITLE

FOOTNOTE WEIGHT
FREQ WHERE

- It includes these PROC REPORT statement options. Others are omitted.

CENTER PANELS
COLWIDTH PCTLDEF
COMPLETECOLS QMARKERS
COMPLETEROWS QMETHOD
MISSING SHOWALL
NAMED SPLIT
NOHEADER VARDEF
PAGESIZE

- It omits SAS system options.
- It resolves automatic macro variables.

QMARKERS=number

specifies the default number of markers to use for the P^2 estimation method. The number of markers controls the size of fixed memory space.

Default The default value depends on which quantiles you request. For the median (P50), *number* is 7. For the quartiles (P25 and P75), *number* is 25. For the quantiles P1, P5, P10, P90, P95, or P99, *number* is 105. If you request several quantiles, then PROC REPORT uses the largest default value of *number*.

Range any odd integer greater than 3

Tip Increase the number of markers above the default settings to improve the accuracy of the estimates; you can reduce the number of markers to conserve computing resources.

QMETHOD=OS | P2

specifies the method that PROC REPORT uses to process the input data when it computes quantiles. If the number of observations is less than or equal to the value of the QMARKERS= option, and the value of the QNTLDEF= option is 5, then both methods produce the same results.

OS

uses order statistics. PROC UNIVARIATE uses this technique.

Note: This technique can be very memory intensive.

P2

uses the P^2 method to approximate the quantile.

Default OS

Restriction When QMETHOD=P2, PROC REPORT does not compute MODE and weighted quantiles.

Tip When QMETHOD=P2, reliable estimates of some quantiles (P1, P5, P95, P99) might not be possible for some data sets such as data sets with heavily tailed or skewed distributions.

QNTLDEF=1 | 2 | 3 | 4 | 5

specifies the mathematical definition that the procedure uses to calculate quantiles when the value of the QMETHOD= option is OS. When QMETHOD=P2, you must use QNTLDEF=5.

Alias PCTLDEF=

Default 5

See [“Quantile and Related Statistics” on page 2432](#)

REPORT=libref.catalog.entry

specifies the report definition to use. PROC REPORT stores all report definitions as entries of type REPT in a SAS catalog.

Interaction If you use REPORT=, then you cannot use the COLUMN statement.

See [“OUTREPT=libref.catalog.entry” on page 1783](#)

SHOWALL

overrides options in the DEFINE statement that suppress the display of a column.

See NOPRINT and NOZERO in [“DEFINE Statement” on page 1810](#)

SPANROWS

specifies that when the value of a GROUP or ORDER column is the same in multiple rows, the value is displayed in a single cell that occupies that column in all the rows for which the value is the same. A box is essentially created for that part of the column, and no rows appear in that box.

The SPANROWS option also allows GROUP and ORDER variables values to repeat when the values break across pages. Only the PDF, PS, and TAGSETS.RTF destinations support this part of the feature.

Notes The SPANROWS option has no effect on the data sets or OUTPUT destinations.

When the LINE statement occurs at the bottom of a page, the GROUP and ORDER variables values do not repeat when the values break across RTF and TAGSETS.RTF pages.

Tip If a summary row appears in the middle of a set of rows that would otherwise be spanned by a single cell, the summary row introduces its own cell in that column. This action breaks the spanning cell into two cells even when the value of the GROUP or ORDER variable that comes after the summary row is unchanged.

SPLIT='character'

specifies the split character. PROC REPORT breaks column text when it reaches that character and continues the text on the next line. The split character itself

is not part of the column heading or text value although each occurrence of the split character counts toward the 256-character maximum for a label.

Default slash (/)

Restriction This option works only in the column heading on ODS destinations.

STYLE<(location(s))>=style-override(s)

specifies one or more style overrides to use for different parts of the report.

location(s)

identifies the part of the report that the STYLE= option affects. The following table shows what parts of a report are affected by values of *location*.

The valid and default values for *location* vary by what statement the STYLE= option appears in. The following table shows valid and default *location* values for each statement. To specify more than one value of *location* in the same STYLE= option, separate each value with a space.

Table 51.2 Locations and Default Style Elements for Each Statement in PROC REPORT

Statement	Valid Location Values	Default Location Value	Part of Report Affected	Default Style Element
PROC REPORT	COLUMN HEADER HDR SUMMARY REPORT LINES CALLDEF	REPORT	Report as a whole	Table
BREAK	SUMMARY LINES	SUMMARY	Summary lines	DataEmphasis
CALL DEFINE	CALLDEF	CALLDEF	Cells identified by a CALL DEFINE statement	Data
COMPUTE	LINES	LINES	Lines generated by LINE statements	LineContent
DEFINE	COLUMN HEADER HDR	COLUMN HEADER	Column cells Column headings	COLUMN: Data HEADER: Header
RBREAK	SUMMARY	SUMMARY	Summary lines	DataEmphasis

Statement	Valid Location Values	Default Location Value	Part of Report Affected	Default Style Element
	LINES		Lines generated by LINE statements	

All names shown in the following table can be used in place of *location(s)* in the PROC statement. The DEFINE statement accepts column and header. The BREAK and RBREAK statement accept summary and lines.

Figure 51.8 PROC REPORT Locations and Corresponding Statements

report		
header	header	header
column	column	column
column	column	column
summary	summary	summary
lines		
column	column	column
column	column	column
summary	summary	summary
lines		
lines		

Specifications in a PROC REPORT statement other than the PROC REPORT location override the same specification in the PROC REPORT statement. However, any style attributes that you specify in the PROC REPORT statement and do not override in another PROC REPORT statement are inherited. For example, if you specify a blue background and a white foreground for all column headings in the PROC REPORT statement, and you specify a gray background for the column headings of a variable in the PROC REPORT DEFINE statement, then the background for that particular column heading is gray, and the foreground is white (as specified in the PROC REPORT statement).

style-override

specifies one or more style attributes or style elements to override the default style element and attributes in a specific area of a report. You can specify a style override in two ways:

- Specify a style element. A style element is a collection of style attributes that apply to a particular part of the output for a SAS program.

- Specify a style attribute. A style attribute is a name-value pair that describes a single behavioral or visual aspect of a piece of output. This is the most specific method of changing the appearance of your output.

Note: These style overrides take precedence over those specified in the PROC statement.

style-override has the following form:

```
style-element-name | [style-attribute-name-1=style-attribute-value-1
<style-attribute-name-2=style-attribute-value-2 ...>]
```

Note: You can use braces ({ and }) instead of square brackets ([and]).

style-element-name

is the name of a style element that is part of an ODS style template. SAS provides some [style templates](#). Users can create their own style templates with the TEMPLATE procedure. See [SAS Output Delivery System: Procedures Guide](#)

See For information about using styles with PROC REPORT, see [“Using ODS Styles with PROC REPORT” on page 1835](#).

For a table of default style attributes and style elements for each ODS destination, see [“Style Elements and Style Attributes for Table Regions ” on page 1842](#).

style-attribute-name

specifies the attribute to change. For a list of the commonly used style attributes that you can set with the STYLE= option in PROC PRINT, PROC TABULATE, and PROC REPORT, see [Table 51.99 on page 1839](#).

See For information about using styles with PROC REPORT, see [“Using ODS Styles with PROC REPORT” on page 1835](#).

For a table of default style attributes and style elements for each ODS destination, see [“Style Elements and Style Attributes for Table Regions ” on page 1842](#).

style-attribute-value

specifies a value for the attribute. Each attribute has a different set of valid values. A SAS format can also be used as an attribute value for conditional formatting.

See For information about using styles with PROC REPORT, see [“Using ODS Styles with PROC REPORT” on page 1835](#).

For information about using SAS formats as style attribute values, see [“Using a Format to Assign a Style Attribute Value” on page 1845](#).

For a table of default style attributes and style elements for each ODS destination, see [“Style Elements and Style Attributes for Table Regions ” on page 1842.](#)

Restriction All ODS destinations except OUTPUT support the STYLE= option.

Tip FONT names that contain characters other than letters or underscores must be enclosed in quotation marks.

See [“Style Elements and Style Attributes for Table Regions ” on page 1842](#) for details.

Example [“Example 13: Specifying Style Elements for ODS Output in Multiple Statements” on page 1905](#)

THREADS | NOTHEADS

enables or disables parallel processing of the input data set. This option overrides the SAS system option THREADS | NOTHEADS unless the system option is restricted. For more information, see [“Threading in Base SAS” in SAS Programmer’s Guide: Essentials](#) and [“THREADS System Option” in SAS System Options: Reference](#).

Default value of SAS system option THREADS | NOTHEADS.

Restriction Your site administrator can create a restricted options table. A restricted options table specifies SAS system option values that are established at start-up and cannot be overridden. If the THREADS | NOTHEADS system option is listed in the restricted options table, any attempt to set these system options is ignored and a warning message is written to the SAS log.

Interaction PROC REPORT uses the value of the SAS system option THREADS except when a BY statement is specified or the value of the SAS system option CPUCOUNT is less than 2. You can specify the THREADS option in the PROC REPORT statement to force PROC REPORT to use parallel processing in these situations.

Note When threaded processing, also known as parallel processing, is in effect, observations might be returned in an unpredictable order. However, the observations are sorted correctly when a BY statement is specified.

VARDEF=divisor

specifies the divisor to use in the calculation of the variance and standard deviation.

DF

Degrees of freedom

The following is the formula for the divisor:

$$n - 1$$

N

Number of observations

The following is the formula for the divisor:

$$n$$

WDF

Sum of weights minus one

The following is the formula for the divisor:

$$(\sum_i w_i) - 1$$

WEIGHT

Sum of weights

The following is the formula for the divisor:

$$\sum_i w_i$$

Alias WGT

The procedure computes the variance as $CSS/divisor$, where CSS is the corrected sums of squares and equals $\sum (x_i - \bar{x})^2$. When you weight the analysis variables, CSS equals $\sum w_i(x_i - \bar{x}_w)^2$, where $\bar{x}_w$ is the weighted mean.

Default	DF
Requirement	To compute the standard error of the mean and Student's t -test, use the default value of VARDEF=.
Tips	When you use the WEIGHT statement and VARDEF=DF, the variance is an estimate of σ^2, where the variance of the ith observation is $var(x_i) = \sigma^2/w_i$ and w_i is the weight for the ith observation. This yields an estimate of the variance of an observation with unit weight. When you use the WEIGHT statement and VARDEF=WGT, the computed variance is asymptotically (for large n) an estimate of $\sigma^2/\bar{w}$, where $\bar{w}$ is the average weight. This yields an asymptotic estimate of the variance of an observation with average weight.

ATTRIB Statement

Associates a format, informat, label, and length with one or more variables.

Syntax

ATTRIB *variable-list(s) attribute-list(s);*

Arguments

variable-list(s)

names the variables that you want to associate with the attributes.

TIP List the variables in any form that SAS allows.

attribute-list(s)

specifies one or more attributes to assign to *variable-list*. Specify one or more of these attributes in the ATTRIB statement:

FORMAT=*format*

associates a format with variables in *variable-list*.

Tip The format can be either a standard SAS format or a format that is defined with the FORMAT procedure.

INFORMAT=*informat*

associates an informat with variables in *variable-list*.

Tip The informat can be either a standard SAS informat or an informat that is defined with the FORMAT procedure.

LABEL=*'label'*

associates a label with variables in *variable-list*.

See Also

[“ATTRIB Statement” in SAS DATA Step Statements: Reference.](#)

BREAK Statement

Produces a default summary at a break (a change in the value of a group or order variable). The information in a summary applies to a set of observations. The observations share a unique combination of values for the break variable and all other group or order variables to the left of the break variable in the report.

Examples:

[“Example 2: Ordering the Rows in a Report” on page 1866](#)

[“Example 5: Consolidating Multiple Observations into One Row of a Report” on page 1875](#)

[“Example 6: Creating a Column for Each Value of a Variable” on page 1878](#)

[“Example 7: Writing a Customized Summary on Each Page” on page 1882](#)

Syntax

BREAK *location break-variable* < / *options* >;

Summary of Optional Arguments

CONTENTS=*'link-text'*

specifies the link text used in the table of contents.

PAGE

starts a new page after the last break line.

STYLE<*location(s)*>=<*style-override*>

specifies a style override to use for default summary lines, customized summary lines or both.

SUMMARIZE

writes a summary line in each group of break lines.

SUPPRESS

suppresses the printing of the value of the break variable in the summary line and of any underlining or overlining in the break lines in the column containing the break variable.

Required Arguments

location

controls the placement of the break lines and is either

AFTER

places the break lines immediately after the last row of each set of rows that have the same value for the break variable.

BEFORE

places the break lines immediately before the first row of each set of rows that have the same value for the break variable.

break-variable

is a group or order variable. The REPORT procedure writes break lines each time the value of this variable changes.

Optional Arguments

CONTENTS=*'link-text'*

specifies the text for the entries in the table of contents created by default or by options settings in ODS destinations that support the STYLE= option. If the PAGE= option and the CONTENTS= option with *link-text* is specified, PROC REPORT uses the value of *link-text* as a link for tables created in the table of contents.

Default

If the BREAK AFTER statement does not have a CONTENTS= option specified, but does have the PAGE option specified, then the

default link text in the table of contents is “Table N” where N is an integer.

Restriction If CONTENTS= is specified, but no PAGE option is specified, then PROC REPORT generates a warning message in the SAS log file.

Interactions If the DEFINE statement has a page option and there is a BREAK BEFORE statement with a PAGE option and the CONTENTS= option has a value other than empty quotation marks specified, then PROC REPORT adds a directory to the table of contents and puts links to the tables in that directory. For more information about this interaction, see the CONTENTS= option in the [DEFINE statement on page 1810](#).

If there is a BREAK BEFORE statement specified and a CONTENTS='' option and a PAGE= option specified, then PROC REPORT does not create a directory in the table of contents. Instead, PROC REPORT uses the CONTENTS= value from the DEFINE statement to create links to the table of contents. If there is no CONTENTS= option in the DEFINE statement, then PROC REPORT creates links using the default text described in the DEFINE statement. Refer to the [DEFINE statement on page 1810](#) CONTENTS= option for an explanation of the default text information.

For RTF output, the CONTENTS= option has no effect on the RTF body file unless you turn on the CONTENTS=YES option in the ODS RTF statement. In that case, a Table of Contents page is inserted at the front of your RTF output file. Your CONTENTS= option text from PROC REPORT then shows up in this separate Table of Contents page.

Note If a BREAK BEFORE statement is present and the PAGE option is specified but no CONTENTS= option is specified, then the default link text is the location variable plus the value of the location variable. The location variable is associated with the BREAK variable. The value is the BREAK variable value. As shown in the following code, the value is rep and the location is before rep. break before rep / summarize page;

Tips If the CONTENTS= option is specified where the value is empty quotation marks, then no table link is created in the table of contents. An example of this code is CONTENTS= ' '

If there are multiple BREAK BEFORE statements, then the link text is the concatenation of all of the CONTENTS= values or of all the default values.

PAGE

Starts a new page after the last break line. In the ODS destinations that support the STYLE= option, the PAGE option starts a new table. All ODS destinations except OUTPUT supports the STYLE= option.

Restriction In the OUTPUT destination, this option has no effect.

Interaction If you use PAGE in the BREAK statement and you create a break at the end of the report, then the summary for the whole report appears on a separate page.

Example [“Example 7: Writing a Customized Summary on Each Page” on page 1882](#)

STYLE<location(s)>=<style-override>

specifies the style override to use for default summary lines that are created with the BREAK statement.

You can specify a style override in two ways:

- Specify a style element. A style element is a collection of style attributes that apply to a particular part of the output for a SAS program.
- Specify a style attribute. A style attribute is a name-value pair that describes a single behavioral or visual aspect of a piece of output. This is the most specific method of changing the appearance of your output.

style-override has the following form:

```
style-element-name | [style-attribute-name-1=style-attribute-value-1  
  <style-attribute-name-2=style-attribute-value-2 ...>]
```

Restriction All ODS destinations except OUTPUT support the STYLE= option.

Tip FONT names that contain characters other than letters or underscores must be enclosed in quotation marks.

See [“Style Elements and Style Attributes for Table Regions ” on page 1842](#)

SUMMARIZE

writes a summary line in each group of break lines. A summary line for a set of observations contains values for the following:

- the break variable (which you can suppress with the SUPPRESS option)
- other group or order variables to the left of the break variable
- statistics
- analysis variables
- computed variables

The following table shows how PROC REPORT calculates the value for each type of report item in a summary line that is created by the BREAK statement:

Report Item	Value
Break variable	Current value of the variable (or a missing value if you use SUPPRESS)
A group or order variable to the left of the break variable	Current value of the variable

Report Item	Value
A group or order variable to the right of the break variable, or a display variable anywhere in the report	Missing [*]
A statistic	Value of the statistic over all observations in the set
An analysis variable	Value of the statistic specified as the usage option in the item's definition. PROC REPORT calculates the value of the statistic over all observations in the set. The default usage is SUM.
A computed variable	Results of the calculations based on the code in the corresponding compute block. See "COMPUTE Statement" on page 1806 .
[*] If you reference a variable with a missing value in a customized summary line, then PROC REPORT displays that variable as a blank (for character variables) or a period (for numeric variables).	

Note: PROC REPORT cannot create groups in a report that contains order or display variables.

Accessibility note	If the SUPPRESS option and SUMMARIZE options are both specified, the summary row headers are not displayed and the table is not accessible.
Examples	"Example 2: Ordering the Rows in a Report" on page 1866 "Example 5: Consolidating Multiple Observations into One Row of a Report" on page 1875 "Example 7: Writing a Customized Summary on Each Page" on page 1882

SUPPRESS

suppresses printing of

- the value of the break variable in the summary line
- any underlining and overlining in the break lines in the column that contains the break variable

Interaction If you use SUPPRESS, then the value of the break variable is unavailable for use in customized break lines unless you assign a

value to it in the compute block that is associated with the break. See [“COMPUTE Statement” on page 1806](#).

Accessibility note If the SUPPRESS option and SUMMARIZE options are both specified, the summary row headers are not displayed and the table is not accessible.

Example [“Example 5: Consolidating Multiple Observations into One Row of a Report” on page 1875](#)

Details

Order of Break Lines

When a default summary contains more than one break line, the following is the order in which the break lines appear:

- 1 summary line (SUMMARIZE)
- 2 skipped line (SKIP)
- 3 page break (PAGE)

Note: If you define a customized summary for the break, then customized break lines appear after underlining or double underlining. For more information about customized break lines, see [“COMPUTE Statement” on page 1806](#) and [“LINE Statement” on page 1825](#).

BY Statement

Orders the output according to the BY groups.

- Restriction: Do not specify a group variable in the BY statement. The BY variable cannot be the same as the group variable. A *group variable* is a variable in the input data set that is used to categorize chart variable values into groups. For more information, see [“Group Variables” on page 1760](#).
- Interaction: If you use the RBREAK statement in a report that uses BY processing, then PROC REPORT creates a default summary for each BY group. In this case, you cannot summarize information for the whole report.
- Tip: Using the BY statement does not make the FIRST. and LAST. variables available in compute blocks.

Syntax

```
BY <DESCENDING> variable-1  
    <... <DESCENDING> variable-n>  
    <NOTSORTED>;
```

Required Argument

variable

specifies the variable that the procedure uses to form BY groups. You can specify more than one variable. If you do not use the NOTSORTED option in the BY statement, then either the observations in the data set must be sorted by all the variables that you specify, or they must be indexed appropriately. Variables in a BY statement are called *BY variables*.

Optional Arguments

DESCENDING

specifies that the observations are sorted in descending order by the variable that immediately follows the word DESCENDING in the BY statement.

NOTSORTED

specifies that observations are not necessarily sorted in alphabetic or numeric order. The observations are grouped in another way (for example, chronological order).

The requirement for ordering or indexing observations according to the values of BY variables is suspended for BY-group processing when you use the NOTSORTED option. In fact, the procedure does not use an index if you specify NOTSORTED. The procedure defines a BY group as a set of contiguous observations that have the same values for all BY variables. If observations with the same values for the BY variables are not contiguous, then the procedure treats each contiguous set as a separate BY group.

You cannot use the NOTSORTED option in a PROC SORT step.

Note: You cannot use the GROUPFORMAT option, which is available in the BY statement in a DATA step, in a BY statement in any PROC step.

Details

BY-Group Processing

Procedures create output for each BY group. For example, the elementary statistics procedures and the scoring procedures perform separate analyses for each BY group. The reporting and ODS procedures produce a report for each BY group.

Note: All Base SAS procedures except PROC PRINT process BY groups independently. PROC PRINT can report the number of observations in each BY group as well as the number of observations in all BY groups. Similarly, PROC PRINT can sum numeric variables in each BY group and across all BY groups.

You can use only one BY statement in each PROC step. When you use a BY statement, the procedure expects an input data set that is sorted by the order of the BY variables or one that has an appropriate index. If your input data set does not meet these criteria, then an error occurs. Either sort it with the SORT procedure or create an appropriate index on the BY variables.

Depending on the order of your data, you might need to use the NOTSORTED or DESCENDING option in the BY statement in the PROC step.

- For more information about the BY statement, see [SAS DATA Step Statements: Reference](#).
- For more information about PROC SORT, see [Chapter 55, “SORT Procedure,” on page 2039](#).
- For more information about BY-group processing in CAS, see [“Grouping and Ordering” in SAS Cloud Analytic Services: DATA Step Programming](#).
- For more information about creating indexes, see [“INDEX CREATE Statement” on page 440](#).

Formatting BY-Variable Values

When a procedure is submitted with a BY statement, the following actions are taken with respect to processing of BY groups:

- 1 The procedure determines whether the data is sorted by the internal (unformatted) values of the BY variable(s).
- 2 The procedure determines whether a format has been applied to the BY variable(s). If the BY variable is numeric and has no user-applied format, then the BEST12. format is applied for the purpose of BY-group processing.
- 3 The procedure continues adding observations to the current BY group until both the internal and formatted values of the BY variable or variables change.

This process can have unexpected results if, for example, nonconsecutive internal BY values share the same formatted value. In this case, the formatted value is represented in different BY groups. Alternatively, if different consecutive internal BY values share the same formatted value, then these observations are grouped into the same BY group.

BY Variables That Have Different Lengths in Two Data Sets

When a procedure has a BY statement and two input data sets, one of the input data sets is called the primary data set and the other is called the secondary data set. The primary data set is usually, but not always, the DATA= data set. A BY statement always applies to the primary data set. The variables in the BY statement must appear in the primary data set.

Each procedure determines whether a BY statement applies to a secondary data set, and performs one of the following actions:

- The procedure might always apply the BY statement to the secondary data set. In this case, one or more variables in the BY statement must appear in the secondary data set.
- The procedure might never apply the BY statement to the secondary data set. In this case, the variables in the BY statement are not required in the secondary data set.
- The procedure might check whether the BY variables are in the secondary data set. If none of the BY variables are in the secondary data set, then BY processing does not occur for the secondary data set. If one or more of the BY variables are in the secondary data set, and they match the BY variables in the primary data set, then BY processing is done for the secondary data set. If some but not all BY variables are in the secondary data set, then the procedure might issue an error message and quit. Or, it might take some other action described in the documentation for that particular procedure.

If the BY statement is applied to the secondary data set, then each BY variable that exists on both the data sets must have the same type, character or numeric, in both data sets. The BY variables are required to have either the same formatted value or the same unformatted value. Formatted values match only if both the formatted lengths and the formatted values are the same. Unformatted values are not required to have the same length in order to match. The unformatted character values match if the unformatted values are the same after stripping the trailing blanks. The unformatted doubles match if they have the same value.

A secondary data set does not need to have all of the BY variables that are in the primary data set. A procedure can define a subset of the BY variables for the secondary data set. For example, if the primary data set has the BY variables A,B,C,D, then the procedure can define the following BY variables on the secondary data set:

- A
- A,B
- A,B,C
- A,B,C,D

If both the primary and secondary data sets have the same number of BY variables, and all the BY variables have the same byte lengths and format lengths, then either the unformatted values or the formatted values in the BY buffer (for all of the BY variables) have to match. If they do not match, then each variable is compared. The formatted values of each variable are compared first. The formatted lengths have to match, and the formatted values have to match. If the formatted lengths and values do not match, then the unformatted values are compared even if the byte lengths are different.

If corresponding character variable lengths differ, then the longer character variable can contain only trailing blanks for the extra characters. If the lengths of the character variables are different, then the values match as long as they are the same after stripping the trailing blanks. For example, 'ABCD' in the primary data set matches 'ABCD ' in the secondary data set. If the secondary data set contained 'ABCDEF', then they would not match.

Base SAS Procedures That Support the BY Statement

COMPARE	SORT (required)
CORR	STANDARD
FREQ	SUMMARY
MEANS	TABULATE
PRINT	TRANSPOSE
RANK	UNIVARIATE
REPORT (nonwindowing environment only)	ODSTEXT
ODSLIST	ODSTABLE

Note: In the SORT procedure, the BY statement specifies how to sort the data. In the other procedures, the BY statement specifies how the data is currently sorted.

See Also

[“BY Statement” in SAS DATA Step Statements: Reference](#)

CALL DEFINE Statement

Sets the value of an attribute for a particular column in the current row. Only the FORMAT, URL, URLBP, and URLP attributes have an effect. The STYLE= and URL attributes are effective only when you are using ODS to create output.

Restriction: Valid only in a compute block that is attached to a report item.

Tip: All ODS destinations except OUTPUT support the STYLE= option.

Example:

```
compute sales;
    if sales.sum>100 and _break_=' ' then
        call define(_col_, "style",
            "style=[backgroundcolor=yellow
                fontfamily=helvetica
                fontweight=bold] ");
endcomp;

compute weight;
    if weight > 100.0 then
```

```
call define(_row_, "style/merge", "style={font_weight=bold}");
endcomp;
```

Examples:

[“Example 5: Consolidating Multiple Observations into One Row of a Report” on page 1875](#)

[“Example 13: Specifying Style Elements for ODS Output in Multiple Statements” on page 1905](#)

[“Example 15: Using STYLE/MERGE in PROC REPORT CALL DEFINE Statement” on page 1916](#)

[“Example 16: Using STYLE/REPLACE in PROC REPORT CALL DEFINE Statement” on page 1918](#)

Syntax

CALL DEFINE (*column-id* | *_ROW_*, ' *attribute-name*', *value*);

Required Arguments

column-id

specifies a column name or a column number (that is, the position of the column from the left edge of the report). A column ID can be one of the following:

- a character literal (in quotation marks) that is the column name
- a character expression that resolves to the column name
- a numeric literal that is the column number
- a numeric expression that resolves to the column number
- a name of the form '*_Cn_*', where *n* is the column number
- the automatic variable *_COL_*, which identifies the column that contains the report item that the compute block is attached to

ROW

is an automatic variable that indicates the entire current row.

attribute-name

is the attribute to define. For attribute names, refer to [Table 51.96 on page 1802](#).

Note: The attributes BLINK, HIGHLIGHT, and RVSVVIDEO do not work on all devices.

value

sets the value for the attribute. For values for each attribute, refer to [Table 51.96 on page 1802](#).

Table 51.3 Attribute Descriptions

Attribute	Description	Values	Affects
FORMAT=	Specifies a format for the column	A SAS format or a user-defined format	nonwindowing environments
STYLE=	Specifies the style override of the column or row	an ODS style attribute	All ODS destinations.
STYLE/ MERGE	Merge the style specified with the existing styles in the same row or column	an ODS style attribute	All ODS destinations.
STYLE/ REPLACE	Replace the existing style in the row or column	an ODS style attribute	All ODS destinations.
URL	Makes the contents of each cell of the column a link to the specified Uniform Resource Locator (URL)	A quoted URL (either single or double quotation marks can be used)	ODS HTML, RTF, PDF, PowerPoint, EPUB destinations
URLBP	Makes the contents of each cell of the column a link. The link points to a Uniform Resource Locator that is a concatenation of 1 the string that is specified by the BASE= option in the ODS HTML statement 2 the string that is specified by the PATH= option in the ODS HTML statement 3 the value of the URLBP attribute	A quoted URL (either single or double quotation marks can be used)	ODS HTML destination
URLP	Makes the contents of each cell of the column a link. The link points to a Uniform Resource Locator that is a concatenation of 1 the string that is specified by the PATH= option in the ODS HTML statement 2 the value of the URLP attribute	A quoted URL (either single or double quotation marks can be used)	ODS HTML destination

1 For information about the BASE= and PATH= options, see the documentation for the ODS HTML Statement in *SAS Output Delivery System: User's Guide*.

Details

Using Style Attributes with the CALL DEFINE Statement

The STYLE attribute specifies the style-override to use in the cells that are affected by the CALL DEFINE statement.

The STYLE= value functions like the STYLE= option in other statements in PROC REPORT. However, instead of acting as an option in a statement, it becomes the value for the STYLE attribute. For example, the following CALL DEFINE statement sets the background color to yellow and the font size to 7 for the specified column:

```
call define(_col_, "style",
           "style=[backgroundcolor=yellow fontsize=7]");
```

For information about style precedence, see [“Order of Precedence When Applying Style Attributes to Data Cells” on page 1844](#).

Restriction: All ODS destinations except OUTPUT support the STYLE= option.

Interaction: If you set a style override for the CALLDEF location in the PROC REPORT statement and you want to use that exact style override in a CALL DEFINE statement, use an empty string as the value for the STYLE attribute, as shown here:

```
call define (_col_, "STYLE", " " );
```

Tip: FONT names that contain characters other than letters or underscores must be enclosed in quotation marks.

Featured in: [“Example 13: Specifying Style Elements for ODS Output in Multiple Statements” on page 1905](#)

Using STYLE/REPLACE and STYLE/MERGE Style Attributes

The STYLE/MERGE and STYLE/REPLACE attributes work only with styles specified by the CALL DEFINE statement. You cannot merge or replace a style that is specified by a STYLE(COLUMN)= option. A good time to use STYLE/REPLACE and STYLE/MERGE is when you have two or more COMPUTE blocks that have a CALL DEFINE statement, and these CALL DEFINE statements refer to the same cell.

STYLE= and STYLE/REPLACE attributes specify the style element to be used for ODS. If a style already exists for a cell or row, these STYLE attributes tell CALL DEFINE to replace the style specified by the STYLE= option. For an example program, see [“Example 16: Using STYLE/REPLACE in PROC REPORT CALL DEFINE Statement” on page 1918](#).

The STYLE/MERGE attribute tells CALL DEFINE to merge the style specified by the STYLE= value with the existing style attributes that are in the same cell or row. If there is no previously existing STYLE= value to merge, STYLE/MERGE acts the same as the STYLE or STYLE/REPLACE attributes. For an example program, see [“Example 15: Using STYLE/MERGE in PROC REPORT CALL DEFINE Statement” on page 1916](#).

COLUMN Statement

Describes the arrangement of all columns and of headings that span more than one column.

Aliases: COL
COLS

COLUMNS

Restriction: You cannot use the COLUMN statement if you use REPORT= in the PROC REPORT statement.

Examples: [“Example 1: Selecting Variables and Creating a Summary Line for a Report” on page 1862](#)

[“Example 3: Using Aliases to Obtain Multiple Statistics for the Same Variable” on page 1869](#)

[“Example 6: Creating a Column for Each Value of a Variable” on page 1878](#)

[“Example 4: Displaying Multiple Statistics for One Variable” on page 1872](#)

[“Example 8: Displaying a Calculated Percentage Column in a Report” on page 1887](#)

[“Example 9: How PROC REPORT Handles Missing Values” on page 1891](#)

Syntax

COLUMN *column-specification(s)*;

Required Argument

column-specification(s)

is one or more of the following:

- *report-item(s)*
- *report-item-1, report-item-2 <... , report-item-n>*
- *('header-1' <... 'header-n'> report-item(s))*
- *report-item=name*

where *report-item* is the name of a data set variable, a computed variable, or a statistic. See [“Statistics That Are Available in PROC REPORT” on page 1833](#) for a list of available statistics.

report-item(s)

identifies items that each form a column in the report.

Examples [“Example 1: Selecting Variables and Creating a Summary Line for a Report” on page 1862](#)

[“Example 9: How PROC REPORT Handles Missing Values” on page 1891](#)

report-item-1, report-item-2 <... , report-item-n>

identifies report items that collectively determine the contents of the column or columns. These items are said to be stacked in the report because each item generates a heading, and the headings are stacked one above the other. The heading for the leftmost item is on top. If one of the items is an analysis variable, a computed variable, a group variable, or a statistic, then its values fill the cells in that part of the report. Otherwise, PROC REPORT fills the cells with frequency counts.

If you stack a statistic with an analysis variable, then the statistic that you name in the column statement overrides the statistic in the definition of the analysis variable. For example, the following PROC REPORT step produces a report that contains the minimum value of Sales for each sector:

```
proc report data=grocery;
column sector sales,min;
define sector/group;
define sales/analysis sum;
run;
```

If you stack a display variable under an across variable, then all the values of that display variable appear in the report.

Interaction A series of stacked report items can include only one analysis variable or statistic. If you include more than one analysis variable or statistic, then PROC REPORT returns an error because it cannot determine which values to put in the cells of the report.

Tip You can use parentheses to group report items whose headings should appear at the same level rather than stacked one above the other.

Examples [“Example 6: Creating a Column for Each Value of a Variable” on page 1878](#)

[“Example 4: Displaying Multiple Statistics for One Variable” on page 1872](#)

[“Example 8: Displaying a Calculated Percentage Column in a Report” on page 1887](#)

(*'header-1' <... 'header-n'> report-item(s)*)

creates one or more headings that span multiple columns.

header

is a string of characters that spans one or more columns in the report. PROC REPORT prints each heading on a separate line. You can use split characters in a heading to split one heading over multiple lines. See the discussion of [SPLIT=](#) on page 1785.

Note: PROC REPORT simply removes the expanding characters when the output is directed to any other destination. Refer to [“Understanding ODS Destinations” in SAS Output Delivery System: User's Guide](#) for more information.

report-item(s)

specifies the columns to span.

Example [“Example 8: Displaying a Calculated Percentage Column in a Report” on page 1887](#)

report-item=name

specifies an alias for a report item. You can use the same report item more than once in a COLUMN statement. However, you can use only one DEFINE statement for any given name. (The DEFINE statement designates characteristics such as formats and customized column headings. If you omit a DEFINE statement for an item, then the REPORT procedure uses defaults.) Assigning an alias in the COLUMN statement does not by itself alter the report. However, it does enable you to use separate DEFINE statements for each occurrence of a variable or statistic.

Note You cannot always use an alias. When you refer in a compute block to a report item that has an alias, you must use the alias. However, if the report item shares a column with an across variable, then you must reference the column by column number. See [“Four Ways to Reference Report Items in a Compute Block”](#) on page 1766.

Example [“Example 3: Using Aliases to Obtain Multiple Statistics for the Same Variable”](#) on page 1869

COMPUTE Statement

Starts a compute block containing one or more programming statements that PROC REPORT executes as it builds the report.

- Restriction:** If you are sending a report to multiple ODS destinations or to an ODS document that is replayed later, avoid the use of non-deterministic functions in a COMPUTE block (for example, LAG, DIF, RANUNI, DATETIME, and so on). If you need to use data created by such functions in your report, call the functions in a DATA step and store the results in the data set before running PROC REPORT.
- Interaction:** An ENDCOMP statement must mark the end of the group of statements in the compute block.
- Note:** A compute block can be associated with a report item or with a location (at the top or bottom of a report; at the top or bottom of a page; before or after a set of observations). One form of the COMPUTE statement associates the compute block with a report item. Another form associates the compute block with a location. For a list of the SAS language elements that you can use in compute blocks, see [“The Contents of Compute Blocks”](#) on page 1766.
- Tip:** For information about how to use compute blocks, see [The REPORT Procedure: A Primer for the Compute Block](#)
- Examples:** [“Example 2: Ordering the Rows in a Report”](#) on page 1866
[“Example 3: Using Aliases to Obtain Multiple Statistics for the Same Variable”](#) on page 1869
[“Example 5: Consolidating Multiple Observations into One Row of a Report”](#) on page 1875
[“Example 6: Creating a Column for Each Value of a Variable”](#) on page 1878
[“Example 7: Writing a Customized Summary on Each Page”](#) on page 1882

[“Example 8: Displaying a Calculated Percentage Column in a Report” on page 1887](#)

[“Example 10: Creating an Output Data Set and Storing Computed Variables” on page 1895](#)

[“Example 13: Specifying Style Elements for ODS Output in Multiple Statements” on page 1905](#)

Syntax

```

COMPUTE location <target>
  </STYLE=<style-override(s)> > >;
  LINE specification(s);
  ... select SAS language elements ...
ENDCOMP;

COMPUTE report-item </ type-specification >;
  CALL DEFINE (column-id, 'attribute-name', value);
  ... select SAS language elements ...
ENDCOMP;

```

Required Arguments

You must specify either a location or a report item in the COMPUTE statement.

location

determines where the compute block executes in relation to *target*.

AFTER

executes the compute block at a break in one of the following places:

- immediately after the last row of a set of rows that have the same value for the variable that you specify as *target* or, if there is a default summary on that variable, immediately after the creation of the preliminary summary line. See [Results: REPORT Procedure on page 1850](#).
- at the end of the report if you omit a target.

BEFORE

executes the compute block at a break in one of the following places:

- immediately before the first row of a set of rows that have the same value for the variable that you specify as *target* or, if there is a default summary on that variable, immediately after the creation of the preliminary summary line. See [Results: REPORT Procedure on page 1850](#).
- immediately before the first detail row if you omit a target.

Note

If a report contains more columns than fit on a printed page, PROC REPORT generates an additional page or pages to contain the remaining columns. In this case, when you specify `_PAGE_` as *target*, the COMPUTE block does NOT re-execute for each of these additional pages; the COMPUTE block re-executes only after all columns have been printed.

Examples [“Example 3: Using Aliases to Obtain Multiple Statistics for the Same Variable” on page 1869](#)

[“Example 7: Writing a Customized Summary on Each Page” on page 1882](#)

report-item

specifies a data set variable, a computed variable, or a statistic to associate the compute block with. You must include the report item in the COLUMN statement. If the item is a computed variable, then you must include a DEFINE statement for it.

Note The position of a computed variable is important. PROC REPORT assigns values to the columns in a row of a report from left to right. Consequently, you cannot base the calculation of a computed variable on any variable that appears to its right in the report.

Examples [“Example 5: Consolidating Multiple Observations into One Row of a Report” on page 1875](#)

[“Example 6: Creating a Column for Each Value of a Variable” on page 1878](#)

Optional Arguments

target

controls when the compute block executes. If you specify a location (BEFORE or AFTER) for the COMPUTE statement, then you can also specify *target*, which can be one of the following:

break-variable

is a group or order variable.

When you specify a break variable, PROC REPORT executes the statements in the compute block each time the value of the break variable changes.

PAGE </ justification>

justification controls the placement of text and values. It can be one of the following:

CENTER

centers each line that the compute block writes.

LEFT

left-justifies each line that the compute block writes.

RIGHT

right-justifies each line that the compute block writes.

Default **CENTER**

Example [“Example 7: Writing a Customized Summary on Each Page” on page 1882](#)

type-specification

specifies the type. (Optional) Also specifies the length of *report-item*. If the report item that is associated with a compute block is a computed variable, then PROC REPORT assumes that it is a numeric variable unless you use a type specification to specify that it is a character variable. A type specification has the form

CHARACTER < LENGTH=*length* >

where

CHARACTER

specifies that the computed variable is a character variable. If you do not specify a length, then the variable's length is 8.

Alias CHAR

Example [“Example 8: Displaying a Calculated Percentage Column in a Report” on page 1887](#)

LENGTH=*length*

specifies the length of a computed character variable.

Default 8

Range 1 to 200

Interaction If you specify a length, then you must use CHARACTER to indicate that the computed variable is a character variable.

Example [“Example 8: Displaying a Calculated Percentage Column in a Report” on page 1887](#)

Options that Follow a Slash Delimiter

STYLE<(location(s))>=<style-override(s)>

specifies the style to use for the text that is created by any LINE statements in this compute block.

You can specify a style override in two ways:

- Specify a style element. A style element is a collection of style attributes that apply to a particular part of the output for a SAS program.
- Specify a style attribute. A style attribute is a name-value pair that describes a single behavioral or visual aspect of a piece of output. This is the most specific method of changing the appearance of your output.

style-override has the following form:

style-element-name | [*style-attribute-name-1*=*style-attribute-value-1*
 <*style-attribute-name-2*=*style-attribute-value-2* ...>]

Alias S=

Restriction All ODS destinations except OUTPUT support the STYLE= option.

Tip	FONT names that contain characters other than letters or underscores must be enclosed in quotation marks.
See	“Style Elements and Style Attributes for Table Regions” on page 1842
Example	“Example 13: Specifying Style Elements for ODS Output in Multiple Statements” on page 1905

DEFINE Statement

Describes how to use and display a report item.

Restriction:	A weight cannot be applied to a <i>report-item</i> alias without also applying it to the report-item. The WEIGHT= option must appear in the DEFINE statement for the <i>report-item</i> .
Accessibility note:	When the DEFINE statement includes an ORDER or GROUP option, the SPANROWS option must also be included in the PROC REPORT statement to generate an accessible table.
Tip:	If you do not use a DEFINE statement, then PROC REPORT uses default characteristics.
Examples:	“Example 2: Ordering the Rows in a Report” on page 1866 “Example 3: Using Aliases to Obtain Multiple Statistics for the Same Variable” on page 1869 “Example 5: Consolidating Multiple Observations into One Row of a Report” on page 1875 “Example 6: Creating a Column for Each Value of a Variable” on page 1878 “Example 4: Displaying Multiple Statistics for One Variable” on page 1872 “Example 7: Writing a Customized Summary on Each Page” on page 1882 “Example 8: Displaying a Calculated Percentage Column in a Report” on page 1887 “Example 10: Creating an Output Data Set and Storing Computed Variables” on page 1895 “Example 11: Using a Format to Create Groups” on page 1899 “Example 13: Specifying Style Elements for ODS Output in Multiple Statements” on page 1905 “Example 12: Using Multilabel Formats” on page 1902 “Example 14: Using the CELLWIDTH= Style Attribute with PROC REPORT” on page 1913

Syntax

DEFINE *report-item* / <options>;

Summary of Optional Arguments

Control the placement of values and column headings

CENTER

centers the formatted values of the report item within the column width and center the column heading over the values.

column-header

defines the column heading for the report item.

LEFT

left-justifies the formatted values of the report item within the column width and left-justifies the column headings over the values.

RIGHT

right-justifies the formatted values of the report item within the column width and right-justifies the column headings over the values.

Customize the appearance of a report item

EXCLUSIVE

excludes all combinations of the item that are not found in the preloaded range of user-defined formats.

FORMAT=*format*

assigns a SAS or user-defined format to the item.

MISSING

considers missing values as valid values for the item.

MLF

enables PROC REPORT to use the format label or labels to create subgroup combinations that have multilabel formats.

ORDER=DATA | FORMATTED | FREQ | INTERNAL

orders the values of a group, order, or across variable according to the specified order.

PRELOADFMT

specifies that all formats are preloaded for the item.

statistic

associates a statistic with an analysis variable.

STYLE<(location(s))>=<style-overrides(s)>

specifies a style element (for the Output Delivery System) for the report item.

WEIGHT=*weight-variable*

specifies a numeric variable whose values weight the value of the analysis variable.

Specify how to use a report item

ACROSS

defines the item, which must be a data set variable, as an across variable.

ANALYSIS

defines the item, which must be a data set variable, as an analysis variable.

COMPUTED

defines the item as a computed variable.

DISPLAY

defines the item, which must be a data set variable, as a display variable.

GROUP

defines the item, which must be a data set variable, as a group variable.

ORDER

defines the item, which must be a data set variable, as an order variable.

Specify options for a report item**CONTENTS='link-text'**

creates a link in the table of contents.

DESCENDING

reverses the order in which PROC REPORT displays rows or values of a group, order, or across variable.

ID

specifies that the item that you are defining is an ID variable.

NOPRINT

suppresses the display of the report item.

NOZERO

suppresses the display of the report item if its values are all zero or missing.

PAGE

inserts a page break just before printing the first column containing values of the report item.

Required Argument***report-item***

specifies the name or alias (established in the COLUMN statement) of the data set variable, computed variable, or statistic to define. The following are types of names that can be used for *report-item*:

- a SAS identifier (determined by the VALIDVARNAME option)
- a name literal
- a numbered range list
- a name range list
- a special name list
- a name prefix list
- a statistic

Notes The names in variable range lists refer to variables in the input data set, not statistic names, or computed variable names. Use only one name for each DEFINE statement. That one name, however, can be a range list. Example syntax using a variable range list is: `DEFINE Var1-Var3/width=10 center "#Visit#Date";`

Do not specify a usage option in the definition of a statistic. The name of the statistic tells PROC REPORT how to use it.

See [“SAS Names” in SAS Programmer’s Guide: Essentials](#), [“Variable Lists” in SAS Programmer’s Guide: Essentials](#), and [“VALIDVARNAME= System Option” in SAS System Options: Reference](#).

Optional Arguments

ACROSS

defines *report-item*, which must be a data set variable, as an across variable. See [“Across Variables” on page 1761](#).

Example [“Example 6: Creating a Column for Each Value of a Variable” on page 1878](#)

ANALYSIS

defines *report-item*, which must be a numeric data set variable, as an analysis variable. See [“Analysis Variables” on page 1761](#).

By default, PROC REPORT calculates the Sum statistic for an analysis variable. Specify an alternate statistic with the *statistic* option in the DEFINE statement.

Note: Naming a statistic in the DEFINE statement implies the ANALYSIS option, so you never need to specify ANALYSIS. However, specifying ANALYSIS can make your code easier for novice users to understand.

Note: Special missing values show up as missing values when they are defined as ANALYSIS variables.

Examples [“Example 2: Ordering the Rows in a Report” on page 1866](#)

[“Example 3: Using Aliases to Obtain Multiple Statistics for the Same Variable” on page 1869](#)

[“Example 5: Consolidating Multiple Observations into One Row of a Report” on page 1875](#)

CENTER

centers the formatted values of the report item within the column width and centers the column heading over the values. This option has no effect on the CENTER option in the PROC REPORT statement, which centers the report on the page.

Restriction In ODS output, only the data is affected by this option.

column-header

defines the column heading for the report item. Enclose each heading in single or double quotation marks. When you specify multiple column headings, PROC REPORT uses a separate line for each one. The split character also splits a column heading over multiple lines.

The following table shows the default variables and statistics:

Item	Header
Variable without a label	Variable name
Variable with a label	Variable label
Statistic	Statistic name

Tip If you want to use names when labels exist, then submit the following SAS statement before invoking PROC REPORT: `options nolabel;`

See [SPLIT=](#) on page 1785

Examples [“Example 3: Using Aliases to Obtain Multiple Statistics for the Same Variable”](#) on page 1869

[“Example 5: Consolidating Multiple Observations into One Row of a Report”](#) on page 1875

[“Example 6: Creating a Column for Each Value of a Variable”](#) on page 1878

COMPUTED

defines the specified item as a computed variable. Computed variables are variables that you define for the report. They are not in the input data set, and PROC REPORT does not add them to the input data set.

Add a computed variable as follows:

- Include the computed variable in the COLUMN statement.
- Define the variable's usage as COMPUTED in the DEFINE statement.
- Compute the value of the variable in a compute block associated with the variable.

Examples [“Example 6: Creating a Column for Each Value of a Variable”](#) on page 1878

[“Example 8: Displaying a Calculated Percentage Column in a Report”](#) on page 1887

CONTENTS=*'link-text'*

specifies the text for the entries in the table of contents created by default or by options settings in ODS destinations that support the STYLE= option. If the DEFINE statement has the PAGE= option and the CONTENTS= option specified with a *link-text* value assigned, then PROC REPORT adds a directory to the table of contents and uses the value of *link-text* as a link for tables created in the table of contents.

Default If the DEFINE statement has a PAGE option, but does not have a CONTENTS= option specified, then a directory is created with the directory text as COLA-COLB. COLA is the name or alias of the

leftmost column and COLB is the name or alias of the rightmost column. If the table has only one column, then the directory text is the column name or alias.

Restriction If CONTENTS= is specified, but no PAGE option is specified, then PROC REPORT generates a warning message in the SAS log file.

Interactions If the DEFINE statement has a page option and there is a BREAK BEFORE statement with a PAGE option and the CONTENTS= option specified has a value other than empty quotation marks, then PROC REPORT adds a directory to the table of contents and puts links to the tables in that directory.

If the DEFINE statement has a PAGE option and there is a BREAK BEFORE statement with no PAGE option, then PROC REPORT does not create a directory in the table of contents. Instead, PROC REPORT uses the CONTENTS= value from the DEFINE statement to create links to the table of contents. If there is no CONTENTS= option in the DEFINE statement, then PROC REPORT creates links using the default text COLA–COLB. Refer to the Default explanation above.

If there is a BREAK BEFORE statement with a CONTENTS=' ' option specified and a PAGE option specified, then PROC REPORT does not create a directory in the table of contents. Instead, PROC REPORT uses the CONTENTS= value from the DEFINE statement to create links to the table of contents. If there is no CONTENTS= option in the DEFINE statement, then PROC REPORT creates links using the default text COLA–COLB. Refer to the Default explanation above.

Tips If the DEFINE statement specifies the CONTENTS= option where the value is empty quotation marks, then the directory to the table of contents is not added. An example of this code is as follows:
 CONTENTS= ' '

If there are multiple BREAK BEFORE statements, then the link text is the concatenation of all of the CONTENTS= values or of all the default values.

All ODS destinations except OUTPUT support the STYLE= option.

DESCENDING

reverses the order in which PROC REPORT displays rows or values of a group, order, or across variable.

Tip By default, PROC REPORT orders group, order, and across variables by their formatted values. Use the ORDER= option in the DEFINE statement to specify an alternate sort order.

DISPLAY

defines *report-item*, which must be a data set variable, as a display variable. See [“Display Variables” on page 1759](#).

EXCLUSIVE

excludes from the report and the output data set all combinations of the group variables and the across variables that are not found in the preloaded range of user-defined formats.

Requirement You must specify the PRELOADFMT option in the DEFINE statement in order to preload the variable formats.

FORMAT=format

assigns a SAS or user-defined format to the item. This format applies to *report-item* as PROC REPORT displays it; the format does not alter the format associated with a variable in the data set. For data set variables, PROC REPORT honors the first of these formats that it finds:

- the format that is assigned with FORMAT= in the DEFINE statement
- the format that is assigned in a FORMAT statement when you invoke PROC REPORT
- the format that is associated with the variable in the data set

If none of these formats is present, then PROC REPORT uses BESTw. for numeric variables and \$w. for character variables. The value of *w* is the default column width. For character variables in the input data set, the default column width is the variable's length.

Alias F=

Examples [“Example 2: Ordering the Rows in a Report” on page 1866](#)

[“Example 4: Displaying Multiple Statistics for One Variable” on page 1872](#)

GROUP

defines *report-item*, which must be a data set variable, as a group variable. See [“Group Variables” on page 1760](#).

Accessibility note When the DEFINE statement includes an ORDER or GROUP option, the SPANROWS option must also be included in the PROC REPORT statement to generate an accessible table.

Examples [“Example 5: Consolidating Multiple Observations into One Row of a Report” on page 1875](#)

[“Example 4: Displaying Multiple Statistics for One Variable” on page 1872](#)

[“Example 11: Using a Format to Create Groups” on page 1899](#)

ID

specifies that the item that you are defining is an ID variable. An ID variable and all columns to its left appear at the left of every page of a report. ID ensures that you can identify each row of the report when the report contains more columns than fits on one page.

LEFT

left-justifies the formatted values of the report item within the column width and left-justifies the column headings over the values. If the format width is the same as the width of the column, then the LEFT option has no effect on the placement of values.

Restriction In ODS output, only the data is affected by this option.

MISSING

considers missing values as valid values for the report item. Special missing values that represent numeric values (the letters A through Z and the underscore (_) character) are each considered as a separate value.

Default If you omit the MISSING option, then PROC REPORT excludes from the report and the output data sets all observations that have a missing value for any group, order, or across variable.

MLF

enables PROC REPORT to use the format label or labels for a given range or for overlapping ranges to create subgroup combinations that use multilabel formatting. These multilabel formats are used only with group and across variables.

MLF is supported on all ODS destinations and data sets.

Note: PROC REPORT supports assigning a numeric variable that has a multilabel format to a character variable.

Requirement Use PROC FORMAT and the MULTILABEL option in the VALUE statement to create a multilabel format.

Tips The MLF option has no effect unless the variable is associated with a multilabel format. If there is no MULTILABEL format associated with the column, then an additional FORMAT statement or FORMAT= option in the DEFINE statement is needed to associate an existing format or informat with one or more variables. If MULTILABEL format is already associated with the column (using any regular method to associate a format with the variable), then no additional FORMAT statement or FORMAT= option is needed.

If the MLF option is omitted, PROC REPORT uses the primary format labels to determine the subgroup combinations. The primary format labels correspond to the first external format value.

See [MULTILABEL option on page 913](#) in the VALUE statement of the FORMAT procedure.

Example [“Example 12: Using Multilabel Formats” on page 1902](#)

NOPRINT

suppresses the display of the report item. Use this option

- if you do not want to show the item in the report but you need to use its values to calculate other values that you use in the report.
- to establish the order of rows in the report.
- if you do not want to use the item as a column but want to have access to its values in summaries. See [“Example 7: Writing a Customized Summary on Each Page” on page 1882](#).

Interactions Even though the columns that you define with NOPRINT do not appear in the report, you must count them when you are referencing columns by number. See [“Four Ways to Reference Report Items in a Compute Block” on page 1766](#).

SHOWALL in the PROC REPORT statement overrides all occurrences of NOPRINT.

Examples [“Example 3: Using Aliases to Obtain Multiple Statistics for the Same Variable” on page 1869](#)

[“Example 7: Writing a Customized Summary on Each Page” on page 1882](#)

NOZERO

suppresses the display of the report item if its values are all zero or missing.

Interactions Even though the columns that you define with NOZERO do not appear in the report, you must count them when you are referencing columns by number. See [“Four Ways to Reference Report Items in a Compute Block” on page 1766](#).

SHOWALL in the PROC REPORT statement overrides all occurrences of NOZERO.

ORDER

defines *report-item*, which must be a data set variable, as an order variable. See [“Order Variables” on page 1760](#).

Accessibility note When the DEFINE statement includes an ORDER or GROUP option, the SPANROWS option must also be included in the PROC REPORT statement to generate an accessible table.

Example [“Example 2: Ordering the Rows in a Report” on page 1866](#)

ORDER=DATA | FORMATTED | FREQ | INTERNAL

orders the values of a group, order, or across variable according to the specified order, where

DATA

orders values according to their order in the input data set.

Note If you specify the ORDER=DATA option for input data in a DBMS table, the order of rows written to a database table from PROC REPORT is not likely to be preserved.

FORMATTED

orders values by their formatted (external) values. If no format has been assigned to a class variable, then the default format, BEST12., is used.

FREQ

orders values by ascending frequency count.

INTERNAL

orders values by their unformatted values, which yields the same order that PROC SORT would yield. This order is operating environment-dependent. This sort sequence is particularly useful for displaying dates chronologically.

Default FORMATTED

Interaction DESCENDING in the item's definition reverses the sort sequence for an item. By default, the order is ascending.

Note The default value for the ORDER= option in PROC REPORT is not the same as the default value in other SAS procedures. In other SAS procedures, the default is ORDER=INTERNAL. The default for the option in PROC REPORT might change in a future release to be consistent with other procedures. Therefore, in production jobs where it is important to order report items by their formatted values, specify ORDER=FORMATTED even though it is currently the default. Doing so ensures that PROC REPORT continues to produce the reports that you expect even if the default changes.

Example [“Example 2: Ordering the Rows in a Report” on page 1866](#)

PAGE

inserts a page break just before printing the first column containing values of the report item.

Restriction This option has no affect on the OUTPUT destination.

Tip The page break does not occur until all the rows in the report have been printed. Therefore, PROC REPORT prints all the rows for all the columns to the left of the PAGE column and then starts over at the top of the report and prints the PAGE column and the columns to the right.

PRELOADFMT

specifies that the format is preloaded for the variable.

Restriction PRELOADFMT applies only to group and across variables.

Requirement PRELOADFMT has no effect unless you specify either EXCLUSIVE or ORDER=DATA and you assign a format to the variable.

Interactions To limit the report to the combination of formatted variable values that are present in the input data set, use the EXCLUSIVE option in the DEFINE statement.

To include all ranges and values of the user-defined formats in the output, use the COMPLETEROWS option in the PROC REPORT statement.

Notes

When preloading a character format, there must be at least one representative raw value that produces the label. If a range does not have any raw values, the range is considered unreachable and an error is generated. For example, you cannot preload a multi-label character format with both Other= and Low-High ranges. Other is a special keyword that refers to values or ranges that remain after all labels have been defined. For character formats, missing values are included in the low range so that there are no raw values left to occupy the Other range.

If you do not specify NOCOMPLETECOLS when you define the across variables, then the report includes a column for every formatted variable. If you specify COMPLETEROWS when you define the group variables, then the report includes a row for every formatted value. Some combinations of rows and columns might not make sense when the report includes a column for every formatted value of the across variable and a row for every formatted value of the group variable.

Example

[“Example 12: Using Multilabel Formats” on page 1902](#)

RIGHT

right-justifies the formatted values of the specified item within the column width and right-justifies the column headings over the values. If the format width is the same as the width of the column, then RIGHT has no effect on the placement of values.

Restriction

In ODS output, only the data is affected by this option.

statistic

associates a statistic with an analysis variable. You must associate a statistic with every analysis variable in its definition. PROC REPORT uses the statistic that you specify to calculate values for the analysis variable for the observations that are represented by each cell of the report. You cannot use *statistic* in the definition of any other type of variable.

See [“Statistics That Are Available in PROC REPORT” on page 1833](#) for a list of available statistics.

Default

SUM

Note

PROC REPORT uses the name of the analysis variable as the default heading for the column. You can customize the column heading with the *column-header* option in the DEFINE statement.

Examples

[“Example 2: Ordering the Rows in a Report” on page 1866](#)

[“Example 3: Using Aliases to Obtain Multiple Statistics for the Same Variable” on page 1869](#)

[“Example 5: Consolidating Multiple Observations into One Row of a Report” on page 1875](#)

STYLE<(location(s))>=<style-overrides(s)>

specifies the style element to use for column headings and for text inside cells for this report item.

You can specify a style override in two ways:

- Specify a style element. A style element is a collection of style attributes that apply to a particular part of the output for a SAS program.
- Specify a style attribute. A style attribute is a name-value pair that describes a single behavioral or visual aspect of a piece of output. This is the most specific method of changing the appearance of your output.

style-override has the following form:

style-element-name | [*style-attribute-name-1*=*style-attribute-value-1*
<*style-attribute-name-2*=*style-attribute-value-2* ...>]

Restriction All ODS destinations except OUTPUT support the STYLE= option.

Tip FONT names that contain characters other than letters or underscores must be enclosed in quotation marks.

See [“Style Elements and Style Attributes for Table Regions ” on page 1842](#)

Example [“Example 13: Specifying Style Elements for ODS Output in Multiple Statements” on page 1905](#)

WEIGHT=weight-variable

specifies a numeric variable whose values weight the values of the analysis variable that is specified in the DEFINE statement. The variable value does not have to be an integer. The following table describes how PROC REPORT treats various values of the WEIGHT variable.

Weight Value	PROC REPORT Response
0	Counts the observation in the total number of observations
Less than 0	Converts the value to zero and counts the observation in the total number of observations
Missing	Excludes the observation

To exclude observations that contain negative and zero weights from the analysis, use the EXCLNPWGT option in the PROC REPORT statement. Note that most SAS/STAT procedures, such as PROC GLM, exclude negative and zero weights by default.

Alias	WGT=
Restrictions	to compute weighted quantiles, use QMETHOD=OS in the PROC REPORT statement. A weight cannot be applied to a <i>report-item</i> alias without also applying it to the <i>report-item</i>. The WEIGHT= option must appear in the DEFINE statement for the <i>report-item</i>.
Tips	When you use the WEIGHT= option, consider which value of the VARDEF= option in the PROC REPORT statement is appropriate. Use the WEIGHT= option in separate variable definitions in order to specify different weights for the variables.

ENDCOMP Statement

Marks the end of one or more programming statements that PROC REPORT executes as it builds the report.

Restriction:	A COMPUTE statement must precede the ENDCOMP statement.
See:	COMPUTE statement
Example:	“Example 2: Ordering the Rows in a Report” on page 1866

Syntax

ENDCOMP;

FORMAT Statement

Associates formats with variables.

Syntax

FORMAT *variable-1* <...*variable-n*> *format variable-1* <...*variable-n*> *format*;

Arguments

variable

names one or more variables for SAS to associate with a format. You must specify at least one *variable*.

Tip To disassociate a format from a variable, use the variable in a FORMAT statement without specifying a format in a DATA step or in PROC DATASETS. In a DATA step, place this FORMAT statement after the SET statement. See [“Removing a Format” in SAS DATA Step Statements: Reference](#). You can also use PROC DATASETS.

format

specifies the format that is listed for writing the values of the variables.

Tip Formats that are associated with variables by using a FORMAT statement behave like formats that are used with a colon modifier in a subsequent PUT statement. For more information about using a colon modifier, see [“PUT Statement: List” in SAS DATA Step Statements: Reference](#).

See [SAS Formats and Informats: Reference](#)

FREQ Statement

Treats observations as if they appear multiple times in the input data set.

Tip: The effects of the FREQ and WEIGHT statements are similar except when calculating degrees of freedom.

Syntax

FREQ *variable*;

Required Argument

variable

specifies a numeric variable whose value represents the frequency of the observation. If you use the FREQ statement, then the procedure assumes that each observation represents n observations, where n is the value of *variable*. If n is not an integer, then SAS truncates it. If n is less than 1 or is missing, then the procedure does not use that observation to calculate statistics.

Details

Frequency Information Is Not Saved

When you store a report definition, PROC REPORT does not store the FREQ statement.

LABEL Statement

Assigns descriptive labels to variables.

Syntax

LABEL *variable-1* = '*text-string*' <...*variable-n* = '*text-string*'>;

LABEL *variable-1* = '' <...*variable-n*=''>; | *variable-1* = <...*variable-n*= >;

Arguments

variable

specifies the variable that you want to label. You can specify labels for multiple variables in a single LABEL statement:

```
data test;
  L = 5;
  W = 10;
  label L = 'Length' W = 'Width';
run;
```

text-string

specifies a label of up to 256 bytes.

```
data test;
  L = 5;
  label L = 'Length';
run;
```

Restrictions If the label includes a semicolon (;) or an equal sign (=), you must enclose the label in either single or double quotation marks.

```
label N = 'Number = ';
```

If the label includes single quotation marks ('), you must enclose the label in double quotation marks.

```
label Name = "Owner's Last Name";
```

There is partial support for variable name substitution with the JAVAIMG device. When the label text for the variable exceeds the space allowed, the variable name is used instead by procedures that are labeling an axis or a legend title. With the exception of the SAS/GRAPH GPLOT procedure, the variable name is not used when the JAVAIMG device is specified.

Notes Quotation marks are not required unless the label includes a semicolon, quotation marks, or an equal sign.

The LABEL statement expects an equal sign (=) after the first variable. If any other character is found, an error occurs. The LABEL statement considers everything after the equal sign, following the first variable, as part of the label until the statement comes to another variable followed by an equal sign, regardless of quotation marks. In this example, the label for x is **this is x y 4 = this is y** because the next variable that is followed by an equal sign is z.

```
data test;
  x = 1;
  y = 1;
  z = 1;
  label x = 'this is x' y 4 = 'this is y' z = 'This is z';
run;
```

In this LABEL statement, the x variable has the same label:

```
label x = this is x y 4 = this is y z = This is z;
```

See [“Character Constants and Character Variables” in SAS Programmer’s Guide: Essentials.](#)

”

removes a label from a variable. Enclose a single blank space in quotation marks to remove an existing label. You can remove labels from multiple variables in a single LABEL statement:

```
data test;
  L=5; W=10;
  label L = ' ' W = ' ';
run;
```

Alternatively, you can remove a label by specifying nothing after the equal sign:

```
data test;
  L=5; W=10;
  label L = W = ;
run;
```

LINE Statement

Provides a subset of the features of the PUT statement for writing customized summaries.

Restrictions: This statement is valid only in a compute block that is associated with a location in the report.

You cannot use the LINE statement in conditional statements (IF-THEN, IF-THEN/ELSE, and SELECT) because it is not executed until PROC REPORT has executed all other statements in the compute block.

Accessibility note: Using the LINE statement causes an inaccessible table to be generated.

Examples:

- “Example 2: Ordering the Rows in a Report” on page 1866
- “Example 3: Using Aliases to Obtain Multiple Statistics for the Same Variable” on page 1869
- “Example 5: Consolidating Multiple Observations into One Row of a Report” on page 1875
- “Example 6: Creating a Column for Each Value of a Variable” on page 1878
- “Example 7: Writing a Customized Summary on Each Page” on page 1882

Syntax

LINE *specification(s)*;

Required Argument

specification(s)

can have one of the following forms. You can mix different forms of specifications in one LINE statement.

item item-format

specifies the item to display and the format to use to display it, where

item

is the name of a data set variable, a computed variable, or a statistic in the report. For information about referencing report items, see “Four Ways to Reference Report Items in a Compute Block” on page 1766.

item-format

is a SAS format or user-defined format. You must specify a format for each item.

Example “Example 2: Ordering the Rows in a Report” on page 1866

'character-string'

specifies a string of text to display. When the string is a blank and nothing else is in *specification(s)*, PROC REPORT prints a blank line.

Note: A hexadecimal value (such as 'D'x) that is specified within *character-string* is not resolved because it is specified within quotation marks. To resolve a hexadecimal value, use the %sysfunc(byte(num)) function, where num is the hexadecimal value. Be sure to enclose *character-string* in double quotation marks (") so that the macro function resolves.

Example “Example 2: Ordering the Rows in a Report” on page 1866

number-of-repetitions*'character-string'

specifies a character string and the number of times to repeat it.

Example “Example 3: Using Aliases to Obtain Multiple Statistics for the Same Variable” on page 1869

RBREAK Statement

Produces a default summary at the beginning or end of a report or at the beginning or end of each BY group.

Examples: [“Example 1: Selecting Variables and Creating a Summary Line for a Report” on page 1862](#)
 [“Example 8: Displaying a Calculated Percentage Column in a Report” on page 1887](#)

Syntax

RBREAK *location* < / *options* >;

Summary of Optional Arguments

CONTENTS=*'link-text'*

specifies the link text used in the table of contents.

PAGE

starts a new page after the last break line of a break located at the beginning of the report.

STYLE<(*location(s)*)>=*<style-overrides(s)>*

specifies a style element (for the Output Delivery System) for default summary lines, customized summary lines, or both.

SUMMARIZE

includes a summary line as one of the break lines.

Required Argument

location

controls the placement of the break lines and is either of the following:

AFTER

places the break lines at the end of the report.

BEFORE

places the break lines at the beginning of the report.

Optional Arguments

CONTENTS=*'link-text'*

specifies the text for the entries in the table of contents created by default or by options settings in ODS destinations that support the STYLE= option. Only the RBREAK BEFORE statement with the PAGE and SUMMARIZE options specified creates a table within the table of contents. If the CONTENTS= option plus the

PAGE and SUMMARIZE options are specified, then PROC REPORT uses the value of *link-text* and places that text in the table of contents for the tables that are created. If the value of CONTENTS= is empty quotation marks, then no link is created in the table of contents.

Default If an RBREAK BEFORE statement is present and the PAGE and SUMMARIZE options are specified, but no CONTENTS= option is specified, then the default link text in the table of contents will show **Summary**.

Restriction If CONTENTS= is specified, but no PAGE option is specified, then PROC REPORT generates a warning message in the SAS log file. Only RBREAK BEFORE / with the SUMMARIZE and PAGE options specified can actually create an entry in the table of contents.

Tips HTML output can now have additional anchor tags.

All ODS destinations except OUTPUT support the STYLE= option.

PAGE

starts a new page after the last break line of a break located at the beginning of the report. On RBREAK BEFORE, the PAGE option starts a new table.

Restriction This option has no affect on the OUTPUT destination.

STYLE<(location(s))>=<style-overrides(s)>

specifies the style element to use for default summary lines that are created with the RBREAK statement.

Restriction All ODS destinations except OUTPUT support the STYLE= option.

Tip FONT names that contain characters other than letters or underscores must be enclosed in quotation marks.

See [“Style Elements and Style Attributes for Table Regions” on page 1842](#)

SUMMARIZE

includes a summary line as one of the break lines. A summary line at the beginning or end of a report contains values for the following:

- statistics
- analysis variables
- computed variables

The following table shows how PROC REPORT calculates the value for each type of report item in a summary line created by the RBREAK statement:

Report Item	Resulting Value
Statistic	Value of the statistic over all observations in the set

Report Item	Resulting Value
Analysis variable	Value of the statistic specified as the usage option in the DEFINE statement. PROC REPORT calculates the value of the statistic over all observations in the set. The default usage is SUM.
Computed variable	Results of the calculations based on the code in the corresponding compute block. See “COMPUTE Statement” on page 1806 .

Examples [“Example 1: Selecting Variables and Creating a Summary Line for a Report” on page 1862](#)

[“Example 8: Displaying a Calculated Percentage Column in a Report” on page 1887](#)

Details

Order of Break Lines

When a default summary contains more than one break line, the order in which the break lines appear is

- 1 summary line (SUMMARIZE)
- 2 page break (PAGE)

For more information about customized break lines, see [the COMPUTE statement on page 1806](#) and the LINE statement [“LINE Statement” on page 1825](#).

WEIGHT Statement

Specifies weights for analysis variables in the statistical calculations.

Alias: WGT

See: For information about calculating weighted statistics and for an example that uses the WEIGHT statement, see [“Calculating Weighted Statistics” on page 2198](#).

Syntax

WEIGHT *variable*;

Required Argument

variable

specifies a numeric variable whose values weight the values of the analysis variables. The value of the variable does not have to be an integer.

Table 51.4 Variable Values and How PROC REPORT Responds

Weight Value	PROC REPORT Response
0	Counts the observation in the total number of observations
Less than 0	Converts the value to zero and counts the observation in the total number of observations
Missing	Excludes the observation

Restriction PROC REPORT will not compute MODE when a weight variable is active. Instead, try using PROC UNIVARIATE when MODE needs to be computed and a weight variable is active.

Tip When you use the WEIGHT statement, consider which value of the VARDEF= option is appropriate. See [VARDEF=](#) on page 1789 and the calculation of weighted statistics in “Keywords and Formulas” on page 2426 for more information.

Details

Weight Information Is Not Saved

When you store a report definition, PROC REPORT does not store the WEIGHT statement.

WHERE Statement

Subsets the input data set by specifying certain conditions that each observation must meet before it is available for processing.

Syntax

WHERE *where-expression-1*

< *logical-operator where-expression-n* >;

Arguments

where-expression

is an arithmetic or logical expression that generally consists of a sequence of operands and operators.

Tips The operands and operators described in the next several sections are also valid for the WHERE= data set option.

You can specify multiple where-expressions.

logical-operator

can be AND, AND NOT, OR, or OR NOT.

Example

Procedures That Support the WHERE Statement

You can use the WHERE statement with any of the following Base SAS procedures that read a SAS data set:

COMPARE	REPORT
CORR	SORT
DATASETS (APPEND statement)	SQL
FREQ	STANDARD
MEANS or SUMMARY	TABULATE
PRINT	TRANSPOSE
RANK	UNIVARIATE

Details

- The COMPARE procedure and the APPEND statement in PROC DATASETS accept more than one input data set. For more information, see the documentation for the specific procedure.
- To subset the output data set, use the WHERE= data set option:

```
proc report data=debate nowd  
            out=onlyfr(where=(year='1'));  
run;
```

For more information about WHERE=, see [SAS DATA Step Statements: Reference](#).

Temporarily Dissociating a Format from a Variable

In this example, PROC PRINT prints only those observations that meet the condition of the WHERE expression.

```
options nodate pageno=1 linesize=64  
        pagesize=40;  
  
proc print data=debate noobs;  
  where gpa>3.5;  
  title 'Team Members with a GPA';  
  title2 'Greater than 3.5';  
run;
```

Output

Team Members with a GPA Greater than 3.5			
Name	Gender	Year	GPA
Capiccio	m	Freshman	3.598
Tucker	m	Freshman	3.901
Bagwell	f	Sophomore	3.722
Gold	f	Junior	3.609
Syme	f	Junior	3.883
Baglione	f	Senior	4.000
Carr	m	Senior	3.750
Hall	m	Senior	3.574

Usage: REPORT Procedure

Statistics That Are Available in PROC REPORT

Use the following keywords to specify statistic keywords in the KEYWORD or KEYLABEL statement.

Table 51.5 Statistics Available in PROC REPORT

Descriptive statistic keywords	
CSS	PCTSUM
CV	RANGE
MAX	STD
MEAN	STDERR
MIN	SUM

MODE	SUMWGT
N	USS
NMISS	VAR
PCTN	
Quantile statistic keywords	
MEDIAN P50	Q3 P75
P1	P60
P5	P70
P10	P80
P20	P90
P30	P95
P40	P99
Q1 P25	QRANGE
Hypothesis testing keyword	
PRT PROBT	T

These statistics, the formulas that are used to calculate them, and their data requirements are discussed in [“Keywords and Formulas” on page 2426](#).

To compute standard error and the Student's *t*-test, you must use the default value of VARDEF=, which is DF.

Every statistic except N must be associated with a variable. You associate a statistic with a variable either by placing the statistic above or below a numeric display variable or by specifying the statistic as a usage option in the DEFINE statement.

You can place N anywhere because it is the number of observations in the input data set that contribute to the value in a cell of the report. The value of N does not depend on a particular variable.

Note: If you use the MISSING option in the PROC REPORT statement, then N includes observations with missing group, order, or across variables.

Using ODS Styles with PROC REPORT

Using Styles with Base SAS Procedures

Most Base SAS procedures that support ODS use one or more table templates to produce output objects. These table templates include templates for table elements: columns, headers, and footers. Each table element can specify the use of one or more style elements for various parts of the output. These style elements cannot be specified within the syntax of the procedure, but you can use customized styles for the ODS destinations that you use. For more information about customizing tables and styles, see [“TEMPLATE Procedure: Creating a Style Template” in SAS Output Delivery System: Procedures Guide](#).

The Base SAS reporting procedures, PROC PRINT, PROC REPORT, and PROC TABULATE, enable you to quickly analyze your data and organize it into easy-to-read tables. You can use the STYLE= option with these procedure statements to modify the appearance of your report. The STYLE= option enables you to make changes in sections of output without changing the default style for all of the output. You can customize specific sections of procedure output by specifying the STYLE= option in specific statements within the procedure.

The following program uses the STYLE= option to create the background colors in the PROC REPORT output below:

```
ods _all_ close;
ods html5 file="file-path/customStyle.html";
title "Height and Weight by Gender and Age";
proc report nowd data=sashelp.class
  style(header)=[background=white];
  col age (('Gender' sex), (weight height));
  define age / style(header)=[background=lightgreen];
  define sex / across style(header)=[background=yellow] ' ';
  define weight / style(header)=[background=orange];
  define height / style(header)=[background=tan];
run;
ods _all_ close;
```

Output 51.1 Enhanced PROC REPORT Output**Height and Weight by Gender and Age**

	Gender			
	F		M	
Age	Weight	Height	Weight	Height
253	811	545.3	1089.5	639.1

Styles, Style Elements, and Style Attributes

Understanding Styles, Style Elements, and Style Attributes

The appearance of SAS output is controlled by ODS style templates (ODS styles). ODS styles are produced from compiled STYLE templates written in PROC TEMPLATE style syntax. An ODS style template is a collection of style elements that provides specific visual attributes for your SAS output.

- A style element is a named collection of style attributes that apply to a particular part of the output. Each area of ODS output has a style element name that is associated with it. The style element name specifies where the style attributes are applied. For example, a style element might contain instructions for the presentation of column headings or for the presentation of the data inside the cells. Style elements might also specify default colors and fonts for output that uses the style.
- A style attribute is a visual property, such as color, font properties, and line characteristics, that is defined in ODS with a reserved name and value. Style attributes are collectively referenced by a style element within a style template. Each *style attribute* specifies a value for one aspect of the presentation. For example, the BACKGROUND_COLOR= attribute specifies the color for the background of an HTML table or for a colored table in printed output. The FONTSTYLE= attribute specifies whether to use a Roman font or an italic font.

Note: Because styles control the presentation of the data, they have no effect on output objects that go to the DOCUMENT or OUTPUT destination.

Available styles are in the SASHELP.TMPLMST item store. You can display the list of available style templates by using the [LIST](#) statement in PROC TEMPLATE:

```
proc template;
  list styles / store=sashelp.tmplmst;
```

```
run;
```

For complete information about viewing ODS styles, see [“Viewing ODS Styles Supplied by SAS” in SAS Output Delivery System: Procedures Guide](#).

HTML5 output uses the HTML5Encore style template. To help you become familiar with styles, style elements, and style attributes, look at the relationship between them.

You can use the [SOURCE](#) statement in PROC TEMPLATE to display the structure of a style template. The following code prints the structure of the HTMLBlue style template to the SAS log:

```
proc template;  
  source styles.HTMLBlue;  
run;
```

The following figure illustrates the structure of a style. The figure shows the relationship between the style, the style elements, and the style attributes.

Figure 51.9 Diagram of the HtmlBlue Style

```

proc template;
  define style Styles.HtmlBlue; ← 1
    parent = styles.statistical;
    class GraphColors /
      'gblockheader' = cxcfd5de
      'gcphasebox' = cx989EA1
      'gphasebox' = cxDBE6F2
      'gczonec' = cxBECEE0
      'gzonec' = cxCCDCEE
      'gczoneb' = cxCCDCEE
      'gzoneb' = cxD7E5F3
      'gzonea' = cxE3EDF7
      'gconramp3cend' = cx9C1C00
      'gconramp3cneutral' = cx222222
      'gconramp3cstart' = cx0E36AC
      'gramp3cend' = cxD05B5B
      'gramp3cneutral' = cxFAFBFE
      'gramp3cstart' = cx667FA2
      'gcontrollim' = cxE6F2FF
      'gccontrollim' = cxBFC7D9
      'gruntest' = cxCAE3FF
      'gcruntest' = cxBF4D4D
      'gclipping' = cxFFFC6
      'gcclipping' = cxC1C100

...more style elements and style attributes...

class Header / ← 2
  bordercolor = cxB0B7BB ← 3
  backgroundcolor = cxEDF2F9 ← 3
  color = cx112277; ← 3
class Footer / ← 2
  bordercolor = cxB0B7BB ← 3
  backgroundcolor = cxEDF2F9 ← 3
  color = cx112277; ← 3
class RowHeader /
  bordercolor = cxB0B7BB
  backgroundcolor = cxEDF2F9
  color = cx112277;
class RowFooter /
  bordercolor = cxB0B7BB
  backgroundcolor = cxEDF2F9
  color = cx112277;
class Table /
  cellpadding = 5;
class Graph /
  attrpriority = "Color";
class GraphFit2 /
  linestyle = 1;
class GraphClipping /
  markersymbol = "circlefilled";
end;
run;
*** END OF TEXT *** ←

```

The following list corresponds to the numbered items in the preceding figure:

- 1 Styles.HtmlBlue is the *style*. Styles describe how to display presentation aspects (color, font, font size, and so on) of the SAS output. A style determines the overall appearance of the ODS documents that use it. The default style for HTML output is HtmlBlue. Each style consists of style elements.

You can create new styles with the “[DEFINE STYLE Statement](#)” in *SAS Output Delivery System: Procedures Guide*. New styles can be created independently or from an existing style. You can use “[PARENT= Statement](#)” in *SAS Output Delivery System: Procedures Guide* to create a new style from an existing style. For complete documentation about ODS styles, see “[Style Templates](#)” in *SAS Output Delivery System: Procedures Guide*.

- 2 Header and Footer are examples of *style elements*. A style element is a collection of style attributes that apply to a particular part of the output for a SAS program. For example, a style element might contain instructions for the presentation of column headings or for the presentation of the data inside table cells. Style elements might also specify default colors and fonts for output that uses the style. Style elements exist inside styles and consist of one or more style attributes. Style elements can be user-defined or supplied by SAS. User-defined style elements can be created by the “[STYLE Statement](#)” in *SAS Output Delivery System: Procedures Guide*.

Note: For a list of the default style elements used for HTML and markup languages and their inheritance, see “[Style Elements](#)” in *SAS Output Delivery System: Procedures Guide*.

- 3 BORDERCOLOR=, BACKGROUNDColor=, and COLOR= are examples of *style attributes*. Style attributes specify a value for one aspect of the area of the output that its style element applies to. For example, the COLOR= attribute specifies the value `cxx112277` for the font color. For a list of style attributes supplied by SAS, see “[Style Attributes](#)” in *SAS Output Delivery System: Procedures Guide*.

Style attributes can be referenced with style references. See “[style-reference](#)” in *SAS Output Delivery System: Procedures Guide* for more information about style references.

The following table shows commonly used style attributes that you can set with the STYLE= option in PROC PRINT, PROC TABULATE, and PROC REPORT. Most of these attributes apply to parts of the table other than cells (for example, table borders and the lines between columns and rows). Note that not all attributes are valid in all destinations. For more information about these style attributes, their valid values, and their applicable destinations, see “[Style Attributes Tables](#)” in *SAS Output Delivery System: Procedures Guide*.

Table 51.6 Style Attributes for PROC REPORT, PROC TABULATE, and PROC PRINT

Attribute	PROC REPORT STATEMENT REPORT Area	PROC REPORT Areas: CALLDEF, COLUMN, HEADER, LINES, SUMMARY	PROC TABULATE STATEMENT TABLE	PROC TABULATE STATEMENTS VAR, CLASS, BOX, CLASSLEV, KEYWORD	PROC PRINT TABLE location	PROC PRINT: all locations other than TABLE
ASIS=	X	X		X		X

Attribute	PROC REPORT STATEMENT REPORT Area	PROC REPORT Areas: CALLDEF, COLUMN, HEADER, LINES, SUMMARY	PROC TABULATE STATEMENT TABLE	PROC TABULATE STATEMENTS VAR, CLASS, BOX, CLASSLEV, KEYWORD	PROC PRINT TABLE location	PROC PRINT: all locations other than TABLE
BACKGROUNDCOLOR=	X	X	X	X	X	X
BACKGROUNDIMAGE=	X	X	X	X	X	X
BORDERBOTTOMCOLOR=	X	X		X		
BORDERBOTTOMSTYLE=	X	X	X	X		
BORDERBOTTOMWIDTH=	X	X	X	X		
BORDERLEFTCOLOR=	X	X		X		
BORDERLEFTSTYLE=	X	X	X	X		
BORDERLEFTWIDTH=	X	X	X	X		
BORDERCOLOR=	X	X		X	X	X
BORDERCOLORDARK=	X	X	X	X	X	X
BORDERCOLORLIGHT=	X	X	X	X	X	X
BODERRIGHTCOLOR=	X	X		X		
BODERRIGHTSTYLE=	X	X	X	X		
BODERRIGHTWIDTH=	X	X	X	X		
BORDERTOPCOLOR=	X	X		X		

Attribute	PROC REPORT STATEMENT REPORT Area	PROC REPORT Areas: CALLDEF, COLUMN, HEADER, LINES, SUMMARY	PROC TABULATE STATEMENT TABLE	PROC TABULATE STATEMENTS VAR, CLASS, BOX, CLASSLEV, KEYWORD	PROC PRINT TABLE location	PROC PRINT: all locations other than TABLE
BORDERTOPSTYLE=	X	X	X	X		
BORDERTOPWIDTH =	X	X	X	X		
BORDERWIDTH=	X	X	X	X	X	X
CELLPADDING=	X		X		X	
CELLSPACING=	X		X		X	
CELLWIDTH=	X	X	X	X		X
CLASS=	X	X	X	X	X	X
COLOR=	X	X	X			
FLYOVER=	X	X		X		X
FONT=	X	X	X	X	X	X
FONTFAMILY=	X	X	X	X	X	X
FONTSIZE=	X	X	X	X	X	X
FONTSTYLE=	X	X	X	X	X	X
FONTWEIGHT=	X	X	X	X	X	X
FONTWIDTH=	X	X	X	X		X
FRAME=	X		X		X	
HEIGHT=	X	X		X	X	X
HREFTARGET=		X		X		X
HTMLSTYLE=	X	X	X	X	X	
NOBREAKSPACE=2	X	X		X		X
OUTPUTWIDTH=	X	X	X	X	X	
POSTHTML=1	X	X	X	X	X	X

Attribute	PROC REPORT STATEMENT REPORT Area	PROC REPORT Areas: CALLDEF, COLUMN, HEADER, LINES, SUMMARY	PROC TABULATE STATEMENT TABLE	PROC TABULATE STATEMENTS VAR, CLASS, BOX, CLASSLEV, KEYWORD	PROC PRINT TABLE location	PROC PRINT: all locations other than TABLE
POSTIMAGE=	X	X	X	X	X	X
POSTTEXT= ¹	X	X	X	X	X	X
PREHTML= ¹	X	X	X	X	X	X
PREIMAGE=	X	X	X	X	X	X
PRETEXT= ¹	X	X	X	X	X	X
PROTECTSPECIALC HARS=		X		X	X	X
RULES=	X		X		X	
TAGATTR=	X	X	X	X	X	X
TEXTALIGN=	X	X	X	X	X	X
URL=		X		X		X
VERTICALALIGN=		X		X		X
WIDTH=	X	X	X	X	X	

- ¹ When you use these attributes in this location, they affect only the text that is specified with the PRETEXT=, POSTTEXT=, PREHTML=, and POSTHTML= attributes. To alter the foreground color or the font for the text that appears in the table, you must set the corresponding attribute in a location that affects the cells rather than the table. For complete documentation about style attributes and their values, see ["Style Attributes" in SAS Output Delivery System: Procedures Guide](#).
- ² To help prevent unexpected wrapping of long text strings when using PROC REPORT with the ODS RTF destination, set NOBREAKSPACE=OFF in a location that affects the LINE statement. The NOBREAKSPACE=OFF attribute must be set in the PROC REPORT code either on the LINE statement or on the PROC REPORT statement where style(lines) is specified.

Style Elements and Style Attributes for Table Regions

The following table lists the default style elements and style attributes for various regions of PROC REPORT output. [The locations in this table correspond to the locations in on page 1786.](#)

The table lists defaults for the most commonly used ODS destinations: HTML5, PDF, and RTF. Each destination has a default style template that is applied to all

output that is written to the destination. For complete documentation about the ODS destinations and their default styles, see [“Style Templates” in SAS Output Delivery System: Procedures Guide](#).

Note: The default style for ODS output in SAS Data and AI Studio is set by the theme that the session is actively using. Depending on the theme, the default is either Illuminate, Ignite, or HighContrast. For more information, see [“Customizing SAS Data and AI Studio” in SAS Data and AI Studio: User’s Guide](#).

- The default style for HTML5 output is HTMLEncore.
- The default style for PDF output is Pearl.
- The default style for RTF output is RTF.

Table 51.7 Default Style Elements and Style Attributes for Table Locations

Locations	Style Element	HTML5 Style Attributes	PDF Style Attributes	RTF Style Attributes
Header	Header	FONTFAMILY = "Arial, 'Albany AMT', Helvetica, Helv" FONTSIZE = 2 FONTWEIGHT = bold FONTSTYLE = roman COLOR = cx112277 BACKGROUNDCOLO R = cxd2f2f9	FONTFAMILY = "Arial, 'Albany AMT'" FONTSIZE = 8pt FONTWEIGHT = bold FONTSTYLE = roman COLOR = cx000000 BACKGROUNDCOLO R = cxffffff BORDERWIDTH = NaN	FONTFAMILY = "'Times New Roman', 'Times Roman'" FONTSIZE = 11pt FONTWEIGHT = bold FONTSTYLE = roman COLOR = cx000000 BACKGROUNDCOLO R = cxbbbbbb
Column	Data	FONTFAMILY = "Arial, 'Albany AMT', Helvetica, Helv" FONTSIZE = 2 FONTWEIGHT = medium FONTSTYLE = roman BACKGROUNDCOLO R = cxffffff	FONTFAMILY = "'Albany AMT', Albany" FONTSIZE = 8pt FONTWEIGHT = medium FONTSTYLE = roman COLOR = cx000000 BORDERWIDTH = NaN COLOR = cx000000	FONTFAMILY = "'Times New Roman', 'Times Roman'" FONTSIZE = 10pt FONTWEIGHT = medium FONTSTYLE = roman COLOR = cx000000
Summary	DataEmphas	FONTFAMILY = "Arial, 'Albany AMT', Helvetica, Helv" FONTSIZE = 2	FONTFAMILY = "'Albany AMT', Albany" FONTSIZE = 8pt	FONTFAMILY = "'Times New Roman', 'Times Roman'" FONTSIZE = 10pt

Locations	Style Element	HTML5 Style Attributes	PDF Style Attributes	RTF Style Attributes
		FONTWEIGHT = medium FONTSTYLE = roman BACKGROUNDCOLOR = cxffffff	FONTWEIGHT = medium FONTSTYLE = roman COLOR = cx000000 BORDERWIDTH = NaN	FONTWEIGHT = medium FONTSTYLE = italic COLOR = cx000000

Order of Precedence When Applying Style Attributes to Data Cells

PROC REPORT determines the style attributes to apply to a particular cell from a default order of precedence. Each step in the order of precedence specifies more granularity.

For example, use the style precedence for non-summary rows shown below. First, for a particular cell, PROC REPORT uses the default style attributes. Next, for each cell in a column, PROC REPORT overrides the default style attributes. By step five, the previous styles that were applied are overwritten, but only for the cells specified by the *column-id*.

The following lists the style precedence for the summary rows and the non-summary rows.

Style precedence for non-summary rows.

- 1 PROC REPORT uses the default style attributes from the default style element (Data).
- 2 The STYLE (COLUMN)= option in the PROC REPORT statement overrides the default style attributes.
- 3 A STYLE (COLUMN)= option in the DEFINE statement applies to all of the cells in the column.
- 4 A row style that is specified by the _ROW_ argument in the CALL DEFINE statement applies to all of the cells in the row.
- 5 A style specified by the *column-id* _COL_ in the CALL DEFINE statement applies to all of the cells in the column.

Style precedence for summary rows.

- 1 PROC REPORT uses the default style attributes from the default style elements for summary rows (DataEmphasis).
- 2 The STYLE (SUMMARY)= option in the PROC REPORT statement overrides the default style attributes for the summary.
- 3 The style specified by the STYLE= option in the BREAK statement, applies to the summary cells.

- 4 The row style specified by a CALL DEFINE statement with the `_ROW_` argument.
- 5 The style specified by a CALL DEFINE statement with the column-id `_COL_`, the column name, or the column number applies to all of the summary cells in the column.

Using a Format to Assign a Style Attribute Value

You can use a format to assign a style attribute value. For example, the following code assigns a red background color to cells in the Difference column for which the value is negative.

```
proc format;
  value proffmt low-<0='red'
    0-high='green';
run;

proc report data=sashelp.prdsale;
  column country predict actual diff;
  define country /group;
  define diff /'Difference' computed format=dollar12.2
    style(column)=[backgroundcolor=proffmt.];

  compute diff;
  diff = predict.sum - actual.sum;
  endcomp;
run;
```

Country	Predicted Sales	Actual Sales	Difference
CANADA	\$233,019.00	\$246,990.00	\$-13,971.00
GERMANY	\$231,554.00	\$245,998.00	\$-14,444.00
U.S.A.	\$241,722.00	\$237,349.00	\$4,373.00

Controlling the Spacing between Rows

There is frequently a need to “shrink” a report to fit more rows on a page. Shrinking a report involves changing both the font size and the spacing between the rows. In order to give maximum flexibility to the programmer, ODS uses the font size that is specified for the REPORT location to calculate the spacing between the rows. Therefore, to shrink a table, change the font size for both the REPORT location and the COLUMN location. Here is an example:

```
proc report
  data=libref.data-set-name
  style(report)=[fontsize=8pt]
  style(column)=[font=(Arial, 8pt)];
```

Storing and Reusing a Report Definition

The OUTREPT= option in the PROC REPORT statement stores a report definition in the specified catalog entry.

Note: A report definition might differ from the SAS program that creates the report. See the discussion of [OUTREPT=](#) on page 1783.

You can use a report definition to create an identically structured report for any SAS data set that contains variables with the same names as the ones that are used in the report definition. Use the REPORT= option in the PROC REPORT statement to load a report definition when you start PROC REPORT. For information, see [“REPORT=libref.catalog.entry”](#) on page 1785.

In-Database Processing for PROC REPORT

In-database processing has several advantages over processing within SAS. These advantages include increased security, reduced network traffic, and the potential for faster processing. Increased security is possible because sensitive data does not have to be extracted from the database management system (DBMS). Faster processing is possible because data is manipulated locally, on the DBMS, using high-speed secondary storage devices instead of being transported across a relatively slow network connection. The DBMS is used because it might have more processing resources at its disposal, and it might be capable of optimizing a query for execution in a highly parallel and scalable fashion.

When the DATA= input data set is stored as a table or view in a DBMS, the PROC REPORT procedure can use in-database processing to perform most of its work within the database. In-database processing can provide the advantages of faster processing and reduced data transfer between the database and SAS software.

In-database processing for PROC REPORT supports the following database management systems:

- Amazon Redshift
- DB2
- Greenplum
- Hadoop
- Impala
- Microsoft SQL Server
- MySQL(as of 2021.1.5)
- Netezza

- Oracle
- PostgreSQL
- SAP HANA
- Snowflake
- Spark
- Teradata
- Vertica

PROC REPORT performs in-database processing by using SQL implicit pass-through. The procedure generates SQL queries that are based on the statements and the PROC REPORT options that are used as well as the output statistics that are specified in the procedure. The database executes these SQL queries and the results of the query are then transmitted to PROC REPORT. To examine the generated SQL, set the SASTRACE= option.

If the SAS format definitions have not been deployed in the database, the in-database aggregation occurs on the raw values, and the relevant formats are applied by SAS as the results' set is merged into the PROC REPORT internal structures. For more information, see the section “Deploying and Using SAS Formats” in *SAS/ACCESS for Relational Databases: Reference*.

In-database processing will not occur if the PROC REPORT step contains variables with usage types DISPLAY or ORDER.

The following statistics are supported for in-database processing: N, NMISS, MIN, MAX, MEAN, RANGE, SUM, SUMWGT, CSS, USS, VAR, STD, STDERR, and CV.

Weighting for in-database processing is supported only for N, NMISS, MIN, MAX, RANGE, SUM, SUMWGT, and MEAN.

The SQLGENERATION system option or LIBNAME statement option controls whether and how in-database procedures are run inside the database. By default, the in-database procedures are run inside the database when possible. There are many data set options that prevent in-database processing. For a complete listing, refer to “In-Database Procedures” in *SAS/ACCESS for Relational Databases: Reference*.

For more information about in-database processing, see *SAS/ACCESS for Relational Databases: Reference*.

CAS Processing for PROC REPORT

If your input data set originates from SAS Cloud Analytic Services (CAS), some of the PROC REPORT analysis can be performed by the CAS server. Reports that require significant summarization of data can benefit from CAS processing. When run against a CAS table, PROC REPORT executes on the analytic server and uses the appropriate statistical actions. For more information, see [Chapter 4, “CAS Processing of Base Procedures,” on page 75](#).

The CAS LIBNAME statement option controls whether and how CAS procedures are run inside CAS. By default, the CAS procedures are run inside CAS when possible. However, there are many data set options that can prevent CAS processing.

When the DATA= input data set references an in-memory table or view in CAS, the REPORT procedure can use CAS actions to perform some of its work within the server. To reference an in-memory table or view, you must specify the CAS engine LIBNAME statement and specify the CAS engine libref option IN= or DATA=. By default, PROC REPORT uses CAS processing whenever a CAS engine libref is specified on the input.

The following example shows how to run PROC REPORT that uses CAS processing. The LIBNAME statement assigns a CAS engine libref named mycas that you use to connect to the CAS session casauto.

```
option casport=10935 cashost="cloud.example.com";
cas casauto ;

libname mycas cas;
data mycas.cars;
    set sashelp.cars;
run;

proc report data=mycas.cars;
title;
column ('Horsepower by Make and Drivetrain as a Percent of All
Horsepower'
origin drivetrain horsepower, (sum pctsum));
define origin / group;
define drivetrain / group;
define horsepower / '';
define sum / 'HP by Make and Drivetrain';
define pctsum / 'Percent of HP' format=percent6.;
rbreak after / summarize style=[font_style=italic];
run;
```

CAS processing does not occur if the PROC REPORT step contains variables with usage types DISPLAY or ORDER. If a DISPLAY or ORDER variable is found, all of the data is brought back to the client and PROC REPORT runs on the SAS client server.

The following statistics are supported for CAS processing:

CSS	RANGE
CV	STDERR
MAX	SUM
MEAN	SUMWGT
MIN	STD
N	USS
NMISS	VAR

Note: Weighting for CAS processing is supported only for N, NMISS, MIN, MAX, RANGE, SUM, SUMWGT, and MEAN.

The computations and processing are done in CAS. However, the rendering of the final table is done by the SAS client.

For information about how to use the CAS LIBNAME statement, see [“LIBNAME Statement: CAS Engine” in SAS Cloud Analytic Services: User’s Guide](#).

Substituting BY Line Values in a Text String

The #BYLINE, #BYVAR, and #BYVAL substitutions are available in the following options:

- The CONTENTS= option in the PROC REPORT statement.
- The CAPTION= option in the PROC REPORT statement

To use the #BYVAR and #BYVAL substitutions, insert the item in the text string at the position where you want the substitution text to appear. Both #BYVAR and #BYVAL specifications must be followed by a delimiting character. The character can be either a space or other non-alphanumeric character, such as a quotation mark. If no delimiting character is provided, then the specification is ignored and its text remains intact and is displayed with the rest of the string. To allow a #BYVAR or #BYVAL substitution to be followed immediately by other text, with no delimiter, use a trailing dot (as with macro variables). The trailing dot is not displayed in the resolved text. If you want a period to be displayed as the last character in the resolved text, use two dots after the #BYVAR or #BYVAL substitution.

The substitution for #BYVAR or #BYVAL does not occur in the following cases:

- if you use a #BYVAR or #BYVAL specification for a variable that is not named in the BY statement. For example, you might use #BYVAL2 when there is only one BY-variable or #BYVAL(ABC) when ABC is non-existent or is not a BY-variable.
- if there is no BY statement

Results: REPORT Procedure

How PROC REPORT Builds a Report

Sequence of Events

This section explains the general process of building a report. For examples that illustrate this process, see [“Report-Building Examples” on page 1852](#).

To understand the process of building a report, you must understand the difference between report variables and temporary variables. **Report variables** are variables that are specified in the COLUMN statement. A report variable can come from the input data set or can be computed (that is, the DEFINE statement for that variable specifies the COMPUTED option). A report variable might or might not appear in a compute block. Variables that appear only in one or more compute blocks are **temporary variables**. Temporary variables do not appear in the report and are not written to the output data set (if one is requested).

PROC REPORT initializes report variables to missing at the beginning of each row of the report. The value for a **temporary variable** is initialized to missing before PROC REPORT begins to construct the rows of the report, and it remains missing until you specifically assign a value to it. PROC REPORT retains the value of a **temporary variable** from the execution of one compute block to another.

PROC REPORT constructs a report as follows:

- 1 It consolidates the data by group, order, and across variables. It calculates all statistics for the report, the statistics for detail rows as well as the statistics for summary lines in breaks. Statistics include those statistics that are computed for analysis variables. PROC REPORT calculates statistics for summary lines whether they appear in the report.
- 2 It initializes all temporary variables to missing.
- 3 It begins constructing the rows of the report.
 - a At the beginning of each row, it initializes all report variables to missing.
 - b It fills in values for report variables from left to right.
 - Values for computed variables come from executing the statements in the corresponding compute blocks.

- Values for all other variables come from the data set or the summary statistics that were computed at the beginning of the report-building process.
- c Whenever it comes to a break, PROC REPORT first constructs the break lines that are created with the BREAK or RBREAK statement or with options in the BREAK window.
- d If there is a compute block attached to the break, PROC REPORT then executes the statements in the compute block. See [“Construction of Summary Lines” on page 1851](#) for details.

Note: You can also use statistics with PROC REPORT as follows.

- Use group statistics in compute blocks for a break before the group variable.
- Use statistics for the whole report in a compute block at the beginning of the report.

This document references these statistics with the appropriate compound name. For information about referencing report items in a compute block, see [“Four Ways to Reference Report Items in a Compute Block” on page 1766](#).

Note: You cannot use the LINE statement in conditional statements (IF-THEN, IF-THEN/ELSE, and SELECT) because it is not executed until PROC REPORT has executed all other statements in the compute block.

- 4 After each report row is completed, PROC REPORT sends the row to all of the ODS destinations that are currently open.

Construction of Summary Lines

PROC REPORT constructs a summary line for a break if either of the following conditions is true:

- You summarize numeric variables in the break.
- You use a compute block at the break. (You can attach a compute block to a break without using a BREAK or RBREAK statement or without selecting any options in the BREAK window.)

For more information about using compute blocks, see [“Using Compute Blocks” on page 1764](#) and [“COMPUTE Statement” on page 1806](#).

The summary line that PROC REPORT constructs at this point is preliminary. If no compute block is attached to the break, then the preliminary summary line becomes the final summary line. However, if a compute block is attached to the break, then the statements in the compute block can alter the values in the preliminary summary line.

PROC REPORT prints the summary line only if you summarize numeric variables in the break.

Report-Building Examples

Building a Report That Uses Groups and a Report Summary

The report in [“Report with Groups and a Report Summary” on page 1853](#) contains five columns:

- Sector and Department are group variables.
- Sales is an analysis variable that is used to calculate the Sum statistic.
- Profit is a computed variable whose value is based on the value of Department.
- The N statistic indicates how many observations each row represents.

At the end of the report a break summarizes the statistics and computed variables in the report and assigns to Sector the value of TOTALS:.

The following statements produce [“Report with Groups and a Report Summary” on page 1853](#). The user-defined formats that are used are created by a [PROC FORMAT step on page 1862](#).

```
libname proclib 'SAS-library';

options nodate pageno=1 linesize=64
        pagesize=60 fmtsearch=(proclib);

proc report data=grocery;
  column sector department sales Profit N ;
  define sector / group format=$sctrfmt.;
  define department / group format=$deptfmt.;
  define sales / analysis sum
                format=dollar9.2;
  define profit / computed format=dollar9.2;

  compute before;
  totprof = 0;
  endcomp;

  compute profit;
  if sector ne ' ' or department ne ' ' then do;
    if department='np1' or department='np2'
      then profit=0.4*sales.sum;
    else profit=0.25*sales.sum;
    totprof = totprof + profit;
  end;
  else
    profit = totprof;
  endcomp;

  rbreak after / summarize;
  compute after;
    sector='TOTALS:.';
  endcomp;
```

```

where sector contains 'n';
title 'Report for Northeast and Northwest Sectors';
run;
ods listing close;

```

Example Code 51.2 Report with Groups and a Report Summary

Report for Northeast and Northwest Sectors				1
Sector	Department	Sales	Profit	N

Northeast	Canned	\$840.00	\$336.00	2
	Meat/Dairy	\$490.00	\$122.50	2
	Paper	\$290.00	\$116.00	2
	Produce	\$211.00	\$52.75	2
Northwest	Canned	\$1,070.00	\$428.00	3
	Meat/Dairy	\$1,055.00	\$263.75	3
	Paper	\$150.00	\$60.00	3
	Produce	\$179.00	\$44.75	3
=====		=====	=====	=====
TOTALS:		\$4,285.00	\$1,423.75	20
=====		=====	=====	=====

A description of how PROC REPORT builds this report follows:

- 1 PROC REPORT starts building the report by consolidating the data (Sector and Department are group variables) and by calculating the statistics (Sales.sum and N) for each detail row and for the break at the end of the report.
- 2 Now, PROC REPORT is ready to start building the first row of the report. This report does not contain a break at the beginning of the report or a break before any groups, so the first row of the report is a detail row. The procedure initializes all report variables to missing, as the following figure illustrates. Missing values for a character variable are represented by a blank, and missing values for a numeric variable are represented by a period.

Figure 51.10 First Detail Row with Values Initialized

Sector	Department	Sales	Profit	N
		.	.	.

- 3 The following figure illustrates the construction of the first three columns of the row. PROC REPORT fills in values for the row from left to right. Values come from the statistics that were computed at the beginning of the report-building process.

Figure 51.11 First Detail Row with Values Filled in from Left to Right

Sector	Department	Sales	Pr ofit	N
Northeast		.	.	.

Sector	Department	Sales	Pr ofit	N
Northeast	Canned	.	.	.

Sector	Department	Sales	Pr ofit	N
Northeast	Canned	\$840.00	.	.

- 4 The next column in the report contains the computed variable Profit. When it gets to this column, PROC REPORT executes the statements in the compute block that is attached to Profit. Nonperishable items (which have a value of **np1** or **np2**) return a profit of 40%; perishable items (which have a value of **p1** or **p2**) return a profit of 25%.

```

if department='np1' or department='np2'
  then profit=0.4*sales.sum;
else profit=0.25*sales.sum;

```

The row now looks like the following figure.

Note: The position of a computed variable is important. PROC REPORT assigns values to the columns in a row of a report from left to right. Consequently, you cannot base the calculation of a computed variable on any variable that appears to its right in the report.

Figure 51.12 A Computed Variable Added to the First Detail Row

Sector	Department	Sales	Pr ofit	N
Northeast	Canned	\$840.00	\$336.00	.

- 5 The **totprof** variable is a temporary variable held in memory to keep a running total of the total profits. When each Sector and Department profits have been calculated, the value stored in variable **totprof** is then moved to Profit and reported in the TOTALS summary line.

```

totprof = totprof + profit;
end;
else
  profit = totprof;

```

- 6 Next, PROC REPORT fills in the value for the N statistic. The value comes from the statistics that are created at the beginning of the report-building process. The following figure illustrates the completed row.

Figure 51.13 First Complete Detail Row

Sector	Department	Sales	Profit	N
Northeast	Canned	\$840.00	\$336.00	2

- 7 The procedure writes the completed row to the report.
- 8 PROC REPORT repeats steps 2, 3, 4, 5, and 6 for each detail row in the report.
- 9 At the break at the end of the report, PROC REPORT constructs the break lines described by the RBREAK statement. These lines include a preliminary version of the summary line. The statistics for the summary line were calculated earlier. (See step 1.) The value for the computed variable is calculated when PROC REPORT reaches the appropriate column, just as it is in detail rows. PROC REPORT uses these values to create the preliminary version of the summary line.

Figure 51.14 Preliminary Summary Line

Sector	Department	Sales	Profit	N
		\$4,285.00	\$1,423.75	20

- 10 If no compute block is attached to the break, then the preliminary version of the summary line is the same as the final version. However, in this example, a compute block is attached to the break. Therefore, PROC REPORT now executes the statements in that compute block. In this case, the compute block contains one statement:

```
sector='TOTALS:' ;
```

This statement replaces the value of Sector, which in the summary line is missing by default, with the word `TOTALS:`. After PROC REPORT executes the statement, it modifies the summary line to reflect this change to the value of Sector. The final version of the summary line appears in the following figure.

Figure 51.15 Final Summary Line

Sector	Department	Sales	Profit	N
TOTALS:		\$4,285.00	\$1,423.75	20

- 11 Finally, PROC REPORT writes all the break lines and the final summary line, to the report. See [“Report with Groups and a Report Summary” on page 1853](#).

Building a Report That Uses Temporary Variables

PROC REPORT initializes report variables to missing at the beginning of each row of the report. The value for a temporary variable is initialized to missing before PROC REPORT begins to construct the rows of the report, and it remains missing

until you specifically assign a value to it. PROC REPORT retains the value of a temporary variable from the execution of one compute block to another.

Because all compute blocks share the current values of all variables, you can initialize temporary variables at a break at the beginning of the report or at a break before a break variable. This report initializes the temporary variable Sctrtot at a break before Sector.

Note: PROC REPORT creates a preliminary summary line for a break before it executes the corresponding compute block. If the summary line contains computed variables, then the computations are based on the values of the contributing variables in the preliminary summary line. If you want to recalculate computed variables based on values that you set in the compute block, then you must do so explicitly in the compute block. This report illustrates this technique. If no compute block is attached to a break, then the preliminary summary line becomes the final summary line.

The report in [“Report with Temporary Variables” on page 1857](#) contains five columns:

- Sector and Department are group variables.
- Sales is an analysis variable that is used twice in this report: once to calculate the Sum statistic, and once to calculate the Pctsum statistic.
- Sctrpct is a computed variable whose values are based on the values of Sales and a temporary variable, Sctrtot, which is the total sales for a sector.

At the beginning of the report, a customized report summary tells what the sales for all stores are. At a break before each group of observations for a department, a default summary summarizes the data for that sector. At the end of each group a break inserts a blank line.

The following statements produce [“Report with Temporary Variables” on page 1857](#). The user-defined formats that are used are created by a [PROC FORMAT step on page 1862](#).

Note: Calculations of the percentages do not multiply their results by 100 because PROC REPORT prints them with the PERCENT. format.

```
libname proclib 'SAS-library';

proc report data=grocery ;
  column sector department sales Sctrpct sales=Salespct;
  define sector / 'Sector' group format=$sctrfmt.;
  define department / group format=$deptfmt.;
  define sales / 'Sum of/sales' analysis sum format=dollar9.2 ;
  define sctrpct / 'Percent/of Sector' computed format=percent9.2 ;
  define salespct / 'Percent of/All Stores' pctsum format=percent9.2;
  compute before _page_;
  line ' ';
  line 'Total for all stores is ' sales.sum dollar9.2;
  line ' ';
  endcomp;
  break before sector / summarize ;
```

```

compute before sector;
sctrtot=sales.sum;
sctrpct=sales.sum/sctrtot;
endcomp;
compute sctrpct;
sctrpct=sales.sum/sctrtot;
endcomp;
compute after sector;
line ' ';
endcomp;;
where sector contains 'n';
title 'Report for Northeast and Northwest Sectors';
run;

```

Example Code 51.3 Report with Temporary Variables

Report for Northeast and Northwest Sectors					1
Total for all stores is \$4,285.00					
Sector	Department	Sum of Sales	Percent of Sector	Percent of All Stores	
=====					
Northeast		\$1,831.00	100.00%	42.73%	
-----		-----	-----	-----	
Northeast	Canned	\$840.00	45.88%	19.60%	
	Meat/Dairy	\$490.00	26.76%	11.44%	
	Paper	\$290.00	15.84%	6.77%	
	Produce	\$211.00	11.52%	4.92%	
Northwest		\$2,454.00	100.00%	57.27%	
-----		-----	-----	-----	
Northwest	Canned	\$1,070.00	43.60%	24.97%	
	Meat/Dairy	\$1,055.00	42.99%	24.62%	
	Paper	\$150.00	6.11%	3.50%	
	Produce	\$179.00	7.29%	4.18%	

A description of how PROC REPORT builds this report follows:

- 1 PROC REPORT starts building the report by consolidating the data (Sector and Department are group variables) and by calculating the statistics (Sales.sum and Sales.pctsum) for each detail row, for the break at the beginning of the report, for the breaks before each group, and for the breaks after each group.
- 2 PROC REPORT initializes the temporary variable, Sctrtot, to missing.

Figure 51.16 Initialized Temporary Variables

Report Variables					Temporary Variable
Sector	Department	Sales.sum	Sctrpct	Sales.pctsum	Sctrtot
		.	.	.	.

- 3 Because this PROC REPORT step contains a COMPUTE BEFORE statement, the procedure constructs a preliminary summary line for the break at the beginning

of the report. This preliminary summary line contains values for the statistics (Sales.sum and Sales.pctsum) and the computed variable (Sctrpct).

At this break, Sales.sum is the sales for all stores, and Sales.pctsum is the percentage those sales represent for all stores (100%). PROC REPORT takes the values for these statistics from the statistics that were computed at the beginning of the report-building process.

The value for Sctrpct comes from executing the statements in the corresponding compute block. Because the value of Sctrtot is missing, PROC REPORT cannot calculate a value for Sctrpct. Therefore, in the preliminary summary line (which is not printed in this case), this variable also has a missing value.

The statements in the COMPUTE BEFORE block do not alter any variables. Therefore, the final summary line is the same as the preliminary summary line.

Note: The COMPUTE BEFORE statement creates a break at the beginning of the report. You do not need to use an RBREAK statement.

Figure 51.17 Preliminary and Final Summary Line for the Break at the Beginning of the Report

Report Variables					Temporary Variable
Sector	Department	Sales.sum	Sctrpct	Sales.pctsum	Sctrtot
		\$4,285.00	.	100.00%	.

- Because the program does not include an RBREAK statement with the SUMMARIZE option, PROC REPORT does not write the final summary line to the report. Instead, it uses LINE statements to write a customized summary that embeds the value of Sales.sum into a sentence and to write customized column headings.
- Next, PROC REPORT constructs a preliminary summary line for the break before the first group of observations. (This break both uses the SUMMARIZE option in the BREAK statement with a compute block attached to it. Either of these conditions generates a summary line.) The preliminary summary line contains values for the break variable (Sector), the statistics (Sales.sum and Sales.pctsum), and the computed variable (Sctrpct). At this break, Sales.sum is the sales for one sector (the northeast sector). PROC REPORT takes the values for Sector, Sales.sum, and Sales.pctsum from the statistics that were computed at the beginning of the report-building process.

The value for Sctrpct comes from executing the statements in the corresponding compute blocks. Because the value of Sctrtot is still missing, PROC REPORT cannot calculate a value for Sctrpct. Therefore, in the preliminary summary line, Sctrpct has a missing value.

Figure 51.18 Preliminary Summary Line for the Break before the First Group of Observations

Report Variables					Temporary Variable
Sector	Department	Sales.sum	Sctrpct	Sales.pctsum	Sctrtot
Northeast		\$1,831.00	.	42.73%	.

- 6 PROC REPORT creates the final version of the summary line by executing the statements in the COMPUTE BEFORE SECTOR compute block. These statements execute once each time the value of Sector changes.
 - The first statement assigns the value of Sales.sum, which in that part of the report represents total sales for one Sector, to the variable Sctrtot.
 - The second statement completes the summary line by recalculating Sctrpct from the new value of Sctrtot. The following figure shows the final summary line.

Note: In this example, you must recalculate the value for Sctrpct in the final summary line. If you do not recalculate the value for Sctrpct, then it is missing because the value of Sctrtot is missing when the COMPUTE Sctrpct block executes.

Figure 51.19 Final Summary Line for the Break before the First Group of Observations

Report Variables					Temporary Variable
Sector	Department	Sales.sum	Sctrpct	Sales.pctsum	Sctrtot
Northeast		\$1,831.00	100.00%	42.73%	\$1,831.00

- 7 Because the program contains a BREAK BEFORE statement with the SUMMARIZE option, PROC REPORT writes the final summary line to the report.
- 8 Now, PROC REPORT is ready to start building the first report row. It initializes all report variables to missing. Values for temporary variables do not change. The following figure illustrates the first detail row at this point.

Figure 51.20 First Detail Row with Initialized Values

Report Variables					Temporary Variable
Sector	Department	Sales.sum	Sctrpct	Sales.pctsum	Sctrtot
		.	.	.	\$1,831.00

- 9 The following figure illustrates the construction of the first three columns of the row. PROC REPORT fills in values for the row from left to right. The values come from the statistics that were computed at the beginning of the report-building process.

Figure 51.21 Filling in Values from Left to Right

Report Variables					Temporary Variable
Sector	Department	Sales.sum	Sctrpct	Sales.pctsum	Sctrtot
Northeast		.	.	.	\$1,831.00

Report Variables					Temporary Variable
Sector	Department	Sales.sum	Sctrpct	Sales.pctsum	Sctrtot
Northeast	Canned	.	.	.	\$1,831.00

Report Variables					Temporary Variable
Sector	Department	Sales.sum	Sctrpct	Sales.pctsum	Sctrtot
Northeast	Canned	\$840.00	.	.	\$1,831.00

- 10 The next column in the report contains the computed variable Sctrpct. When it gets to this column, PROC REPORT executes the statement in the compute block attached to Sctrpct. This statement calculates the percentage of the sector's total sales that this department accounts for.

```
sctrpct=sales.sum/sctrtot;
```

The row now looks like the following figure.

Figure 51.22 First Detail Row with the First Computed Variable Added

Report Variables					Temporary Variable
Sector	Department	Sales.sum	Sctrpct	Sales.pctsum	Sctrtot
Northeast	Canned	\$840.00	45.88%	.	\$1,831.00

- 11 The next column in the report contains the statistic Sales.pctsum. PROC REPORT gets this value from the statistics that are created at the beginning of the report-building process. The first detail row is now complete.

Figure 51.23 First Complete Detail Row

Report Variables					Temporary Variable
Sector	Department	Sales.sum	Sctrpct	Sales.pctsum	Sctrtot
Northeast	Canned	\$840.00	45.88%	19.60%	\$1,831.00

- 12 PROC REPORT writes the detail row to the report. It repeats steps 8, 9, 10, 11, and 12 for each detail row in the group.
- 13 After writing the last detail row in the group to the report, PROC REPORT constructs the default group summary. Because no compute block is attached to this break and because the BREAK AFTER statement does not include the SUMMARIZE option, PROC REPORT does not construct a summary line.
- 14 Now the value of the break variable changes from Northeast to Northwest. PROC REPORT constructs a preliminary summary line for the break before this group of observations. As at the beginning of any row, PROC REPORT initializes all report variables to missing but retains the value of the temporary variable. Next, it completes the preliminary summary line with the appropriate values for

the break variable (Sector), the statistics (Sales.sum and Sales.pctsum), and the computed variable (Sctrpct). At this break, Sales.sum is the sales for the Northwest sector. Because the COMPUTE BEFORE Sector block has not yet executed, the value of Sctrtot is still \$1,831.00, the value for the Northeast sector. Thus, the value that PROC REPORT calculates for Sctrpct in this preliminary summary line is incorrect. The statements in the compute block for this break calculate the correct value.

Figure 51.24 Preliminary Summary Line for the Break before the Second Group of Observations

Report Variables					Temporary Variable
Sector	Department	Sales.sum	Sctrpct	Sales.pctsum	Sctrtot
Northwest		\$2,454.00	134.00%	57.27%	\$1,831.00

CAUTION

Synchronize values for computed variables in break lines to prevent incorrect results. If the PROC REPORT step does not recalculate Sctrpct in the compute block that is attached to the break, then the value in the final summary line is not synchronized with the other values in the summary line, and the report is incorrect.

- 15 PROC REPORT creates the final version of the summary line by executing the statements in the COMPUTE BEFORE Sector compute block. These statements execute once each time the value of Sector changes.
 - The first statement assigns the value of Sales.sum, which in that part of the report represents sales for the Northwest sector, to the variable Sctrtot.
 - The second statement completes the summary line by recalculating Sctrpct from the new, appropriate value of Sctrtot. The following figure shows the final summary line.

Figure 51.25 Final Summary Line for the Break before the Second Group of Observations

Report Variables					Temporary Variable
Sector	Department	Sales.sum	Sctrpct	Sales.pctsum	Sctrtot
Northwest		\$2,454.00	100.00%	57.27%	\$2,454.00

Because the program contains a BREAK BEFORE statement with the SUMMARIZE option, PROC REPORT writes the final summary line to the report.

- 16 Now, PROC REPORT is ready to start building the first row for this group of observations. It repeats steps 8 through 16 until it has processed all observations in the input data set (stopping with step 14 for the last group of observations).

Examples: REPORT Procedure

Example 1: Selecting Variables and Creating a Summary Line for a Report

Features:

- PROC REPORT statement options
- COLUMN statement
- RBREAK statement options
 - AFTER
 - STYLE=
 - SUMMARIZE
- LIBNAME statement
- PROC FORMAT statement
- DATA step
- OPTIONS statement
- FORMAT statement
- WHERE statement
- TITLE statement

Details

This example uses a permanent data set and permanent formats to create a report that contains the following:

- one row for every observation
- a default summary for the whole report

Program

```
libname proclib 'SAS-library';  
data grocery;  
    input Sector $ Manager $ Department $ Sales @@;  
    datalines;  
se 1 np1 50      se 1 p1 100      se 1 np2 120      se 1 p2 80
```

```

se 2 np1 40    se 2 p1 300    se 2 np2 220    se 2 p2 70
nw 3 np1 60    nw 3 p1 600    nw 3 np2 420    nw 3 p2 30
nw 4 np1 45    nw 4 p1 250    nw 4 np2 230    nw 4 p2 73
nw 9 np1 45    nw 9 p1 205    nw 9 np2 420    nw 9 p2 76
sw 5 np1 53    sw 5 p1 130    sw 5 np2 120    sw 5 p2 50
sw 6 np1 40    sw 6 p1 350    sw 6 np2 225    sw 6 p2 80
ne 7 np1 90    ne 7 p1 190    ne 7 np2 420    ne 7 p2 86
ne 8 np1 200   ne 8 p1 300    ne 8 np2 420    ne 8 p2 125
;

proc format library=proclib;
    value $sctrfmt 'se' = 'Southeast'
                  'ne' = 'Northeast'
                  'nw' = 'Northwest'
                  'sw' = 'Southwest';

    value $mgrfmt '1' = 'Smith'    '2' = 'Jones'
                  '3' = 'Reveiz'  '4' = 'Brown'
                  '5' = 'Taylor'  '6' = 'Adams'
                  '7' = 'Alomar'  '8' = 'Andrews'
                  '9' = 'Pelfrey';

    value $deptfmt 'np1' = 'Paper'
                  'np2' = 'Canned'
                  'p1'  = 'Meat/Dairy'
                  'p2'  = 'Produce';

run;

options fmtsearch=(proclib);

ods html file="myhtmlfile.html";

proc report data=grocery;

    column manager department sales;

    rbreak after / summarize style=[font_weight=bold];

    where sector='se';

    format manager $mgrfmt. department $deptfmt.
             sales dollar11.2;

    title 'Sales for the Southeast Sector';
    title2 "for &sysdate";

run;

ods html close;

```

Program Description

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

```
libname proclib 'SAS-library';
```

Create the GROCERY data set. GROCERY contains one day's sales figures for eight stores in the Grocery Mart chain. Each observation contains one day's sales data for one department in one store.

```

data grocery;
  input Sector $ Manager $ Department $ Sales @@;
  datalines;
se 1 np1 50      se 1 p1 100      se 1 np2 120      se 1 p2 80
se 2 np1 40      se 2 p1 300      se 2 np2 220      se 2 p2 70
nw 3 np1 60      nw 3 p1 600      nw 3 np2 420      nw 3 p2 30
nw 4 np1 45      nw 4 p1 250      nw 4 np2 230      nw 4 p2 73
nw 9 np1 45      nw 9 p1 205      nw 9 np2 420      nw 9 p2 76
sw 5 np1 53      sw 5 p1 130      sw 5 np2 120      sw 5 p2 50
sw 6 np1 40      sw 6 p1 350      sw 6 np2 225      sw 6 p2 80
ne 7 np1 90      ne 7 p1 190      ne 7 np2 420      ne 7 p2 86
ne 8 np1 200     ne 8 p1 300      ne 8 np2 420      ne 8 p2 125
;

```

Create the \$SCTRFMT., \$MGRFMT., and \$DEPTFMT. formats. PROC FORMAT creates permanent formats for Sector, Manager, and Department. The LIBRARY= option specifies a permanent storage location so that the formats are available in subsequent SAS sessions. These formats are used for examples throughout this section.

```

proc format library=proclib;
  value $sctrfmt 'se' = 'Southeast'
                'ne' = 'Northeast'
                'nw' = 'Northwest'
                'sw' = 'Southwest';

  value $mgrfmt '1' = 'Smith'   '2' = 'Jones'
               '3' = 'Reveiz'  '4' = 'Brown'
               '5' = 'Taylor'  '6' = 'Adams'
               '7' = 'Alomar'  '8' = 'Andrews'
               '9' = 'Pelfrey';

  value $deptfmt 'np1' = 'Paper'
                'np2' = 'Canned'
                'p1'  = 'Meat/Dairy'
                'p2'  = 'Produce';

run;

```

Specify the format search library. The SAS system option FMTSEARCH= adds the SAS library PROCLIB to the search path that is used to locate formats.

```
options fmtsearch=(proclib);
```

Specify the ODS output.

```
ods html file="myhtmlfile.html";
```

Specify the report options. By default, REPORT procedure sends its output to the open output destinations.

```
proc report data=grocery;
```

Specify the report columns. The report contains a column for Manager, Department, and Sales. Because there is no DEFINE statement for any of these variables, PROC REPORT uses the character variables (Manager and Department) as display variables and the numeric variable (Sales) as an analysis variable that is used to calculate the sum statistic.

```
column manager department sales;
```

Produce a report summary. The RBREAK statement produces a default summary at the end of the report. SUMMARIZE sums the value of Sales for all observations in the report. STYLE= is used to bold the summarized value.

```
rbreak after / summarize style=[font_weight=bold];
```

Select the observations to process. The WHERE statement selects for the report only the observations for stores in the southeast sector.

```
where sector='se';
```

Format the report columns. The FORMAT statement assigns formats to use in the report. You can use the FORMAT statement only with data set variables.

```
format manager $mgrfmt. department $deptfmt.
sales dollar11.2;
```

Specify the titles. SYSDATE is an automatic macro variable that returns the date on which the SAS job or SAS session began. The TITLE2 statement uses double rather than single quotation marks so that the macro variable resolves.

```
title 'Sales for the Southeast Sector';
title2 "for &sysdate";
run;
```

Close the ODS destination.

```
ods html close;
```

Output: HTML

Output 51.2 Selecting Variables and Creating a Summary Line for a Report

Sales for the Southeast Sector for 03MAY14

Manager	Department	Sales
Smith	Paper	\$50.00
Smith	Meat/Dairy	\$100.00
Smith	Canned	\$120.00
Smith	Produce	\$80.00
Jones	Paper	\$40.00
Jones	Meat/Dairy	\$300.00
Jones	Canned	\$220.00
Jones	Produce	\$70.00
		\$980.00

Example 2: Ordering the Rows in a Report

Features: PROC REPORT statement options
COLUMN statement
DEFINE statement options
 ANALYSIS
 FORMAT=
 ORDER
 ORDER=
 SUM
BREAK statement options
 AFTER
 SUMMARIZE
 STYLE=
LIBNAME statement
OPTIONS statement
WHERE statement
TITLE statement

Data set: [GROCERY](#)

Format: [\\$MGRFMT](#)

Format: [\\$DEPTFMT](#)

Details

This example does the following:

- arranges the rows alphabetically by the formatted values of Manager and the internal values of Department (so that sales for the two departments that sell nonperishable goods precede sales for the two departments that sell perishable goods)
- controls the default column width and the spacing between columns
- creates a default summary of Sales for each manager
- creates a customized summary of Sales for the whole report

Program

```
libname proclib 'SAS-library';  
options fmtsearch=(proclib);
```



```
ods html file="myhtmlfile.html";

proc report data=grocery;

    column manager department sales;

    define manager / order order=formatted format=$mgrfmt.;
    define department / order order=internal format=$deptfmt.;

    define sales / analysis sum format=dollar7.2;

    break after manager / summarize
                        style=[font_style=italic];

    where sector='se';

    title 'Sales for the Southeast Sector';
run;

ods html close;
```

Program Description

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

```
libname proclib 'SAS-library';
```

Specify the format search library. The SAS system option FMTSEARCH= adds the SAS library PROCLIB to the search path that is used to locate formats.

```
options fmtsearch=(proclib);
```

Specify the ODS output.

```
ods html file="myhtmlfile.html";
```

Specify the report options. By default, PROC REPORT sends its output to the open output destinations.

```
proc report data=grocery;
```

Specify the report columns. The report contains a column for Manager, Department, and Sales.

```
column manager department sales;
```

Define the sort order variables. The values of all variables with the ORDER option in the DEFINE statement determine the order of the rows in the report. In this report, PROC REPORT arranges the rows first by the value of Manager (because it is the first variable in the COLUMN statement) and then by the values of Department. ORDER= specifies the sort order for a variable. This report arranges the rows according to the formatted values of Manager and the internal values of Department (np1, np2, p1, and p2). FORMAT= specifies the formats to use in the report.

```
define manager / order order=formatted format=$mgrfmt.;
define department / order order=internal format=$deptfmt.;
```

Define the analysis variable. Sum calculates the sum statistic for all observations that are represented by the current row. In this report each row represents only one observation. Therefore, the Sum statistic is the same as the value of Sales for that

observation in the input data set. Using Sales as an analysis variable in this report enables you to summarize the values for each group and at the end of the report.

```
define sales / analysis sum format=dollar7.2;
```

Produce a report summary. This BREAK statement produces a default summary after the last row for each manager. PROC REPORT sums the values of Sales for each manager because Sales is an analysis variable that is used to calculate the Sum statistic.

```
break after manager / summarize
                        style=[font_style=italic];
```

Select the observations to process. The WHERE statement selects for the report only the observations for stores in the southeast sector.

```
where sector='se';
```

Specify the title.

```
title 'Sales for the Southeast Sector';
run;
```

Close the ODS destination.

```
ods html close;
```

Output: HTML

Output 51.3 Ordering the Rows in a Report

Sales for the Southeast Sector

Manager	Department	Sales
Jones	Paper	\$40.00
	Canned	\$220.00
	Meat/Dairy	\$300.00
	Produce	\$70.00
<i>Jones</i>		<i>\$630.00</i>
Smith	Paper	\$50.00
	Canned	\$120.00
	Meat/Dairy	\$100.00
	Produce	\$80.00
<i>Smith</i>		<i>\$350.00</i>

Example 3: Using Aliases to Obtain Multiple Statistics for the Same Variable

Features:	PROC REPORT statement options COLUMN statement with aliases COMPUTE statement arguments AFTER DEFINE statement options ORDER= ANALYSIS SUM FORMAT= MAX MIN NOPRINT customizing column headings LINE statement quoted text variable values and formats writing a blank line LIBNAME statement OPTIONS statement WHERE statement TITLE statement automatic macro variables SYSDATE
Data set:	GROCERY
Format:	\$MGRFMT
Format:	\$DEPTFMT

Details

The customized summary at the end of this report displays the minimum and maximum values of Sales over all departments for stores in the southeast sector. To determine these values, PROC REPORT needs the MIN and MAX statistic for Sales in every row of the report. However, to keep the report simple, the display of these statistics is suppressed.

Program

```

libname proclib 'SAS-library';
options fmtsearch=(proclib);
ods html file="myhtmlfile.html";
proc report data=grocery;
    column manager department sales
           sales=salesmin
           sales=salesmax;
    define manager / order
           order=formatted
           format=$mgrfmt.
           'Manager';
    define department / order
           order=internal
           format=$deptfmt.
           'Department';

    define sales / analysis sum format=dollar7.2 'Sales';
    define salesmin / analysis min noprint;
    define salesmax / analysis max noprint;

    compute after;
        line 'Departmental sales ranged from'
            salesmin dollar7.2 " " 'to' " " salesmax dollar7.2;
    endcomp;

    where sector='se';

    title 'Sales for the Southeast Sector';
    title2 "for &sysdate";
run;
ods html close;

```

Program Description

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

```
libname proclib 'SAS-library';
```

Specify the format search library. The SAS system option FMTSEARCH= adds the SAS library PROCLIB to the search path that is used to locate formats.

```
options fmtsearch=(proclib);
```

Specify the ODS output.

```
ods html file="myhtmlfile.html";
```

Specify the report options. By default, PROC REPORT sends its output to the open output destinations.

```
proc report data=grocery;
```

Specify the report columns. The report contains columns for Manager and Department. It also contains three columns for Sales. The column specifications SALES=SALESMIN and SALES=SALES MAX create aliases for Sales. These aliases enable you to use a separate definition of Sales for each of the three columns.

```
column manager department sales
       sales=salesmin
       sales=salesmax;
```

Define the sort order variables. The values of all variables with the ORDER option in the DEFINE statement determine the order of the rows in the report. In this report, PROC REPORT arranges the rows first by the value of Manager (because it is the first variable in the COLUMN statement) and then by the values of Department. The ORDER= option specifies the sort order for a variable. This report arranges the values of Manager by their formatted values and arranges the values of Department by their internal values (np1, np2, p1, and p2). FORMAT= specifies the formats to use in the report. Text in quotation marks specifies column headings.

```
define manager / order
              order=formatted
              format=$mgrfmt.
              'Manager';
define department / order
              order=internal
              format=$deptfmt.
              'Department';
```

Define the analysis variable. The value of an analysis variable in any row of a report is the value of the statistic that is associated with it (in this case Sum), calculated for all observations that are represented by that row. In a detail report each row represents only one observation. Therefore, the Sum statistic is the same as the value of Sales for that observation in the input data set.

```
define sales / analysis sum format=dollar7.2 'Sales';
```

Define additional analysis variables for use in the summary. These DEFINE statements use aliases from the COLUMN statement to create separate columns for the MIN and MAX statistics for the analysis variable Sales. NOPRINT suppresses the printing of these statistics. Although PROC REPORT does not print these values in columns, it has access to them so that it can print them in the summary.

```
define salesmin / analysis min noprint;
define salesmax / analysis max noprint;
```

Produce a customized summary. This COMPUTE statement begins a compute block that executes at the end of the report. The line statement writes the text shown in the quotation marks, the value of Salesmin with the DOLLAR7.2 format, a space in quotation marks, the text in quotation marks "to", a space, the value of Salesmax with the DOLLAR7.2 format. (Note that the program must reference the variables by their aliases.) The ENDCOMP statement ends the compute block.

```
compute after;
  line 'Departmental sales ranged from'
      salesmin dollar7.2 " " 'to' " " salesmax dollar7.2;
endcomp;
```

Select the observations to process. The WHERE statement selects for the report only the observations for stores in the southeast sector.

```
where sector='se';
```

Specify the titles. SYSDATE is an automatic macro variable that returns the date on which the SAS job or SAS session began. The TITLE2 statement uses double rather than single quotation marks so that the macro variable resolves.

```
title 'Sales for the Southeast Sector';
title2 "for &sysdate";
run;
```

Close the ODS destination.

```
ods html close;
```

Output: HTML

Output 51.4 Using Aliases to Obtain Multiple Statistics for the Same Variable

Sales for the Southeast Sector for 27MAY14		
Manager	Department	Sales
Jones	Paper	\$40.00
	Canned	\$220.00
	Meat/Dairy	\$300.00
	Produce	\$70.00
Smith	Paper	\$50.00
	Canned	\$120.00
	Meat/Dairy	\$100.00
	Produce	\$80.00
Departmental sales ranged from \$40.00 to \$300.00		

Example 4: Displaying Multiple Statistics for One Variable

Features:

- PROC REPORT statement options
- COLUMN statement
 - specifying statistics for stacked variables
- DEFINE statement options

FORMAT=
 GROUP
 LIBNAME statement
 OPTIONS statement
 TITLE statement
 Data set: **GROCERY**
 Format: **\$MGRFMT**

Details

The report in this example displays six statistics for the sales for each manager's store.

Program

```

libname proclib 'SAS-library';
options fmtsearch=(proclib);
ods html file="myhtmlfile.html";
proc report data=grocery;
  column sector manager (Sum Min Max Range Mean Std),sales;
  define manager / group format=$mgrfmt.;
  define sector / group format=$ctrfmt.;
  define sales / format=dollar11.2 ;
  title 'Sales Statistics for All Sectors';
run;
ods html close;

```

Program Description

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

```
libname proclib 'SAS-library';
```

Specify the format search library. The SAS system option FMTSEARCH= adds the SAS library PROCLIB to the search path that is used to locate formats.

```
options fmtsearch=(proclib);
```

Specify the ODS output.

```
ods html file="myhtmlfile.html";
```

Specify the report options.

```
proc report data=grocery;
```

Specify the report columns. This COLUMN statement creates a column for Sector, Manager, and each of the six statistics that are associated with Sales.

```
column sector manager (Sum Min Max Range Mean Std),sales;
```

Define the group variables and the analysis variable. In this report, Sector and Manager are group variables. Each detail row of the report consolidates the information for all observations with the same values of the group variables. FORMAT= specifies the formats to use in the report.

```
define manager / group format=$mgrfmt.;
define sector / group format=$sctrfmt.;
define sales / format=dollar11.2 ;
```

Specify the title.

```
title 'Sales Statistics for All Sectors';
run;
```

Close the ODS destination.

```
ods html close;
```

Output: HTML

Output 51.5 *Displaying Multiple Statistics for One Variable*

Sales Statistics for All Sectors							
		Sum	Min	Max	Range	Mean	Std
Sector	Manager	Sales	Sales	Sales	Sales	Sales	Sales
Northeast	Alomar	\$786.00	\$86.00	\$420.00	\$334.00	\$196.50	\$156.57
	Andrews	\$1,045.00	\$125.00	\$420.00	\$295.00	\$261.25	\$127.83
Northwest	Brown	\$598.00	\$45.00	\$250.00	\$205.00	\$149.50	\$105.44
	Pelfrey	\$746.00	\$45.00	\$420.00	\$375.00	\$186.50	\$170.39
	Reveiz	\$1,110.00	\$30.00	\$600.00	\$570.00	\$277.50	\$278.61
Southeast	Jones	\$630.00	\$40.00	\$300.00	\$260.00	\$157.50	\$123.39
	Smith	\$350.00	\$50.00	\$120.00	\$70.00	\$87.50	\$29.86
Southwest	Adams	\$695.00	\$40.00	\$350.00	\$310.00	\$173.75	\$141.86
	Taylor	\$353.00	\$50.00	\$130.00	\$80.00	\$88.25	\$42.65

Example 5: Consolidating Multiple Observations into One Row of a Report

Features:	PROC REPORT statement options COLUMN statement DEFINE statement options ANALYSIS GROUP SUM FORMAT= BREAK statement options AFTER SUMMARIZE SUPPRESS CALL DEFINE statement COMPUTE statement arguments AFTER CALL DEFINE statement LINE statement quoted text variable values LIBNAME statement OPTIONS statement TITLE statement WHERE statement
Data set:	GROCERY
Format:	\$MGRFMT
Format:	\$DEPTFMT

Details

This example creates a summary report that does the following:

- consolidates information for each combination of Sector and Manager into one row of the report
- contains default summaries of sales for each sector
- contains a customized summary of sales for all sectors
- uses one format for sales in detail rows and a different format in summary rows

- uses customized column headings

Program

```
libname proclib 'SAS-library';
options fmtsearch=(proclib);
ods html file="myhtmlfile.html";
proc report data=grocery;
    column sector manager sales;
    define sector / group format=$sctrfmt.'Sector';
    define manager / group format=$mgrfmt.'Manager';
    define sales / analysis sum format=comma10.2 'Sales';
    break after sector / summarize
        style=[font_style=italic]
        suppress;
    compute after;
        line 'Combined sales for the northern sectors were '
            sales.sum dollar9.2 '.';
    endcomp;
    compute sales;
        if _break_ ne ' ' then
            call define(_col_, "format", "dollar11.2");
    endcomp;
    where sector contains 'n';
    title 'Sales Figures for Northern Sectors';
run;
ods html close;
```

Program Description

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

```
libname proclib 'SAS-library';
```

Specify the format search library. The SAS system option FMTSEARCH= adds the SAS library PROCLIB to the search path that is used to locate formats.

```
options fmtsearch=(proclib);
```

Specify the ODS output.

```
ods html file="myhtmlfile.html";
```

Specify the report options. By default, PROC REPORT sends its output to the open output destinations.

```
proc report data=grocery;
```

Specify the report columns. The report contains columns for Sector, Manager, and Sales.

```
column sector manager sales;
```

Define the group and analysis variables. In this report, Sector and Manager are group variables. Sales is an analysis variable that is used to calculate the Sum statistic. Each detail row represents a set of observations that have a unique combination of formatted values for all group variables. The value of Sales in each detail row is the sum of Sales for all observations in the group. FORMAT= specifies the format to use in the report. Text in quotation marks in a DEFINE statement specifies the column heading.

```
define sector / group format=$sctrfmt.'Sector';
define manager / group format=$mgrfmt.'Manager';
define sales / analysis sum format=comma10.2 'Sales';
```

Produce a report summary. This BREAK statement produces a default summary after the last row for each sector. SUMMARIZE writes the value of Sales in the summary line. The summary value is for each sector. It sums all managers in the sector sales. SUPPRESS prevents PROC REPORT from displaying the value of Sector in the summary line. The summary lines are italicized.

```
break after sector / summarize
                    style=[font_style=italic]
                    suppress;
```

Produce a customized summary. This compute block creates a customized summary at the end of the report. The LINE statement writes the quoted text and the value of Sales.sum (with a format of DOLLAR9.2) in the summary. An ENDCOMP statement must end the compute block.

```
compute after;
  line 'Combined sales for the northern sectors were '
      sales.sum dollar9.2 '.';
endcomp;
```

Specify a format for the summary rows. In detail rows, PROC REPORT displays the value of Sales with the format that is specified in its definition (COMMA10.2). The compute block specifies an alternate format to use in the current column on summary rows. Summary rows are identified as a value other than a blank for _BREAK_.

```
compute sales;
  if _break_ ne ' ' then
    call define(_col_, "format", "dollar11.2");
endcomp;
```

Select the observations to process. The WHERE statement selects for the report only the observations for stores in the northeast and northwest sectors. The TITLE statement specifies the title.

```
where sector contains 'n';
```

Specify the title.

```
title 'Sales Figures for Northern Sectors';
run;
```

Close the ODS destination.

```
ods html close;
```

Output: HTML

Output 51.6 Consolidating Multiple Observations into One Row of a Report

Sales Figures for Northern Sectors		
Sector	Manager	Sales
Northeast	Alomar	786.00
	Andrews	1,045.00
		\$1,831.00
Northwest	Brown	598.00
	Pelfrey	746.00
	Reveiz	1,110.00
		\$2,454.00
Combined sales for the northern sectors were \$4,285.00		

Example 6: Creating a Column for Each Value of a Variable

Features:

- PROC REPORT statement options
 - SPLIT=
- COLUMN statement
 - stacking variables
- DEFINE statement options
 - GROUP
 - ACROSS
 - ANALYSIS
 - COMPUTED
 - SUM
 - FORMAT=
- COMPUTE statement arguments
 - with a computed variable as *report-item*
 - AFTER
- LINE statement
- ENDCOMP statement
- LIBNAME statement

OPTIONS statement

TITLE statement

WHERE statement

Data set: GROCERY

Format: \$SCTRFMT

Format: \$MGRFMT

Format: \$DEPTFMT

Details

The report in this example does the following:

- consolidates multiple observations into one row
- contains a column for each value of Department that is selected for the report (the departments that sell perishable items)
- contains a variable that is not in the input data set
- uses customized column headings, some of which contain blanks
- uses variable values in a customized summary

Program

```
libname proclib 'SAS-library';
options fmtsearch=(proclib);
ods html file="myhtmlfile.html";
proc report data=grocery split='*';
    column sector manager department,sales perish;
    define sector / group format=$sctrfmt. 'Sector';
    define manager / group format=$mgrfmt. 'Manager* ';
    define department / across format=$deptfmt. 'Department';
    define sales / analysis sum format=dollar11.2 ' ';
    define perish / computed format=dollar11.2
        'Perishable*Total';
    compute perish;
        perish=sum(_c3_, _c4_);
    endcomp;
    compute after;
        line 'Combined sales for meat and dairy : '
            _c3_ dollar11.2 ' ';
        line 'Combined sales for produce : '
```

```

        _c4_ dollar11.2 '';
    line 'Combined sales for all perishables: '
        _c5_ dollar11.2 '';
    endcomp;

    where sector contains 'n'
        and (department='p1' or department='p2');

    title 'Sales Figures for Perishables in Northern Sectors';
run;

ods html close;

```

Program Description

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

```
libname proclib 'SAS-library';
```

Specify the format search library. The SAS system option FMTSEARCH= adds the SAS library PROCLIB to the search path that is used to locate formats.

```
options fmtsearch=(proclib);
```

Specify the ODS output.

```
ods html file="myhtmlfile.html";
```

Specify the report options. By default, PROC REPORT sends its output to the open output destinations. SPLIT= defines the split character as an asterisk (*) because the default split character (/) is part of the name of a department.

```
proc report data=grocery split='*';
```

Specify the report columns. Department and Sales are separated by a comma in the COLUMN statement, so they collectively determine the contents of the column that they define. Each item generates a heading, but the heading for Sales is set to blank in its definition. Because Sales is an analysis variable, its values fill the cells that are created by these two variables.

```

column sector manager department,sales perish;

define sector / group format=$sctrfmt. 'Sector';
define manager / group format=$mgrfmt. 'Manager* ';

```

Define the across variable. PROC REPORT creates a column and a column heading for each formatted value of the across variable Department. PROC REPORT orders the columns by these values. PROC REPORT also generates a column heading that spans all these columns. Quoted text in the DEFINE statement for Department customizes this heading.

```
define department / across format=$deptfmt. 'Department';
```

Define the analysis variable. Sales is an analysis variable that is used to calculate the sum statistic. In each case, the value of Sales is the sum of Sales for all observations in one department in one group. (In this case, the value represents a single observation.)

```
define sales / analysis sum format=dollar11.2 ' ';
```

Define the computed variable. The COMPUTED option indicates that PROC REPORT must compute values for Perish. You compute the variable's values in a compute block that is associated with Perish.

```
define perish / computed format=dollar11.2
    'Perishable*Total';
```

Calculate values for the computed variable. This compute block computes the value of Perish from the values for the Meat/Dairy department and the Produce department. Because the variables Sales and Department collectively define these columns, there is no way to identify the values to PROC REPORT by name. Therefore, the assignment statement uses column numbers to unambiguously specify the values to use. Each time PROC REPORT needs a value for Perish, it sums the values in the third and fourth columns of that row of the report.

```
compute perish;
    perish=sum(_c3_, _c4_);
endcomp;
```

Produce a customized summary. This compute block creates a customized summary at the end of the report. LINE statements write the quoted text in the specified columns and the values of the variables _C3_, _C4_, and _C5_ with the DOLLAR11.2 format.

```
compute after;
    line 'Combined sales for meat and dairy : '
        _c3_ dollar11.2 '';
    line 'Combined sales for produce : '
        _c4_ dollar11.2 '';
    line 'Combined sales for all perishables: '
        _c5_ dollar11.2 '';
endcomp;

where sector contains 'n'
    and (department='p1' or department='p2');
```

Specify the title.

```
title 'Sales Figures for Perishables in Northern Sectors';
run;
```

Close the ODS destination.

```
ods html close;
```

Output: HTML

Output 51.7 Creating a Column for Each Value of a Variable

Sales Figures for Perishables in Northern Sectors				
		Department		
		Meat/Dairy	Produce	
Sector	Manager			Perishable Total
Northeast	Alomar	\$190.00	\$86.00	\$276.00
	Andrews	\$300.00	\$125.00	\$425.00
Northwest	Brown	\$250.00	\$73.00	\$323.00
	Pelfrey	\$205.00	\$76.00	\$281.00
	Reveiz	\$600.00	\$30.00	\$630.00
Combined sales for meat and dairy: \$1,545.00 Combined sales for produce: \$390.00 Combined sales for all perishables: \$1,935.00				

Example 7: Writing a Customized Summary on Each Page

Features:

- PROC REPORT statement options
- COLUMN statement
- DEFINE statement options
 - NOPRINT
 - GROUP
 - COMPUTED
 - ANALYSIS
- COMPUTE statement arguments
 - BEFORE
 - AFTER
 - BEFORE _PAGE_
 - STYLE=
 - TEXT=
- BREAK statement options
 - PAGE
 - SUMMARIZE
 - AFTER
 - STYLE=

LINE statement
 LIBNAME statement
 OPTIONS statement
 TITLE statement
 Data set: GROCERY
 Format: \$SCTRFMT
 Format: \$MGRFMT
 Format: \$DEPTFMT

Details

The report in this example displays a record of one day's sales for each store. The rows are arranged so that all the information about one store is together, and the information for each store begins on a new page. Some variables appear in columns. Others appear only in the page heading that identifies the sector and the store's manager.

The heading that appears at the top of each page is created with the `_PAGE_` argument in the `COMPUTE` statement.

Profit is a computed variable based on the value of Sales and Department.

The text that appears at the bottom of the page depends on the total of Sales for the store. Only the first two pages of the report appear here.

Program

```

libname proclib 'SAS-library';
options fmtsearch=(proclib);
ods html file="myhtmlfile.html";
proc report data=grocery;
    title 'Sales for Individual Stores';
    column sector manager department sales Profit;
    define sector / group noprint;
    define manager / group noprint;
    define profit / computed format=dollar11.2;
    define sales / analysis sum format=dollar11.2;
    define department / group format=$deptfmt.;

    compute profit;
        if department='np1' or department='np2'
            then profit=0.4*sales.sum;
        else profit=0.25*sales.sum;
    endcomp;

```

```

compute before _page_ / style={just=left};
  line sector $sctrfmt. ' Sector';
  line 'Store managed by ' manager $mgrfmt.;
  endcomp;

break after manager / summarize style=[font_style=italic] page;

compute after manager;

  length text $ 35;

  if sales.sum lt 500 then
    text='Sales are below the target region.';
  else if sales.sum ge 500 and sales.sum lt 1000 then
    text='Sales are in the target region.';
  else if sales.sum ge 1000 then
    text='Sales exceeded goal!';
  line text $35.;
endcomp;
run;

ods html close;

```

Program Description

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

```
libname proclib 'SAS-library';
```

Specify the format search library. The SAS system option FMTSEARCH= adds the SAS library PROCLIB to the search path that is used to locate formats.

```
options fmtsearch=(proclib);
```

Specify the ODS output.

```
ods html file="myhtmlfile.html";
```

Specify the report options. By default, PROC REPORT sends its output to the open output destinations.

```
proc report data=grocery;
```

Specify the title.

```
title 'Sales for Individual Stores';
```

Specify the report columns. The report contains a column for Sector, Manager, Department, Sales, and Profit, but the NOPRINT option suppresses the printing of the columns for Sector and Manager. The page heading (created later in the program) includes their values. To get these variable values into the page heading, Sector and Manager must be in the COLUMN statement.

```
column sector manager department sales Profit;
```

Define the group, computed, and analysis variables. In this report, Sector, Manager, and Department are group variables. Each detail row of the report consolidates the information for all observations with the same values of the group

variables. Profit is a computed variable whose values are calculated in the next section of the program. FORMAT= specifies the formats to use in the report.

```
define sector / group noprint;
define manager / group noprint;
define profit / computed format=dollar11.2;
define sales / analysis sum format=dollar11.2;
define department / group format=$deptfmt.;
```

Calculate the computed variable. Profit is computed as a percentage of Sales. For nonperishable items, the profit is 40% of the sale price. For perishable items the profit is 25%. Notice that in the compute block, you must reference the variable Sales with a compound name (Sales.sum) that identifies both the variable and the statistic that you calculate with it.

```
compute profit;
  if department='np1' or department='np2'
    then profit=0.4*sales.sum;
  else profit=0.25*sales.sum;
endcomp;
```

Create a customized page heading. This compute block executes at the top of each page, after PROC REPORT writes the title. It writes the page heading for the current manager's store. The STYLE= option left-justifies the text in the LINE statements. The LINE statements write a variable value with the format specified immediately after the variable's name.

```
compute before _page_ / style={just=left};
  line sector $ctrfmt. ' Sector';
  line 'Store managed by ' manager $mgrfmt.;
endcomp;
```

Produce a report summary. This BREAK statement creates a default summary after the last row for each manager. SUMMARIZE writes the value of Sales (the only analysis or computed variable) in the summary line. The PAGE option starts a new page after each default summary so that the page heading that is created in the preceding compute block always pertains to the correct manager. The STYLE= option italicizes the summary information.

```
break after manager / summarize style=[font_style=italic] page;
```

Produce a customized summary. This compute block places conditional text in a customized summary that appears after the last detail row for each manager.

```
compute after manager;
```

Specify the length of the customized summary text. The LENGTH statement assigns a length of 35 to the temporary variable TEXT. In this particular case, the LENGTH statement is unnecessary because the longest version appears in the first IF/THEN statement. However, using the LENGTH statement ensures that even if the order of the conditional statements changes, TEXT is long enough to hold the longest version.

```
length text $ 35;
```

Specify the conditional logic for the customized summary text. You cannot use the LINE statement in conditional statements (IF-THEN, IF-THEN/ELSE, and SELECT) because it does not take effect until PROC REPORT has executed all other statements in the compute block. These IF-THEN/ELSE statements assign a

value to TEXT based on the value of Sales.sum in the summary row. A LINE statement writes that variable, whatever its value happens to be.

```
if sales.sum lt 500 then
    text='Sales are below the target region.';
else if sales.sum ge 500 and sales.sum lt 1000 then
    text='Sales are in the target region.';
else if sales.sum ge 1000 then
    text='Sales exceeded goal!';
    line text $35.;
endcomp;
run;
```

Close the ODS destination.

```
ods html close;
```

Output: HTML

Output 51.8 Writing a Customized Summary on Each Page

Sales for Individual Stores

Northeast Sector Store managed by Alomar		
Department	Sales	Profit
Canned	\$420.00	\$168.00
Meat/Dairy	\$190.00	\$47.50
Paper	\$90.00	\$36.00
Produce	\$86.00	\$21.50
	\$786.00	\$196.50
<i>Sales are in the target region.</i>		

Sales for Individual Stores

Northeast Sector Store managed by Andrews		
Department	Sales	Profit
Canned	\$420.00	\$168.00
Meat/Dairy	\$300.00	\$75.00
Paper	\$200.00	\$80.00
Produce	\$125.00	\$31.25
	\$1,045.00	\$261.25
<i>Sales exceeded goal!</i>		

Example 8: Displaying a Calculated Percentage Column in a Report

Features:

- PROC REPORT statement options
- COLUMN statement arguments
 - PCTSUM
 - SUM
- DEFINE statement options

COMPUTED
 STYLE(COLUMN)=
 GROUP
 COMPUTE statement options
 CHAR
 LENGTH=
 RBREAK statement options
 SUMMARIZE
 STYLE=
 LIBNAME statement
 OPTIONS statement
 TITLE statement

Data set: **GROCERY**

Format: **\$MGRFMT**

Format: **\$DEPTFMT**

Details

The summary report in this example shows the total sales for each store and the percentage that these sales represent of sales for all stores. Each of these columns has its own heading. A single heading also spans all the columns.

The report includes a computed character variable, COMMENT, that flags stores with an unusually high percentage of sales.

Program

```

libname proclib 'SAS-library';
options fmtsearch=(proclib);
ods html file="myhtmlfile.html";
proc report data=grocery;
  title;

  column ('Individual Store Sales as a Percent of All Sales'
         sector manager sales,(sum pctsum) comment);

  define manager / group
    format=$mgrfmt.;
  define sector / group
    format=$sctrfmt.;
  define sales / format=dollar11.2
    '';
  define sum / format=dollar9.2
    'Total Sales';

  define pctsum / 'Percent of Sales' format=percent6.;

```

```

define comment / computed style(column)=[cellwidth=2.5in];
compute comment / char length=40;
    if sales.pctsum gt .15 and _break_ = ' '
    then comment='Sales substantially above expectations.';
    else comment=' ';
endcomp;

rbreak after / summarize style=[font_style=italic];
run;

ods html close;

```

Program Description

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

```
libname proclib 'SAS-library';
```

Specify the format search library. The SAS system option FMTSEARCH= adds the SAS library PROCLIB to the search path that is used to locate formats.

```
options fmtsearch=(proclib);
```

Specify the ODS output.

```
ods html file="myhtmlfile.html";
```

Specify the report options. Specify the data set and other options.

```
proc report data=grocery;
    title;
```

Specify the report columns. The COLUMN statement uses the text in quotation marks as a spanning heading. The heading spans all the columns in the report because they are all included in the pair of parentheses that contains the heading. The COLUMN statement associates two statistics with Sales: Sum and Pctsum. The Sum statistic sums the values of Sales for all observations that are included in a row of the report. The Pctsum statistic shows what percentage of Sales sum is for all observations in the report.

```

column ('Individual Store Sales as a Percent of All Sales'
       sector manager sales,(sum pctsum) comment);

```

Define the group and analysis columns. In this report, Sector and Manager are group variables. Each detail row represents a set of observations that have a unique combination of formatted values for all group variables. Sales is, by default, an analysis variable that is used to calculate the Sum statistic. However, because statistics are associated with Sales in the column statement, those statistics override the default. FORMAT= specifies the formats to use in the report. Text between quotation marks specifies the column heading.

```

define manager / group
    format=$mgrfmt.;
define sector / group
    format=$sctrfmt.;
define sales / format=dollar11.2

```

```

        '';
define sum / format=dollar9.2
        'Total Sales';

```

Define the percentage and computed columns. The DEFINE statement for Pctsum specifies a column heading and a format. The PERCENT. format presents the value of Pctsum as a percentage rather than a decimal. The DEFINE statement for COMMENT defines a computed variable and assigns it a column.

```

define pctsum / 'Percent of Sales' format=percent6.;
define comment / computed style(column)=[cellwidth=2.5in];

```

Calculate the computed variable. Options in the COMPUTE statement define COMMENT as a character variable with a length of 40.

```

compute comment / char length=40;

```

Specify the conditional logic for the computed variable. For every store where sales exceed 15% of the sales for all stores, the compute block creates a comment that says **Sales substantially above expectations**. Of course, on the summary row for the report, the value of Pctsum is 100. However, it is inappropriate to flag this row as having exceptional sales. The automatic variable `_BREAK_` distinguishes detail rows from summary rows. In a detail row, the value of `_BREAK_` is blank. The THEN statement executes only on detail rows where the value of Pctsum exceeds 0.15.

```

        if sales.pctsum gt .15 and _break_ = ' '
        then comment='Sales substantially above expectations.';
        else comment=' ';
endcomp;

```

Produce the report summary. This RBREAK statement creates a default summary at the end of the report. SUMMARIZE writes the values of Sales.sum and Sales.pctsum in the summary line. STYLE= italicizes the summary line.

```

        rbreak after / summarize style=[font_style=italic];
run;

```

Close the ODS destination.

```

ods html close;

```


Output: HTML

Output 51.9 Calculating Percentages

Individual Store Sales as a Percent of All Sales				
Sector	Manager	Total Sales	Percent of Sales	comment
Northeast	Alomar	\$786.00	12%	
	Andrews	\$1,045.00	17%	Sales substantially above expectations.
Northwest	Brown	\$598.00	9%	
	Pelfrey	\$746.00	12%	
	Reveiz	\$1,110.00	18%	Sales substantially above expectations.
Southeast	Jones	\$630.00	10%	
	Smith	\$350.00	6%	
Southwest	Adams	\$695.00	11%	
	Taylor	\$353.00	6%	
		\$6,313.00	100%	

Example 9: How PROC REPORT Handles Missing Values

Features:

- PROC REPORT statement options
 - MISSING
- COLUMN statement
 - with the N statistic
- DEFINE statement options
 - STYLE(COLUMN)=
 - GROUP
- RBREAK statement options
 - AFTER
 - SUMMARIZE
 - STYLE=
- LIBNAME statement
- OPTIONS statement
- TITLE statement

Format: `$MGRFMT`

Details

This example illustrates how PROC REPORT handles missing values for group (or order or across) variables with and without the MISSING option. The differences in the reports are apparent if you compare the values of N for each row and compare the totals in the default summary at the end of the report.

Program – Data Set with No Missing Values

```
libname proclib 'SAS-library';

options fmtsearch=(proclib);

data grocmiss;
  input Sector $ Manager $ Department $ Sales @@;
datalines;
se 1 np1 50      . 1 p1 100      se . np2 120      se 1 p2 80
se 2 np1 40      se 2 p1 300      se 2 np2 220      se 2 p2 70
nw 3 np1 60      nw 3 p1 600      . 3 np2 420      nw 3 p2 30
nw 4 np1 45      nw 4 p1 250      nw 4 np2 230      nw 4 p2 73
nw 9 np1 45      nw 9 p1 205      nw 9 np2 420      nw 9 p2 76
sw 5 np1 53      sw 5 p1 130      sw 5 np2 120      sw 5 p2 50
. . np1 40      sw 6 p1 350      sw 6 np2 225      sw 6 p2 80
ne 7 np1 90      ne . p1 190      ne 7 np2 420      ne 7 p2 86
ne 8 np1 200     ne 8 p1 300      ne 8 np2 420      ne 8 p2 125
;

ods html file="myhtmlfile.html";

proc report data=grocmiss;

  column sector manager N sales;

  define sector / group format=$sctrfmt.;
  define manager / group format=$mgrfmt.;
  define sales / format=dollar9.2;

  rbreak after / summarize style=[fontstyle=italic];

  title 'Summary Report for All Sectors and Managers';
run;

ods html close;
```

Program Description

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

```
libname proclib 'SAS-library';
```

Specify the format search library. The SAS system option FMTSEARCH= adds the SAS library PROCLIB to the search path that is used to locate formats.

```
options fmtsearch=(proclib);
```

Create the GROCMISS data set. GROCMISS is identical to GROCERY except that it contains some observations with missing values for Sector, Manager, or both.

```
data grocmis;
  input Sector $ Manager $ Department $ Sales @@;
datalines;
se 1 np1 50      . 1 p1 100    se . np2 120    se 1 p2 80
se 2 np1 40      se 2 p1 300    se 2 np2 220    se 2 p2 70
nw 3 np1 60      nw 3 p1 600    . 3 np2 420    nw 3 p2 30
nw 4 np1 45      nw 4 p1 250    nw 4 np2 230    nw 4 p2 73
nw 9 np1 45      nw 9 p1 205    nw 9 np2 420    nw 9 p2 76
sw 5 np1 53      sw 5 p1 130    sw 5 np2 120    sw 5 p2 50
. . np1 40      sw 6 p1 350    sw 6 np2 225    sw 6 p2 80
ne 7 np1 90      ne . p1 190    ne 7 np2 420    ne 7 p2 86
ne 8 np1 200     ne 8 p1 300    ne 8 np2 420    ne 8 p2 125
;
```

Specify the ODS output.

```
ods html file="myhtmlfile.html";
```

Specify the report options. By default, PROC REPORT sends its output to the open output destinations.

```
proc report data=grocmis;
```

Specify the report columns. The report contains columns for Sector, Manager, the N statistic, and Sales.

```
column sector manager N sales;
```

Define the group and analysis variables. In this report, Sector and Manager are group variables. Sales is, by default, an analysis variable that is used to calculate the Sum statistic. Each detail row represents a set of observations that have a unique combination of formatted values for all group variables. The value of Sales in each detail row is the sum of Sales for all observations in the group. In this PROC REPORT step, the procedure does not include observations with a missing value for the group variable. FORMAT= specifies formats to use in the report.

```
define sector / group format=$sctrfmt.;
define manager / group format=$mgrfmt.;
define sales / format=dollar9.2;
```

Produce a report summary. This RBREAK statement creates a default summary at the end of the report. SUMMARIZE writes the values of N and Sales.sum in the summary line. STYLE= italicizes the summary line.

```
rbreak after / summarize style=[fontstyle=italic];
```

Specify the title.

```
title 'Summary Report for All Sectors and Managers';
run;
```

Close the ODS destination.

```
ods html close;
```

OUTPUT: HTML

Output 51.10 *Output with No Missing Values*

Summary Report for All Sectors and Managers			
Sector	Manager	N	Sales
Northeast	Alomar	3	\$596.00
	Andrews	4	\$1,045.00
Northwest	Brown	4	\$598.00
	Pelfrey	4	\$746.00
	Reveiz	3	\$690.00
Southeast	Jones	4	\$630.00
	Smith	2	\$130.00
Southwest	Adams	3	\$655.00
	Taylor	4	\$353.00
		<i>31</i>	<i>\$5,443.00</i>

Program – Data Set with Missing Values

```
ods html file="myhtmlfile.html";
proc report data=grocmiss missing;
  column sector manager N sales;
  define sector / group format=$ctrfmt.;
  define manager / group format=$mgrfmt.;
  define sales / format=dollar9.2;
  rbreak after / summarize style=[fontstyle=italic];
run;
ods html close;
```

Program Description

Specify the ODS output.

```
ods html file="myhtmlfile.html";
```

Include the missing values. The MISSING option in the second PROC REPORT step includes the observations with missing values for the group variable.

```
proc report data=grocmiss missing;
  column sector manager N sales;
```

```

define sector / group format=$sctrfmt.;
define manager / group format=$mgrfmt.;
define sales / format=dollar9.2;
rbreak after / summarize style=[fontstyle=italic];
run;

```

Close the ODS destination.

```
ods html close;
```

OUTPUT: HTML

Output 51.11 Output with Missing Values

Summary Report for All Sectors and Managers			
Sector	Manager	N	Sales
		1	\$40.00
	Reveiz	1	\$420.00
	Smith	1	\$100.00
Northeast		1	\$190.00
	Alomar	3	\$596.00
	Andrews	4	\$1,045.00
Northwest	Brown	4	\$598.00
	Pelfrey	4	\$746.00
	Reveiz	3	\$690.00
Southeast		1	\$120.00
	Jones	4	\$630.00
	Smith	2	\$130.00
Southwest	Adams	3	\$655.00
	Taylor	4	\$353.00
		36	\$6,313.00

Example 10: Creating an Output Data Set and Storing Computed Variables

Features: PROC REPORT statement options

```

        OUT=
COLUMN
DEFINE statement options
    GROUP
    COMPUTED
    ANALYSIS
    SUM
COMPUTE
Data set options
    WHERE=
LIBNAME statement
OPTIONS statement
TITLE
Data set:  GROCERY
Format:    $MGRFMT

```

Details

This example uses WHERE processing as it builds an output data set. This technique enables you to do WHERE processing after you have consolidated multiple observations into a single row.

Note: This technique is needed because you cannot subset on the results of analysis variables. You cannot subset on a value calculated by PROC REPORT.

The first PROC REPORT step creates a report in which each row represents all the observations from the input data set for a single manager. The second PROC REPORT step builds a report from the output data set.

Program to Create Output Data Set

```

libname proclib 'SAS-library';
options fmtsearch=(proclib);
ods html file="myhtmlfile.html";
proc report data=grocery
    out=profit( where=(sales gt 1000 and _break='') );
    column manager sales manager_pct;
    define manager / group;
    define manager_pct / computed;
    compute before;
        total_sales = sales.sum;
    endcomp;

```

```

compute manager_pct;
  manager_pct = sales.sum /total_sales;
endcomp;
run;

ods html close;

```

Program Description

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

```
libname proclib 'SAS-library';
```

Specify the format search library. The SAS system option FMTSEARCH= adds the SAS library PROCLIB to the search path that is used to locate formats.

```
options fmtsearch=(proclib);
```

Specify the ODS output.

```
ods html file="myhtmlfile.html";
```

Specify the report options and columns. By default, PROC REPORT sends its output to the open output destinations. OUT= creates the output data set PROFIT. The output data set contains a variable for each column in the report (Manager, Sales, and the computed column manager_pct) as well as for the variable _BREAK_, which is not used in this example. Each observation in the data set represents a row of the report. Because Manager is a group variable and Sales is an analysis variable that is used to calculate the Sum statistic, each row in the report (and therefore each observation in the output data set) represents multiple observations from the input data set. In particular, each value of Sales in the output data set is the total of all values of Sales for that manager. The WHERE= data set option in the OUT= option filters those rows as PROC REPORT creates the output data set. Only those observations with sales that exceed \$1,000 become observations in the output data set.

```

proc report data=grocery
  out=profit( where=(sales gt 1000 and _break_='') );
  column manager sales manager_pct;

```

Define the group and analysis variables and compute the values for the percent of sales. The overall sum is placed in a temporary variable, total_sales, and the values for the percent of sales is computed.

```

define manager / group;
define manager_pct / computed;
compute before;
  total_sales = sales.sum;
endcomp;
compute manager_pct;
  manager_pct = sales.sum /total_sales;
endcomp;
run;

```

Close the ODS destination.

```
ods html close;
```

OUTPUT: HTML

Here is the data set created by PROC REPORT. It is used as the input set in the next PROC REPORT step.

Output 51.12 The Output Data Set PROFIT

	Manager	Sales	manager_pct	_BREAK_
1	3	1110	0.1758276572	
2	8	1045	0.1655314431	

Program That Uses the Output Data Set

```
ods html file="myhtmlfile.html";
proc report data=profit;
  column manager sales manager_pct;
  define manager / group format=$mgrfmt.;
  define sales / analysis sum format=dollar11.2;
  define manager_pct / 'Percent of Total Sales' format=percent8.2;
  title 'Managers with Daily Sales';
  title2 'of over';
  title3 'One Thousand Dollars';
run;
ods html close;
```

Program Description

Specify the ODS output.

```
ods html file="myhtmlfile.html";
```

Specify the report options and columns, define the group and analysis columns, and specify the titles. DATA= specifies the output data set from the first PROC REPORT step as the input data set for this report. The TITLE statements specify a title for the report.

```
proc report data=profit;
  column manager sales manager_pct;
```



```

define manager / group format=$mgrfmt.;
define sales / analysis sum format=dollar11.2;
define manager_pct / 'Percent of Total Sales' format=percent8.2;
title 'Managers with Daily Sales';
title2 'of over';
title3 'One Thousand Dollars';
run;

```

Close the ODS destination.

```
ods html close;
```

OUTPUT: HTML

Output 51.13 Report Based on the Output Data Set

Managers with Daily Sales of over One Thousand Dollars		
Manager	Sales	Percent of Total Sales
Andrews	\$1,045.00	16.55%
Reveiz	\$1,110.00	17.58%

Example 11: Using a Format to Create Groups

Features:

- PROC REPORT statement
- DEFINE statement options
 - GROUP
 - ORDER
 - ACROSS
 - ANALYSIS
 - SUM
 - STYLE=
- COMPUTE statement options
 - AFTER
 - STYLE=
- LINE statement
- FORMAT procedure
- LIBNAME statement
- OPTIONS statement
- TITLE

Data set: **GROCERY**

Format: `$MGRFMT`

Details

This example shows how to use formats to control the number of groups that PROC REPORT creates. The program creates a format for Department that classifies the four departments as one of two types: perishable or nonperishable. Consequently, when Department is an across variable, PROC REPORT creates only two columns instead of four. The column heading is the formatted value of the variable.

Program

```
libname proclib 'SAS-library';
options fmtsearch=(proclib);

proc format;
    value $perish 'p1','p2'='Perishable'
                'np1','np2'='Nonperishable';
run;

ods html file="myhtmlfile.html";

proc report data=grocery;

    column manager department,sales sales;

    define manager / group order=formatted
                    format=$mgrfmt.;
    define department / across order=formatted
                       format=$perish. ' ';

    define sales / analysis sum
                  format=dollar9.2 style=[cellwidth=13];

    compute after / style=[font_style=italic];
        line 'Total sales for these stores were: '
              sales.sum dollar9.2;
    endcomp;

title 'Sales Summary for All Stores';
run;

ods html close;
```

Program Description

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

```
libname proclib 'SAS-library';
```

Specify the format search library. The SAS system option FMTSEARCH= adds the SAS library PROCLIB to the search path that is used to locate formats.

```
options fmtsearch=(proclib);
```

Create the \$PERISH. format. PROC FORMAT creates a format for Department. This variable has four different values in the data set, but the format has only two values.

```
proc format;
  value $perish 'p1','p2'='Perishable'
               'np1','np2'='Nonperishable';
run;
```

Specify the ODS output.

```
ods html file="myhtmlfile.html";
```

Specify the report options. By default, the REPORT procedure sends its output to the open output destinations.

```
proc report data=grocery;
```

Specify the report columns. Department and Sales are separated by a comma in the COLUMN statement, so they collectively determine the contents of the column that they define. Because Sales is an analysis variable, its values fill the cells that are created by these two variables. The report also contains a column for Manager and a column for Sales by itself (which is the sales for all departments).

```
column manager department,sales sales;
```

Define the group and across variables. Manager is a group variable. Each detail row of the report consolidates the information for all observations with the same value of Manager. Department is an across variable. PROC REPORT creates a column and a column heading for each formatted value of Department. ORDER=FORMATTED arranges the values of Manager and Department alphabetically according to their formatted values. FORMAT= specifies the formats to use. The empty quotation marks in the definition of Department specify a blank column heading, so no heading spans all the departments. However, PROC REPORT uses the formatted values of Department to create a column heading for each individual department.

```
define manager / group order=formatted
                format=$mgrfmt.;
define department / across order=formatted
                format=$perish. '';
```

Define the analysis variable. Sales is an analysis variable that is used to calculate the Sum statistic. Sales appears twice in the COLUMN statement, and the same definition applies to both occurrences. FORMAT= specifies the format to use in the report. STYLE= specifies the width of the column. Notice that the column headings for the columns that both Department and Sales create are a combination of the heading for Department and the (default) heading for Sales.

```
define sales / analysis sum
              format=dollar9.2 style=[cellwidth=13];
```

Produce a customized summary. This COMPUTE statement begins a compute block that produces a customized summary at the end of the report. The LINE

statement places the quoted text and the value of Sales.sum (with the DOLLAR9.2 format) in the summary. The STYLE= option italicizes the total sales. An ENDCOMP statement must end the compute block.

```
compute after / style=[font_style=italic];
  line 'Total sales for these stores were: '
      sales.sum dollar9.2;
endcomp;
```

Specify the title.

```
title 'Sales Summary for All Stores';
run;
```

Close the ODS destination.

```
ods html close;
```

Output: HTML

Output 51.14 Using a Format to Create Groups

Sales Summary for All Stores			
	Nonperishable	Perishable	
Manager	Sales	Sales	Sales
Adams	\$265.00	\$430.00	\$695.00
Alomar	\$510.00	\$276.00	\$786.00
Andrews	\$620.00	\$425.00	\$1,045.00
Brown	\$275.00	\$323.00	\$598.00
Jones	\$260.00	\$370.00	\$630.00
Pelfrey	\$465.00	\$281.00	\$746.00
Reveiz	\$480.00	\$630.00	\$1,110.00
Smith	\$170.00	\$180.00	\$350.00
Taylor	\$173.00	\$180.00	\$353.00
<i>Total sales for these stores were: \$6,313.00</i>			

Example 12: Using Multilabel Formats

Features:

- PROC REPORT statement
- COLUMN statement
- DEFINE statement options

```

FORMAT=
GROUP
MLF
ORDER=
PRELOADFMT
MEAN
FORMAT procedure option
MULTILABEL
NOTSORTED
LIBNAME statement
OPTIONS statement
TITLE statement
WHERE statement

```

Details

This example uses a multilabel format to create a report that does the following:

- shows how to specify a multilabel format in the VALUE statement of PROC FORMAT
- shows how to activate multilabel format processing using the MLF option with the DEFINE statement
- shows how using NOTSORTED and PRELOADFMT use the sort order shown in PROC FORMAT.

Program

```

proc format;
  value agelfmt (multilabel notsorted)
    11='11'
    12='12'
    13='13'
    14='14'
    15='15'
    16='16'
    11-12='11 or 12'
    13-14='13 or 14'
    15-16='15 or 16'
    low-13='13 and below'
    14-high='14 and above' ;
run;

ods html file="example.html";

title "GROUP Variable with MLF Option";

proc report data=sashelp.class;

```

```

col age ('Mean' height weight);

define age / group mlf format=agelfmt. 'Age Group' order=data
preloadfmt;
define height / mean format=6.2 'Height (in.)';
define weight / mean format=6.2 'Weight (lbs.)';
run;

ods html close;

```

Program Description

Create the AGE1FMT. format. The FORMAT procedure creates a multilabel format for ages by using the MULTILABEL option. A multilabel format is one in which multiple labels can be assigned to the same value. Each value is represented in the table for each range in which it occurs. Use the NOTSORTED to ensure that the sort order is kept.

```

proc format;
  value agelfmt (multilabel notsorted)
    11='11'
    12='12'
    13='13'
    14='14'
    15='15'
    16='16'
    11-12='11 or 12'
    13-14='13 or 14'
    15-16='15 or 16'
    low-13='13 and below'
    14-high='14 and above' ;
run;

```

Specify the ODS output.

```
ods html file="example.html";
```

Specify a title.

```
title "GROUP Variable with MLF Option";
```

Specify the report options. By default, the REPORT procedure sends its output to the open output destination.

```
proc report data=sashelp.class;
```

Specify the report columns. The report contains a column for Age, Height, and Weight. The Mean of the Height and Weight are calculated.

```
col age ('Mean' height weight);
```

Define the variables. AGE is a group variable. The AGE variable uses the MLF option to activate multilabel format processing. MLF should be used only with group and across variables. FORMAT= specifies that the multilabel format, AGE1FMT. is used. The ORDER=DATA option along with the NOTSORTED option on PROC FORMAT, keeps the desired sort order. Each detail row of the report consolidates the information for all observations with the same values of the group

variables. The mean is calculated for the HEIGHT and WEIGHT column values. PRELOADFMT option specifies that the format is preloaded for the variable.

```
define age / group mlf format=agelfmt. 'Age Group' order=data
preloadfmt;
define height / mean format=6.2 'Height (in.)';
define weight / mean format=6.2 'Weight (lbs.)';
run;
```

Close the ODS destination.

```
ods html close;
```

Output: HTML

Output 51.15 Using Multilabel Formats

GROUP Variable with MLF Option		
	Mean	
Age Group	Height (in.)	Weight (lbs.)
11	54.40	67.75
12	59.44	94.40
13	61.43	88.67
14	64.90	101.88
15	65.63	117.38
16	72.00	150.00
11 or 12	58.00	86.79
13 or 14	63.41	96.21
15 or 16	66.90	123.90
13 and below	59.03	87.35
14 and above	66.01	114.11

Example 13: Specifying Style Elements for ODS Output in Multiple Statements

Features:

- PROC REPORT statement options STYLE= option
- STYLE=
- DEFINE statement options
- ORDER

```

STYLE=
BREAK statement options
  AFTER
  SUMMARIZE
COMPUTE statement options
  AFTER
  STYLE=
CALL DEFINE
  STYLE=
LINE statement
FORMAT procedure
LIBNAME statement
OPTIONS statement
TITLE statement
WHERE statement
ODS PDF statement
ODS RTF statement

```

```

Data set:  GROCERY
Format:    $MGRFMT
Format:    $DEPTFMT

```

Details

This example creates HTML, PDF, and RTF files and sets the style elements for each location in the report in the PROC REPORT statement. It then overrides some of these settings by specifying style elements in other statements. For more information, see [“Style Elements and Style Attributes for Table Regions”](#) on page 1842.

Program

```

libname proclib 'SAS-library';
options fmtsearch=(proclib);
ods html file="myhtmlfile.html";
ods pdf file='external-PDF-file';
ods rtf file='external-RTF-file';

proc report data=grocery
  style(report)=[cellspacing=5 borderwidth=10 bordercolor=blue]
  style(header)=[color=yellow
                 fontstyle=italic fontsize=6]

```



```

style(column)=[color=moderate brown
               fontfamily=helvetica fontsize=4]
style(lines)=[color=white backgroundcolor=black
              fontstyle=italic fontweight=bold fontsize=5]
style(summary)=[color=cx3e3d73 backgroundcolor=cxaeadd9
                fontfamily=helvetica fontsize=3 textalign=r];

column manager department sales;

define manager / order
               order=formatted
               format=$mgrfmt.
               'Manager'

               style(header)=[color=white
                              backgroundcolor=black];

define department / order
                  order=internal
                  format=$deptfmt.
                  'Department'

                  style(column)=[fontstyle=italic];

break after manager / summarize;

compute after manager
      / style=[fontstyle=roman fontsize=3 fontweight=bold
               backgroundcolor=white color=black];

      line 'Subtotal for ' manager $mgrfmt. 'is '
           sales.sum dollar7.2 '.';
endcomp;

compute sales;
      if sales.sum>100 and _break_=' ' then
      call define(_col_, "style",
                  "style=[backgroundcolor=yellow
                           fontfamily=helvetica
                           fontweight=bold]");

endcomp;

compute after;
      line 'Total for all departments is: '
           sales.sum dollar7.2 '.';
endcomp;

where sector='se';

title 'Sales for the Southeast Sector';

run;

ods pdf close;
ods rtf close;
ods html close;

```

Program Description

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

```
libname proclib 'SAS-library';
```

Specify the format search library. The SAS system option FMTSEARCH= adds the SAS library PROCLIB to the search path that is used to locate formats.

```
options fmtsearch=(proclib);
```

Specify the ODS output.

```
ods html file="myhtmlfile.html";
```

Specify the ODS output filenames. By opening multiple ODS destinations, you can produce multiple output files in a single execution. HTML output is produced by default. The ODS PDF statement produces output in Portable Document Format (PDF). The ODS RTF statement produces output in Rich Text Format (RTF). The output from PROC REPORT goes to each of these files.

```
ods pdf file='external-PDF-file';
ods rtf file='external-RTF-file';
```

Specify the report options. In this case, SAS writes the output to the traditional procedure output, the HTML body file, and the RTF and PDF files.

```
proc report data=grocery
```

Specify the style attributes for the report. This STYLE= option sets the style element for the structural part of the report. Because no style element is specified, PROC REPORT uses all the style attributes of the default style element for this location except for the ones that are specified here.

```
style(report)=[cellspacing=5 borderwidth=10 bordercolor=blue]
```

Specify the style attributes for the column headings. This STYLE= option sets the style element for all column headings. Because no style element is specified, PROC REPORT uses all the style attributes of the default style element for this location except for the ones that are specified here.

```
style(header)=[color=yellow
               fontstyle=italic fontsize=6]
```

Specify the style attributes for the report columns. This STYLE= option sets the style element for all the cells in all the columns. Because no style element is specified, PROC REPORT uses all the style attributes of the default style element for this location except for the ones that are specified here.

```
style(column)=[color=moderate brown
               fontfamily=helvetica fontsize=4]
```

Specify the style attributes for the compute block lines. This STYLE= option sets the style element for all the LINE statements in all compute blocks. Because no style element is specified, PROC REPORT uses all the style attributes of the default style element for this location except for the ones that are specified here.

```
style(lines)=[color=white backgroundcolor=black
               fontstyle=italic fontweight=bold fontsize=5]
```

Specify the style attributes for the report summaries. This STYLE= option sets the style element for all the default summary lines. Because no style element is specified, PROC REPORT uses all the style attributes of the default style element for this location except for the ones that are specified here.

```
style(summary)=[color=cx3e3d73 backgroundcolor=cxaeadd9
                fontfamily=helvetica fontsize=3 textalign=r];
```

Specify the report columns. The report contains columns for Manager, Department, and Sales.

```
column manager department sales;
```

Define the first sort order variable. In this report Manager is an order variable. PROC REPORT arranges the rows first by the value of Manager (because it is the first variable in the COLUMN statement). ORDER= specifies that values of Manager are arranged according to their formatted values. FORMAT= specifies the format to use for this variable. Text in quotation marks specifies the column headings.

```
define manager / order
                order=formatted
                format=$mgrfmt.
                'Manager'
```

Specify the style attributes for the first sort order variable column heading. The STYLE= option sets the foreground and background colors of the column heading for the Manager column heading.

```
style(header)=[color=white
                backgroundcolor=black];
```

Define the second sort order variable. In this report Department is an order variable. PROC REPORT arranges the rows first by the value of Manager (because it is the first variable in the COLUMN statement), then by the value of Department. ORDER= specifies that values of Department are arranged according to their internal values. FORMAT= specifies the format to use for this variable. Text in quotation marks specifies the column heading.

```
define department / order
                  order=internal
                  format=$deptfmt.
                  'Department'
```

Specify the style attributes for the second sort order variable column. The STYLE= option sets the font of the cells in the column Department to italic. The style attributes for the cells match the ones that were established for the COLUMN location in the PROC REPORT statement.

```
style(column)=[fontstyle=italic];
```

Produce a report summary. The BREAK statement produces a default summary after the last row for each manager. SUMMARIZE writes the values of Sales (the only analysis or computed variable in the report) in the summary line. PROC REPORT sums the values of Sales for each manager because Sales is an analysis variable that is used to calculate the Sum statistic.

```
break after manager / summarize;
```

Produce a customized summary. The COMPUTE statement begins a compute block that produces a customized summary at the end of the report. This STYLE=

option specifies the style element to use for the text that is created by the LINE statement in this compute block. This style element switches the foreground and background colors that were specified for the LINES location in the PROC REPORT statement. It also changes the font style, the font weight, and the font size.

```
compute after manager
  / style=[fontstyle=roman fontsize=3 fontweight=bold
    backgroundcolor=white color=black];
```

Specify the text for the customized summary. The LINE statement places the quoted text and the values of Manager and Sales.sum (with the formats \$MGRFMT. and DOLLAR7.2) in the summary. An ENDCOMP statement must end the compute block.

```
line 'Subtotal for ' manager $mgrfmt. 'is '
      sales.sum dollar7.2 '.';
endcomp;
```

Produce a customized background for the analysis column. This compute block specifies a background color and a bold font for all cells in the Sales column that contain values of 100 or greater and that are not summary lines.

```
compute sales;
  if sales.sum>100 and _break_=' ' then
    call define(_col_, "style",
      "style=[backgroundcolor=yellow
        fontfamily=helvetica
        fontweight=bold]");
endcomp;
```

Produce a customized end-of-report summary. This COMPUTE statement begins a compute block that executes at the end of the report. The LINE statement writes the quoted text and the value of Sales.sum (with the DOLLAR7.2 format). An ENDCOMP statement must end the compute block.

```
compute after;
  line 'Total for all departments is: '
      sales.sum dollar7.2 '.';
endcomp;
```

Select the observations to process. The WHERE statement selects for the report only the observations for stores in the southeast sector.

```
where sector='se';
```

Specify the title.

```
title 'Sales for the Southeast Sector';
run;
```

Close the ODS destinations.

```
ods pdf close;
ods rtf close;
ods html close;
```

Output: HTML

Output 51.16 Style Elements for ODS HTML Output in Multiple Statements

Sales for the Southeast Sector		
Manager	Department	Sales
Jones	Paper	40
	Canned	220
	Meat/Dairy	300
	Produce	70
Jones		630
Subtotal for Jones is \$630.00.		
Smith	Paper	50
	Canned	120
	Meat/Dairy	100
	Produce	80
Smith		350
Subtotal for Smith is \$350.00.		
Total for all departments is: \$980.00.		

Output: PDF

Output 51.17 Style Elements for ODS PDF Output in Multiple Statements

<i>Sales for the Southeast Sector</i>		
<i>Manager</i>	<i>Department</i>	<i>Sales</i>
Jones	Paper	40
	Canned	220
	Meat/Dairy	300
	Produce	70
Jones		630
Subtotal for Jones is \$630.00.		
Smith	Paper	50
	Canned	120
	Meat/Dairy	100
	Produce	80
Smith		350
Subtotal for Smith is \$350.00.		
<i>Total for all departments is: \$980.00.</i>		

Output: RTF

Output 51.18 Style Elements for ODS RTF Output in Multiple Statements

<i>Sales for the Southeast Sector</i>		
Manager	Department	Sales
Jones	Paper	40
	Canned	220
	Meat/Dairy	300
	Produce	70
Jones		630
Subtotal for Jones is \$630.00.		
Smith	Paper	50
	Canned	120
	Meat/Dairy	100
	Produce	80
Smith		350
Subtotal for Smith is \$350.00.		
Total for all departments is: \$980.00.		

Example 14: Using the CELLWIDTH= Style Attribute with PROC REPORT

Features:

- PROC REPORT statement options
- STYLE=
- COLUMN statement
- DEFINE statement options
- STYLE(COLUMN)=
- TITLE statement

Details

This example uses PROC REPORT to create a table and uses ODS style attributes. This example

- sets the cell width of the total report
- defines the cell width for the columns in the ODS output.

Note: CELLWIDTH= and WIDTH= are aliases.

Refer to [“Style Attributes Tables” in SAS Output Delivery System: Procedures Guide](#) for details.

Program

```
ods html file="myhtmlfile.html";
proc report data=sashelp.class;
    col name age sex;
    define name / style(column)=[cellwidth=1in];
    define age / style(column)=[cellwidth=.5in];
    define sex / style(column)=[cellwidth=.5in];
    title "Using the CELLWIDTH= Style with PROC REPORT";
run;
ods html close;
```

Program Description

Specify the ODS output.

```
ods html file="myhtmlfile.html";
```

Specify the cell width for all the columns in the ODS output. By default, the REPORT procedure sends its output to the open output destination. ODS HTML is the output destination used as the default in this example.

```
proc report data=sashelp.class;
```

Specify the Columns to be used.

```
col name age sex;
```

Define the column widths using the CELLWIDTH= style attribute. Define the dimensions of the NAME, AGE, and SEX column using the STYLE= option.

```
define name / style(column)=[cellwidth=1in];
define age / style(column)=[cellwidth=.5in];
```



```
define sex / style(column)=[cellwidth=.5in];
```

Specify a table title. Provide a table name and run the SAS program.

```
title "Using the CELLWIDTH= Style with PROC REPORT";
run;
```

Close the ODS destination.

```
ods html close;
```

Output: HTML

Output 51.19 Using the CELLWIDTH= Style Attributes with PROC REPORT

Using CELLWIDTH= Style with PROC REPORT

Name	Age	Sex
Alfred	14	M
Alice	13	F
Barbara	13	F
Carol	14	F
Henry	14	M
James	12	M
Jane	12	F
Janet	15	F
Jeffrey	13	M
John	12	M
Joyce	11	F
Judy	14	F
Louise	12	F
Mary	15	F
Philip	16	M
Robert	12	M
Ronald	15	M
Thomas	11	M
William	15	M

Example 15: Using STYLE/MERGE in PROC REPORT CALL DEFINE Statement

Features:

- PROC REPORT statement
- COLUMN statement
- COMPUTE statement
- CALL DEFINE statement options
 - STYLE
 - STYLE/MERGE
- TITLE statement

Details

This example uses PROC REPORT to create a table that uses the STYLE/MERGE option in the CALL DEFINE statement.

Program

```
ods html file="myhtmlfile.html";
proc report data=sashelp.class;
col name sex age height weight;
define name--weight / display;
  compute sex;
  if sex = 'M' then
    call define('name', "style", "style=[background=cyan]");
  endcomp;
  compute age;
  if age > 13 then
    call define('name', "style/merge", "style=[color=red]");
  endcomp;
title "Using STYLE/MERGE Style with PROC REPORT";
run;
ods html close;
```

Program Description

Specify the ODS output.

```
ods html file="myhtmlfile.html";
```

Specify the cell width for all the columns in the ODS output. By default, the REPORT procedure sends its output to the open output destination. ODS HTML is the output destination used as the default in this example.

```
proc report data=sashelp.class;
```

Specify the Columns to be used.

```
col name sex age height weight;
```

Display all of the Columns.

```
define name--weight / display;
```

Apply style attributes. Apply the cyan color to the background of the names that are males.

```
compute sex;
  if sex = 'M' then
    call define('name', "style", "style=[background=cyan]");
  endcomp;
```

Merge style attributes. Apply the color red to the names that are over the age 13. That name color is merged with cells that have the cyan background color.

```
compute age;
  if age > 13 then
    call define('name', "style/merge", "style=[color=red]");
  endcomp;
```

Specify a table title. Provide a table name and run the SAS program.

```
title "Using STYLE/MERGE Style with PROC REPORT";
run;
```

Close the ODS destination.

```
ods html close;
```

Output: HTML

Output 51.20 Using STYLE/MERGE

Using STYLE/MERGE				
Name	Sex	Age	Height	Weight
Alfred	M	14	69	112.5
Alice	F	13	56.5	84
Barbara	F	13	65.3	98
Carol	F	14	62.8	102.5
Henry	M	14	63.5	102.5
James	M	12	57.3	83
Jane	F	12	59.8	84.5
Janet	F	15	62.5	112.5
Jeffrey	M	13	62.5	84
John	M	12	59	99.5
Joyce	F	11	51.3	50.5
Judy	F	14	64.3	90
Louise	F	12	56.3	77
Mary	F	15	66.5	112
Philip	M	16	72	150
Robert	M	12	64.8	128
Ronald	M	15	67	133
Thomas	M	11	57.5	85
William	M	15	66.5	112

Example 16: Using STYLE/REPLACE in PROC REPORT CALL DEFINE Statement

Features:

- PROC REPORT statement
- COLUMN statement
- COMPUTE statement
- CALL DEFINE statement options
- STYLE

STYLE/REPLACE TITLE statement

Details

This example uses PROC REPORT to create a table that uses the STYLE/REPLACE option in the CALL DEFINE statement.

Program

```
ods html file="myhtmlfile.html";
proc report data=sashelp.class;
col name sex age height weight;
define name--weight / display;

  compute sex;
    if sex = 'M' then
      call define('name', "style", "style=[background=cyan]");
    endcomp;

  compute age;
    if age > 13 then
      call define('name', "style/replace", "style=[color=red]");
    endcomp;

title "Using STYLE/REPLACE";
run;

ods html close;
```

Program Description

Specify the ODS output.

```
ods html file="myhtmlfile.html";
```

Specify the cell width for all the columns in the ODS output. By default, the REPORT procedure sends its output to the open output destination. ODS HTML is the output destination used as the default in this example.

```
proc report data=sashelp.class;
```

Specify the Columns to be used.

```
col name sex age height weight;
```

Display all of the Columns.

```
define name--weight / display;
```

Apply style attributes. Apply the cyan color to the background of the names that are males.

```
compute sex;  
  if sex = 'M' then  
    call define('name', "style", "style=[background=cyan]");  
  endcomp;
```

Replace style attributes. Apply the color red to the names that are over the age 13. For the names that are over the age of 13, the red name color replaces the background color cyan.

```
compute age;  
  if age > 13 then  
    call define('name', "style/replace", "style=[color=red]");  
  endcomp;
```

Specify a table title. Provide a table name and run the SAS program.

```
title "Using STYLE/REPLACE";  
run;
```

Close the ODS destination.

```
ods html close;
```

Output: HTML

Output 51.21 Using STYLE/REPLACE

Using STYLE/REPLACE				
Name	Sex	Age	Height	Weight
Alfred	M	14	69	112.5
Alice	F	13	56.5	84
Barbara	F	13	65.3	98
Carol	F	14	62.8	102.5
Henry	M	14	63.5	102.5
James	M	12	57.3	83
Jane	F	12	59.8	84.5
Janet	F	15	62.5	112.5
Jeffrey	M	13	62.5	84
John	M	12	59	99.5
Joyce	F	11	51.3	50.5
Judy	F	14	64.3	90
Louise	F	12	56.3	77
Mary	F	15	66.5	112
Philip	M	16	72	150
Robert	M	12	64.8	128
Ronald	M	15	67	133
Thomas	M	11	57.5	85
William	M	15	66.5	112

S3 Procedure

Overview: S3 Procedure	1923
What Does the S3 Procedure Do?	1924
Concepts: S3 Procedure	1924
PROC S3 Configuration	1924
Support for IMDS and IAM Roles	1926
Defining Custom Regions	1927
Using Server-Side Encryption with AWS Data	1928
Transfer Acceleration	1929
Syntax: S3 Procedure	1929
PROC S3 Statement	1930
BUCKET Statement	1935
COPY Statement	1936
CREATE Statement	1937
DELETE Statement	1937
DESTROY Statement	1938
ENCKEY Statement	1938
GET Statement	1940
GETACCEL Statement	1941
GETDIR Statement	1941
INFO Statement	1942
LIST Statement	1942
MKDIR Statement	1943
PUT Statement	1944
PUTDIR Statement	1945
REGION Statement	1946
RMDIR Statement	1947
Examples: S3 Procedure	1948
Example 1: Create a Bucket and Add a File	1948
Example 2: Manage the Contents of a Bucket	1949
Example 3: Manage Encryptions with S3 Data	1950
Example 4: Define a Custom Region with PROC S3	1955

Overview: S3 Procedure

What Does the S3 Procedure Do?

Use the S3 procedure to perform object management for objects in Amazon S3 or in MinIO environments. For example, you can create buckets and add files to S3. Common object types for use with SAS are files and directories.

Before you can use the S3 procedure, you need an Amazon Web Service (AWS) key ID and secret. When using temporary credentials, you also need a security token. For more information, see the [Amazon S3 documentation](#).

When the data that you add to or retrieve from S3 is larger than 5 MB, the procedure creates additional threads. These threads enable parallel processing for faster transfer speeds.

Note: You can use the [S3 action set](#) to interact with S3 objects when you use CAS.

Concepts: S3 Procedure

PROC S3 Configuration

Overview of PROC S3 Configuration

The S3 procedure works with data objects that are stored in Amazon S3 or in MinIO Enterprise Object Store. For information about system requirements, see “[Support for Data Storage in Amazon S3 and MinIO](#)” in *System Requirements for the SAS Viya Platform*.

The S3 procedure reads configuration and security information from configuration files, from an Amazon EC2 instance role, or from options in the PROC S3 statement. When configuration files are used, these configuration files can be either AWS Command-Line Interface (CLI) configuration files or a local PROC S3 configuration file. PROC S3 reads both types of configuration files when it runs.

Options that are specified in AWS CLI configuration files override options that are specified in the PROC S3 configuration file. Options that you specify in the S3 procedure override options that are set in configuration files.

AWS CLI Configuration Files

Unless you specify otherwise, PROC S3 reads AWS CLI configuration files from their default locations in your home directory and uses the default profile. The AWS CLI configuration files are called `config` and `credentials`. You can specify an alternative location for the `config` file with the `AWSCONFIG=` option. You can specify an alternative location for the `credentials` file with the `AWSCREDENTIALS=` option.

The `config` file contains multiple sets of options called *profiles*. Specify which configuration profile to use with the `PROFILE=` option in the PROC S3 statement. Similarly, the `credentials` file contains groups of options in profiles. Specify which credentials profile to use with the `CREDENTIALSPROFILE=` option in the PROC S3 statement.

PROC S3 Configuration File

You can create a local PROC S3 configuration file to use for connection options. The default PROC S3 configuration file is `.tks3.conf`. It is located in your home directory. If the local configuration file with the default name exists, the S3 procedure reads it automatically. If the PROC S3 configuration file uses a different name or is in a different location, specify its name and path with the `CONFIG=` option in the PROC S3 statement.

Note: If an option is specified in the AWS CLI configuration file as well as in the local PROC S3 configuration file, then the value in the AWS CLI configuration file takes precedence.

The PROC S3 configuration file contains case-sensitive name-value pairs that you specify as `name=value`. Do not enclose values in quotation marks. The file can contain one or more sets of the following name-value pairs:

region

specifies the AWS region to connect to. See the [Region argument](#) for the list of valid region values.

keyId

specifies the AWS access key ID. This value is a 20-character, alphanumeric string.

`secret`

specifies the AWS secret access key. This value is a 40-character string.

Note: See “[Securing the PROC S3 Configuration File](#)” for information about restricting access to this information.

`sessionToken`

specifies the session token. Specify this value if you are using temporary AWS security credentials.

Note: See “[Securing the PROC S3 Configuration File](#)” for information about restricting access to this information.

`ssl`

specifies whether SSL or TLS should be used for connections. Specify *true* or *yes* to use SSL or TLS. Any other value deactivates SSL or TLS.

Here is a sample PROC S3 configuration file:

```
ssl=yes
keyID=AKFKI80MEVIM3XJHKEUQ
secret=wb89GergI/3xejxudQugFj5Wi4iqlFJhGpLvYVv
region=usstd
```

Securing the PROC S3 Configuration File

Because the `secret` and `sessionToken` values are confidential, use your operating system to restrict access to the file. For example, you might issue this command to restrict access to just yourself:

```
chmod 700 /u/$user/.tk3.conf
```

Support for IMDS and IAM Roles

Support has been added for EC2 Instance Metadata Service version 2 (IMDSv2). IMDSv1 continues to be supported. When deployed to Amazon EKS, SAS uses IMDS to obtain an IAM role that enables you to access S3 resources.

SAS first attempts to use IMDSv2 to obtain a session token. If using IMDSv2 fails, then IMDSv1 is used.

Specify the following options in your Amazon environment:

- When `http-tokens` is set to `required`, you must use IMDSv2. If `http-tokens` is set to `optional`, then either IMDSv1 or IMDSv2 can be used.
- Set `http_put_response_hop_limit` to 2 when you use IMDSv2.

Defining Custom Regions

You can define custom regions in addition to the predefined list of regions that is supplied with the S3 environment. One way to manage custom regions is by using the [REGION statement](#) with PROC S3. If you define custom regions with PROC S3, the region definition lasts for the duration of your SAS session. You need to redefine the region in subsequent SAS sessions.

Another way to manage custom regions is with the TKS3_CUSTOM_REGION environment variable. You can define this environment variable in the Autoexec file or by using SAS Environment Manager. In this way, the custom region definition is automatically used when you start a subsequent SAS session.

Here is the syntax to set the TKS3_CUSTOM_REGION environment variable:

TKS3_CUSTOM_REGION=<'>*region-name*<'>,<*HTTP-host*>,<*HTTP-port*>,<*SSL-HTTP-port*>,<*SSLRequired*>,<*SSLAllowed*>

region-name

specifies the custom region name. Enclose the name in quotation marks if it contains spaces or special characters.

HTTP-host

specifies the host name of the server for the region.

HTTP-port

specifies the port number of the server when communicating without SSL. Use 0 to use the default port number for the HTTP protocol.

SSL-HTTP-port

specifies the port number of the server when communicating with SSL. Use 0 to use the default port number for the HTTPS protocol.

SSLRequired

specifies whether all communications with the S3 environment must use SSL. Values are TRUE or FALSE.

SSLAllowed

specifies whether communications with the S3 environment can use SSL. Values are TRUE or FALSE.

Interaction This value is ignored if the *SSLRequired* value is TRUE.

Here is a sample setting for this environment variable that uses a BackBlaze B2 server.

```
TKS3_CUSTOM_REGION=us-west-002,s3.us-west-002.backblazeb2.com,0,0,TRUE,TRUE
```

You can define multiple regions by separating sets of values with a colon (:).

Using Server-Side Encryption with AWS Data

Manage Encryption Keys with the ENCKEY Statement

Use the ENCKEY statement to manage encryption keys. Use the ADD option to add a new encryption key. You can then call the named encryption key when you transfer data between the Amazon S3 or MinIO environment and SAS using the GET, GETDIR, PUT, and PUTDIR statements. You can also request a list of currently registered encryption keys with the ENCKEY statement LIST option or delete a previously registered encryption key with the REMOVE option.

You can specify an encryption key in three forms: an Amazon Key Management Service (KMS) key, a customer-supplied server-side encryption key (SSE-C) key represented as a character string, or an SSE-C key represented by a hexadecimal value.

You can specify a KMS key from the Amazon S3 environment. This type of key works with AWS server-side encryption and is managed in the S3 environment as part of the IAM service.

For SSE-C encryption keys, you can specify the key as a 32-byte character string or as a 64-digit hexadecimal value. User-specified keys are then used by the S3 environment to encrypt data using the AES256 encryption algorithm.

As a best practice, manage encryption keys in a separate SAS program. You can then use the %INCLUDE statement to read in your named encryptions and work with the encryption key names. For an example, see [“Example 3: Manage Encryptions with S3 Data” on page 1950](#).

Use Encryption Keys with SAS/ACCESS

After you add an encryption key, you can use that key with SAS/ACCESS Interface to Amazon Redshift. In a LIBNAME statement, use the BL_ENCKEY= option to call an encryption key by the name that you assigned with the ENCKEY statement. This enables encryption during bulk loading or unloading between the Amazon Redshift environment and SAS. Alternatively, you can specify the BL_ENCKEY= data set option when referring to a specific Amazon Redshift data set, such as in a DATA step or in a call to PROC SQL.

Note: Only IAM:KMS keys are supported for SAS/ACCESS Interface to Amazon Redshift.

For more information, see [“Encryption for Bulk Loading or Unloading with Amazon S3” in SAS/ACCESS for Relational Databases: Reference](#).

Transfer Acceleration

Transfer acceleration is a special mode that is used to load or write data. In this mode, an alternative host enables faster data transfer. For more information, see [your AWS documentation](#).

The GET, GETDIR, PUT, and PUTDIR statements take advantage of this faster data transfer method when transfer acceleration is enabled.

Syntax: S3 Procedure

```
PROC S3 <AWSCONFIG="AWS-CLI-file-path"> <CONFIG="local-configuration-file-  
path"> <PROFILE="configuration-profile-name"> <AWSCREDENTIALS="AWS-  
credentials-file-path"> <CREDENTIALSPROFILE="credentials-profile-name">  
<AUTHDOMAIN="AWS-authentication-domain"> <AUTHSCOPE="scope-URI">  
<ROLENAME="IAM-role-name" | ROLEARN="IAM-role-ARN"> <KEYID="AWS-key-  
ID"> <SECRET="AWS-secret"> <SESSION="session-token"> <SSL | NOSSL>  
<REGION="AWS-region">;
```

```
    BUCKET "bucket-name" ACCELERATE | NOACCELERATE;
```

```
    COPY <SRCKEY="key-name"> <ENCKEY="key-name"> <ACL="canned-ACL-  
value"> "source-s3-location" "destination-s3-location";
```

```
    CREATE "bucket-name";
```

```
    DELETE "s3-location";
```

```
    DESTROY "bucket-name";
```

```
    ENCKEY ADD | REPLACE NAME="key-name" ID="key-ID"  
    <CONTEXT="key-context">;
```

```
    GET <ENCKEY="key-name"> "s3-location" "local-file-path";
```

```
    GETACCEL "bucket-name";
```

```
    GETDIR <ENCKEY="key-name"> "s3-location" "local-path";
```

```
    INFO <ENCKEY="key-name"> "s3-location";
```

```
    LIST <_SHORT_> "s3-location" <OUT=libref.data-set>;
```

```
    MKDIR <ACL="canned-ACL-value"> "s3-location";
```

```
    PUT <ENCKEY="key-name"> <ACL="canned-ACL-value"> "local-path" "s3-  
location";
```

```
    PUTDIR <ENCKEY="key-name"> <ACL="canned-ACL-value"> "local-path" "s3-  
location";
```

```
REGION ADD HOST="AWS-server" NAME="region-name" <PORT=port-value>
  <SSLPORT=SSL-port> <SSLALLOWED> <SSLREQUIRED> <REPLACE>;
```

```
RMDIR "s3-location";
```

```
RUN;
```

Statement	Task	Example
PROC S3	Specifies the connection parameters to S3.	Ex. 1, Ex. 2, Ex. 3
BUCKET	Specifies whether to enable transfer acceleration for a bucket.	
COPY	Copies an S3 object to an S3 destination.	Ex. 2, Ex. 3
CREATE	Creates an S3 bucket.	Ex. 1
DELETE	Deletes an S3 location or object.	Ex. 2
DESTROY	Deletes an S3 bucket.	
ENCKEY	Enables you to work with encryption keys.	Ex. 3
GET	Retrieves an S3 object.	Ex. 3
GETACCEL	Retrieves the transfer acceleration status for a bucket.	
GETDIR	Retrieves the contents of an S3 directory.	Ex. 3
INFO	Lists information about an S3 location or object.	
LIST	Lists the contents of an S3 location.	Ex. 1
MKDIR	Specifies a directory to create in an S3 location.	Ex. 2
PUT	Specifies a local object to write to an S3 location.	Ex. 1, Ex. 3
PUTDIR	Specifies a local directory to write to an S3 location.	Ex. 3
REGION	Enables you to add, list, or remove custom regions.	Ex. 4
RMDIR	Deletes a directory from an S3 location.	

PROC S3 Statement

Specifies the connection parameters for connecting to Amazon S3 or MinIO.

Supports: Threaded processing

Syntax

```
PROC S3 <AWSCONFIG="AWS-CLI-file-path"> <CONFIG="local-configuration-file-  
path"> <PROFILE="configuration-profile-name"> <AWSCREDENTIALS="AWS-  
credentials-file-path"> <CREDENTIALSPROFILE="credentials-profile-name">  
<AUTHDOMAIN="AWS-authentication-domain"> <AUTHSCOPE="scope-URI">  
<ROLENAME="IAM-role-name" | ROLEARN="IAM-role-ARN"> <KEYID="AWS-key-  
ID"> <SECRET="AWS-secret"> <SESSION="session-token"> <SSL | NOSSL>  
<REGION="AWS-region">;
```

Optional Arguments

AUTHDOMAIN="AWS-authentication-domain"

specifies an authentication domain to use when you connect to AWS. An authentication domain is typically defined by your administrator. For more information, see [“Credentials Domain: How to \(CLI\)” in SAS Viya Platform: External Credentials](#).

Default AWSAUTH

Note Support for this option was added in 2024.02.

AUTHSCOPE="scope-URI"

specifies the Microsoft Azure resource and permissions that are required for authorizing access. Enclose the scope URI value in single or double quotation marks.

The authentication scope is used when connecting to Amazon S3 using single sign-on capabilities for the SAS Viya platform. This requires an administrator to configure Microsoft Entra ID as an OpenID Connect provider. For more information, see [“Set Up Federated Authentication for Azure and AWS” in SAS Viya Platform: Authentication](#).

Default none

Interaction When you specify AUTHSCOPE=, you must also specify ROLEARN=.

Note Support for this option was added in 2024.12.

Example authscope="storage.azure.com/user_impersonation"

AWSCONFIG="AWS-CLI-file-path"

specifies the path of the AWS CLI configuration file, called config. This file contains connection parameters to access Amazon S3 or MinIO. If a value is specified in the AWS CLI configuration file and in the PROC S3 configuration file, the value in the AWS CLI file takes precedence.

AWS_CREDENTIALS="AWS-credentials-file-path"

specifies the path of the AWS credentials file. This file contains the credentials that are used to access Amazon S3 or MinIO.

CONFIG="local-configuration-file-path"

specifies the path and file name of the PROC S3 configuration file that contains connection parameters to access Amazon S3 or MinIO. If you specify this option, then the specified configuration file is read instead of the default PROC S3 configuration file in the default location.

The default PROC S3 configuration file is file .tks3.conf in your home directory in Linux. You do not need to specify the CONFIG= option if your configuration file is in the default location.

For more information, see ["PROC S3 Configuration File" on page 1925](#).

CREDENTIALS_PROFILE="credentials-profile-name"

specifies the name of the profile that contains the credentials that you want to use to access Amazon S3 or MinIO. Sets of credential options are grouped into profiles. If you do not specify the CREDENTIALS_PROFILE= option, PROC S3 uses the default profile.

KEY_ID="AWS-key-ID"

specifies the AWS access key ID. This value is a 20-character, alphanumeric string. A sample key ID value is AKIAIOSFODNN7EXAMPLE.

PROFILE="profile-name"

specifies the profile to use in the AWS CLI file config. Sets of options are grouped into profiles. If you do not specify the PROFILE= option, PROC S3 uses the default profile.

REGION="AWS-region"

specifies the AWS region to connect to. Here are the predefined region values.

Note: You can specify either the region value for SAS or the AWS region value for this argument.

Table 52.1 Predefined Region Values for S3

SAS Region Value	Description	Corresponding Amazon S3 Region
afcapetown	Africa (Cape Town)	af-south-1
aphongkong	Asia Pacific (Hong Kong)	ap-east-1
aphyderabad	Asia Pacific (Hyderabad)	ap-south-2
apindia	Asia Pacific (India)	ap-south-1
apjakarta	Asia Pacific (Jakarta)	ap-southeast-3
apmelbourne	Asia Pacific (Melbourne)	ap-southeast-4

SAS Region Value	Description	Corresponding Amazon S3 Region
aposaka	Asia Pacific (Osaka)	ap-northeast-3
apseoul	Asia Pacific (Seoul)	ap-northeast-2
apsingapore	Asia Pacific (Singapore)	ap-southeast-1
apsydney	Asia Pacific (Sydney)	ap-southeast-2
aptokyo	Asia Pacific (Tokyo)	ap-northeast-1
cacalgary	Canada (Calgary)	ca-west-1
cacentral	Canada (Central)	ca-central-1
cnbeijing	China (Beijing)	cn-north-1
cnningxia	China (Ningxia)	cn-northwest-1
eufrankfurt	EU (Frankfurt)	eu-central-1
euireland	EU (Ireland)	eu-west-1
eulondon	EU (London)	eu-west-2
eumilan	EU (Milan)	eu-south-1
euparis	EU (Paris)	eu-west-3
euspain	EU (Spain)	eu-south-2
eustockholm	EU (Stockholm)	eu-north-1
euzurich	EU (Zurich)	eu-central-2
fips	US Government Cloud (FIPS)	fips-us-gov-west-1
fipseast	US Government Cloud (FIPS East)	fips-us-gov-east-1
iltelaviv	Israel (Tel Aviv)	il-central-1
mebahrain	Middle East (Bahrain)	me-south-1
meuae	Middle East (UAE)	me-central-1

SAS Region Value	Description	Corresponding Amazon S3 Region
sa	South America (Sao Paulo)	sa-east-1
useast	US East (Virginia)	us-east-1
useastoh	US East (Ohio)	us-east-2
usgov	US Government Cloud	us-gov-west-1
usgoveast	US Government Cloud (East)	us-gov-east-1
usstd	US Standard	us-east-1
uswest	US West (Oregon)	us-west-2
uswestca	US West (California)	us-west-1

ROLEARN="IAM-role-ARN"

specifies an Amazon Resource Name (ARN) that identifies an IAM role. The associated IAM role enables you to get access credentials to S3 via the AssumeRole IAM operation.

PROC S3 uses an EC2 instance and an associated IAM role by default. If the role that you want to use is already associated with the EC2 instance profile where SAS processes are executing, then you do not need to specify ROLEARN=. The role ARN that you specify must already be defined on AWS.

Interaction Specify either the ROLEARN= option or the ROLENAME= option. Do not specify both.

Note Specify this option only to use a role ARN that is not associated with the EC2 instance that you are using.

Example

```
proc s3 rolearn="arn:aws:iam::123456789012:role/s3AccessRole"
      <additional-options>;
```

ROLENAME="IAM-role-name"

specifies an IAM role name that enables you to get access credentials to S3 via the AssumeRole IAM operation.

PROC S3 uses an EC2 instance and an associated IAM role by default. If the role that you want to use is already associated with the EC2 instance profile where SAS processes are executing, then you do not need to specify ROLENAME=. The role name that you specify must already be defined on AWS.

Interaction Specify either the ROLENAME= option or the ROLEARN= option. Do not specify both.

Note Specify this option only to use a role name that is not associated with the EC2 instance that you are using.

Example `proc s3 rolename="s3AccessRole" <additional-options>;`

SECRET="AWS-secret"

specifies the AWS secret access key. This value is a 40-character string. A sample secret access value is wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY.

SESSION="session-token"

specifies the AWS session token. Use this value if you are using temporary AWS security credentials.

For more information, see [“PROC S3 Configuration” on page 1924](#).

SSL | NOSSL

SSL specifies that SSL or TLS should be enabled during data transfer. NOSSL specifies that SSL or TLS should be disabled during data transfer.

You can use the SSL value to override an `ssl=no` setting in the PROC S3 configuration file. If there is no value specified for SSL in the PROC S3 statement or in a PROC S3 configuration file, then SSL or TLS is used during data transfer by default.

Interaction: If you specify NOSSL in the PROC S3 statement, then enabling server-side encryption with the ENCKEY statement fails.

BUCKET Statement

Sets the acceleration transfer mode for the specified bucket.

Syntax

BUCKET *"bucket-name"* [ACCELERATE](#) | [NOACCELERATE](#);

Required Arguments

bucket-name

specifies the name of the S3 bucket for which you are setting transfer acceleration mode. For more information, see [“Transfer Acceleration” on page 1929](#).

ACCELERATE

enables transfer acceleration for the specified bucket.

NOACCELERATE

disables transfer acceleration for the specified bucket.

COPY Statement

Copies an object from an S3 source location to an S3 destination.

Syntax

```
COPY <SRCKEY="key-name"> <ENCKEY="key-name"> <ACL="canned-ACL-value">
"source-s3-location""destination-s3-location";
```

Required Arguments

- source-s3-location**
specifies the S3 location of the object to be copied. Fully qualify the S3 location from the bucket name to the object name.
- destination-s3-location**
specifies the S3 location to which an object should be copied. Fully qualify the S3 location from the bucket name to the object name.

Optional Arguments

ACL="canned-ACL-value"
specifies a canned ACL value that applies to a newly created object in the destination location.

Here are the valid values:

authenticated-read	log-delivery-write
aws-exec-read	private
bucket-owner-full-control	public-read
bucket-owner-read	public-read-write

For more information about these values, see [Amazon S3 Canned ACL documentation](#).

Default	private
Requirement	The destination location that you specify must be within an ACL-enabled S3 bucket.
Note	Support for this option was added in 2024.10.

ENCKEY="key-name"
specifies an encryption key name that identifies the encryption for the created object. The name must match an encryption key name that you specified in an ENCKEY statement.

The ENCKEY= option defaults to no encryption. You can use the value `_DEFAULT_` to use the default encryption key. Your default encryption key is automatically created.

SRCKEY="key-name"

specifies the encryption of the source object. The name must match an encryption key name that you specified in an ENCKEY statement.

Note: This value is not applied to the created object if the ENCKEY= option is not specified.

CREATE Statement

Creates a bucket.

Syntax

CREATE "*bucket-name*";

Required Argument

bucket-name

specifies the name of the S3 bucket that you are creating. The name of the S3 bucket that you create must be unique across S3.

DELETE Statement

Deletes an S3 location or object.

Syntax

DELETE "*s3-location*";

Required Argument

s3-location

specifies the S3 location that you are deleting. Fully qualify the S3 location from the bucket name to the object name.

DESTROY Statement

Deletes an S3 bucket.

Syntax

DESTROY "*bucket-name*";

Required Argument

bucket-name

specifies the name of an S3 bucket that you are deleting. The bucket that you specify must exist in S3 and must be empty.

ENCKEY Statement

Specifies the encryption key for an S3 location.

Syntax

- Form 1: **ENCKEY ADD | REPLACE** NAME="*key-name*" ID="*key-ID*" <CONTEXT="*key-context*">;
- Form 2: **ENCKEY ADD | REPLACE** NAME="*key-name*" KEY="*key-string*" <ALGORITHM="*S3-encryption-algorithm*"> <CONTEXT="*key-context*">;
- Form 3: **ENCKEY ADD | REPLACE** NAME="*key-name*" HEXKEY="*hexadecimal-value*" <ALGORITHM="*S3-encryption-algorithm*"> <CONTEXT="*key-context*">;
- Form 4: **ENCKEY LIST** <NAME="*key-name*">;
- Form 5: **ENCKEY REMOVE** NAME="*key-name*";

Arguments

ADD

indicates that you are adding an encryption key.

Requirement: Specify the HEXKEY=, ID=, or KEY= options with ADD, depending on the type of key that you are using.

ALGORITHM="*S3-encryption-algorithm*"

specifies the S3 encryption algorithm that is used in the S3 location.

The only value that is currently supported is AES256. This value is the default when you do not specify the ALGORITHM= option.

CONTEXT="key-context"

specifies the key context for the encryption key.

HEXKEY="hexadecimal-value"

specifies an encryption key in the form of a 64-digit hexadecimal value.

ID="key-ID"

specifies the key ID for the encryption key.

KEY="key-string"

specifies an encryption key as a 32-byte character string.

LIST

requests a list of defined encryption keys.

NAME="key-name"

specifies the encryption key name. This is a user-friendly name that you can use to refer to an encryption key. The value is case insensitive.

NAME= is required with the ADD, REPLACE, or REMOVE options. When used with the LIST option, NAME= is optional.

REMOVE

removes an encryption key from the list of defined keys.

Requirement: Specify the NAME= option with REMOVE.

REPLACE

specifies that the existing encryption key should be replaced. If not set, any attempt to redefine an encryption key fails.

Requirement: Specify the HEXKEY=, ID=, or KEY= options with REPLACE, depending on the type of key that you are using.

Details

The ENCKEY statement is used to support server-side encryption in an Amazon S3 or MinIO environment. A *MinIO environment* is an S3-compatible object storage platform that can run on AWS or on other platforms.

All Amazon S3 buckets are encryption enabled by default. New objects that are uploaded to S3 are automatically encrypted by using server-side encryption with S3-managed keys (SSE-S3).

MinIO supports server-side encryption using AES-256-GCM. The default encryption behavior is specific to your MinIO environment.

To work with encryption in Amazon S3 or MinIO:

- Use Form 1 of the ENCKEY statement to add or replace an IAM:KMS encryption key.
- Use Form 2 of the ENCKEY statement to add or replace an encryption key in the form of a string. The string must be 32 bytes long.

- Use Form 3 of the ENCKEY statement to add or replace an encryption key expressed as hexadecimal characters. Do not use the “0x” prefix when you specify the hexadecimal key string. The hexadecimal string should be 64 digits long.
- Use Form 4 of the ENCKEY statement to view the list of defined encryption keys.
- Use Form 5 of the ENCKEY statement to remove an encryption key.

Note: If you specify the NOSSL option in the PROC S3 statement, then using encryption with the ENCKEY statement fails.

You must specify one of these options in the ENCKEY statement: ADD, LIST, REMOVE, or REPLACE.

Encryption keys are specified for the duration of your SAS session.

The encryption key `_DEFAULT_` cannot be modified. The `_DEFAULT_` encryption key is defined by the AWS environment.

For more information, see [Protecting Data Using Server-Side Encryption](#) in your AWS documentation.

GET Statement

Retrieves an S3 object.

Note: This statement uses transfer acceleration when it is available. For more information, see [“Transfer Acceleration” on page 1929](#).

Syntax

```
GET <ENCKEY="key-name"> "s3-location" "local-file-path";
```

Required Arguments

s3-location

specifies the name of an S3 object to retrieve. Fully qualify the location from the bucket name to the object name.

local-file-path

specifies a local file in which to store the object. For example, you might set this value to `/u/$USER/myfile.txt`.

Optional Argument

ENCKEY="key-name"

specifies an encryption key name that identifies the encryption to use during data retrieval. The name must match an encryption key name that you specified

in an ENCKEY statement. The name must also correspond to the encryption key that was used when the object was created in the S3 environment. Otherwise, S3 returns an error.

GETACCEL Statement

Retrieves the transfer acceleration status for a bucket.

Syntax

GETACCEL "*bucket-name*";

Required Argument

bucket-name

specifies the name of an S3 bucket for which you want the transfer acceleration status. For more information, see ["Transfer Acceleration" on page 1929](#).

GETDIR Statement

Retrieves the contents of an S3 directory.

Note: This statement uses transfer acceleration when it is available. For more information, see ["Transfer Acceleration" on page 1929](#).

Syntax

GETDIR <ENCKEY="*key-name*"> "*s3-location*" "*local-path*";

Required Arguments

s3-location

specifies the name of an S3 directory to retrieve. Fully qualify the location from the bucket name to the directory.

local-path

specifies a directory that is local to the SAS client. For example, you might set this value to `/u/$USER/`.

Optional Argument

ENCKEY="key-name"

specifies an encryption key name that identifies the encryption to use during data retrieval. The name must match an encryption key name that you specified in an ENCKEY statement.

Note: Encryption fails if any files in the source location do not use the specified encryption key.

INFO Statement

Requests information in the SAS log about an S3 location.

Note: Information about a file includes the region, path, and the date and time at which the file was last modified.

Syntax

INFO <ENCKEY="key-name"> "s3-location";

Required Argument

s3-location

specifies a fully qualified path from the bucket name to the object name.

Optional Argument

ENCKEY="key-name"

specifies an encryption key name. The name must match an encryption key name that you specified in an ENCKEY statement.

Note: The ENCKEY= option is required if any objects in the S3 location are encrypted.

LIST Statement

Requests information in the SAS log about the contents of an S3 location.

Syntax

```
LIST <_SHORT_> "s3-location" <OUT=libref.data-set | "/path/to/output-file" >;
```

Required Argument

s3-location

specifies a fully qualified path from the bucket name to the object name. The object that you specify is typically a container, such as a bucket or a directory.

Optional Arguments

SHORT

specifies that you want a list of object names only.

OUT=libref.data-set | "/path/to/output-file"

specifies a data set or file location that contains the output of the LIST statement.

If you omit the libref, then you must use quotation marks around the data set name, and the data set is stored in the Work library.

Example `list "/myS3location" out=sasuser.list;`

MKDIR Statement

Specifies a directory to create in an S3 location.

Syntax

```
MKDIR <ACL="canned-ACL-value"> "s3-location";
```

Required Argument

s3-location

specifies a fully qualified path from the bucket name to the directory name that you want to create.

Optional Argument

ACL="canned-ACL-value"

specifies a canned ACL value that applies to newly created objects in the specified directory.

Here are the valid values:

authenticated-read log-delivery-write

aws-exec-read	private
bucket-owner-full-control	public-read
bucket-owner-read	public-read-write

For more information about these values, see [Amazon S3 Canned ACL documentation](#).

Default	private
Requirement	The directory that you specify must be within an ACL enabled S3 bucket.
Note	Support for this option was added in 2024.10.

PUT Statement

Specifies a local object to write to an S3 location.

Note: This statement uses transfer acceleration when it is available. For more information, see [“Transfer Acceleration” on page 1929](#).

Syntax

PUT <ENCKEY="*key-name*"> <ACL="*canned-ACL-value*"> "*local-path*" "*s3-location*";

Required Arguments

- local-path***
specifies a file or directory that is local to the SAS client.
- s3-location***
specifies a location in S3. Fully qualify the path from the bucket name to the object name.

Optional Arguments

ACL="*canned-ACL-value*"
specifies a canned ACL value that applies to a new object in the specified S3 location.

Here are the valid values:

authenticated-read	log-delivery-write
aws-exec-read	private
bucket-owner-full-control	public-read
bucket-owner-read	public-read-write

For more information about these values, see [Amazon S3 Canned ACL documentation](#).

Default private

Requirement The location that you specify must be within an ACL enabled S3 bucket.

Note Support for this option was added in 2024.10.

ENCKEY="key-name"

specifies an encryption key name that identifies the encryption to use when storing data in S3. The name must match an encryption key name that you specified in an ENCKEY statement.

Example To specify your default encryption key:

```
put enckey=_default_ "/home/demouser/s3/foo.txt" "/mybucket/foo.txt";
```

PUTDIR Statement

Specifies a local directory to write to an S3 location.

Note: This statement uses transfer acceleration when it is available. For more information, see ["Transfer Acceleration" on page 1929](#).

Syntax

PUTDIR <ENCKEY="key-name"> <ACL="canned-ACL-value"> "local-path" "s3-location";

Required Arguments

local-path

specifies a directory that is local to the SAS client.

s3-location

specifies a location in S3. Fully qualify the path from the bucket name to the object name.

Optional Arguments

ACL="canned-ACL-value"

specifies a canned ACL value that applies to an object in the specified S3 location.

Here are the valid values:

authenticated-read	log-delivery-write
aws-exec-read	private

bucket-owner-full-control	public-read
bucket-owner-read	public-read-write

For more information about these values, see [Amazon S3 Canned ACL documentation](#).

Default	private
Requirement	The location that you specify must be within an ACL enabled S3 bucket.
Note	Support for this option was added in 2024.10.

ENCKEY="key-name"
specifies an encryption key name that identifies the encryption to use when writing to S3. The name must match an encryption key name that you specified in an ENCKEY statement.

Note: All files in the S3 location are encrypted with the specified encryption key.

REGION Statement

Enables you to add, list, or remove custom regions.

Syntax

- Form 1: **REGION** [ADD](#) [HOST](#)="AWS-server" < [REGION](#)="AWS-region-value" > [NAME](#)="region-name" < [PORT](#)=port-value > < [SSLPORT](#)=SSL-port > < [SSLALLOWED](#) > < [SSLREQUIRED](#) > < [REPLACE](#) >;
- Form 2: **REGION** [LIST](#);
- Form 3: **REGION** [REMOVE](#) [NAME](#)="region-name";

Required Arguments

ADD
specifies that you are adding a custom region for the S3 environment.

Note: If you define custom regions with the **REGION ADD** statement, the region definition lasts for the duration of your SAS session. You need to redefine the region in subsequent SAS sessions. As an alternative, you can use the `TKS3_CUSTOM_REGION` environment variable. For more information, see ["Defining Custom Regions" on page 1927](#).

HOST="AWS-server"

specifies the server that PROC S3 connects to on AWS.

Example `host="myhost.amazonaws.com"`

NAME="region-name"

specifies a unique name for the region that you are adding or removing.

LIST

requests a list of all defined S3 regions.

REMOVE

specifies to remove the region that you identify with the NAME= option.

Restriction You cannot remove any of the predefined S3 regions. See the [REGION=](#) argument for the PROC S3 statement for the list of predefined region values.

Optional Arguments

PORT=port-value

specifies the HTTP port value to use to connect to S3 without SSL.

If you do not specify a port value, the default value is used.

REGION="AWS-region-code"

specifies the region code for the region that you are adding or replacing.

REPLACE

indicates that a custom region should be added and should overwrite a region of the same name, if it exists.

Restriction You cannot replace any of the predefined regions for S3. See the [REGION=](#) argument for the PROC S3 statement for the list of predefined region values.

SSLALLOWED

indicates that SSL is allowed when communicating with the S3 environment.

Interaction This value is ignored if SSLREQUIRED is specified.

SSLPORT=SSL-port

specifies the HTTP port value to use to connect to S3 with SSL.

If you do not specify a port value, the default value is used.

SSLREQUIRED

indicates that all communications with the S3 environment must be made using SSL.

RMDIR Statement

Specifies a directory to delete from an S3 location.

Syntax

RMDIR "*s3-location*";

Required Argument

s3-location

specifies a directory in S3. Fully qualify the path from the bucket name to the directory name. The directory that you specify must be empty.

Examples: S3 Procedure

Example 1: Create a Bucket and Add a File

Features: PROC S3 statement, CONFIG= option
CREATE statement
PUT statement
LIST statement

Details

This example uses the CREATE statement in PROC S3 to create a bucket in S3. Next, the example shows how to add a file to the bucket and then list the contents of the bucket.

Program

```
proc s3 config="/u/marti/.tks3.conf";  
    create "/mybucket";  
    put "/u/marti/project/licj.csv" "/mybucket/licj.csv";  
    list "/mybucket";  
run;
```

Program Description

Execute the PROC S3 statement and specify the location of the PROC S3 configuration file. Connect to S3 with connection options that are contained in the .tks3.conf file in the specified directory.

```
proc s3 config="/u/marti/.tks3.conf";
```

Create a bucket in S3. Use the CREATE statement to create the bucket *mybucket*.

```
create "/mybucket";
```

Store a copy of a local file in the new S3 bucket. Execute the PUT statement to copy the local file licj.csv into a file of the same name in the S3 bucket *mybucket*.

```
put "/u/marti/project/licj.csv" "/mybucket/licj.csv";
```

List the contents of the S3 bucket. Use the LIST statement to see a list of the contents of *mybucket*.

```
list "/mybucket";  
run;
```

If the file licj.csv is the only file in the bucket, then the following message is printed to the SAS log.

Output 52.1 List of Bucket Contents

```
licj.csv      2791403  2015-06-05T19:27:35.000Z
```

Example 2: Manage the Contents of a Bucket

Features:

- PROC S3 statement
- MKDIR statement
- COPY statement
- DELETE statement

Details

This example shows how to create a directory in a bucket and how to copy and delete files.

Program

```
proc s3;
```

```

mkdir "/mybucket/csv";
copy "/mybucket/licj.csv" "/mybucket/csv/licj.csv";
delete "/mybucket/licj.csv";
run;

```

Program Description

Connect to S3 using the default configuration file. Use the PROC S3 statement to connect to S3 with connection options that are specified in the .tks3.conf file in your home directory. This is the default configuration file location.

```
proc s3;
```

Create a new directory. Use the MKDIR statement to create a directory, csv, in the *mybucket*. A note is printed to the SAS log.

```
mkdir "/mybucket/csv";
```

Copy a file into the new directory. Use the COPY statement to copy a file, licj.csv, from *mybucket* into the csv directory.

```
copy "/mybucket/licj.csv" "/mybucket/csv/licj.csv";
```

Delete the original copy of the file. Use the DELETE statement to delete the original copy of the licj.csv file from *mybucket*. Only the copy of the file in /mybucket/csv remains.

```
delete "/mybucket/licj.csv";
run;
```

Output 52.2 Managing Data in a Bucket

```
NOTE: Created directory /mybucket/csv.
```

```
NOTE: Deleted object /mybucket/licj.csv.
```

Example 3: Manage Encryptions with S3 Data

Features:	PROC S3 statement
	COPY statement
	ENCKEY statement
	GET statement
	GETDIR statement
	PUT statement
	PUTDIR statement

Details

This example shows how to work with encryption keys. As a best practice, manage encryption keys in a separate SAS program that you include into another SAS program that interacts with the AWS environment. In this example, the program that manages encryption keys is called `keys.sas`. All encryptions use the default AES256 encryption algorithm that is supported by the S3 environment.

PROGRAM: Keys.sas

```
/* keys.sas */
proc s3;

  /* IAM:KMS keys */
  enckey add name="foo" id="98v27390-1si1-8sc9-38k0-k893j354nw5g";
  enckey add name="bar" id="22f87852-0ak1-5xy2-01j1-1852g809gx4q";

  /* SSE-C keys (user-specified) as character string */
  enckey add name="foo2" key="ke-sj-eb-ok-qk-zi-nb-lu-fh-wi-zi";
  enckey add name="bar2" key="wl-tk-64-rk-i0-zn-wk-7c-s8-a8-jk";

  /* SSE-C (user-specified) keys as hex data */
  enckey replace name="foo2"

hexkey="e1d2e3bb4a5f6079889706b3c4d1e2f999f2c3bb4c5f6079888706f6a4b3e2d
a";
  enckey replace name="bar2"

hexkey="989294020345689209375802937572839401092837467483920183758392019
2";

run;
```

Program Description

Add an IAM:KMS encryption key. Use the ENCKEY statement with the ADD option to add two encryption keys named `foo` and `bar`. You supply the `ID=` value that is defined by the IAM service in the AWS environment. Use the `NAME=` option to provide a user-friendly identifier for the encryption key.

```
/* keys.sas */
proc s3;

  /* IAM:KMS keys */
  enckey add name="foo" id="98v27390-1si1-8sc9-38k0-k893j354nw5g";
  enckey add name="bar" id="22f87852-0ak1-5xy2-01j1-1852g809gx4q";
```

Add an SSE-C key with a character string. You can use the ENCKEY statement and the ADD option to add a user-specified encryption key. Specify a 32-byte character string as the KEY= value and provide a user-friendly value for NAME=.

```
/* SSE-C keys (user-specified) as character string */
enckey add name="foo2" key="ke-sj-eb-ok-qk-zi-nb-lu-fh-wi-zi";
enckey add name="bar2" key="wl-tk-64-rk-i0-zn-wk-7c-s8-a8-jk";
```

Replace existing encryption keys with a user-defined hexadecimal key. Use the REPLACE option with the NAME= option to identify an encryption key that you want to replace. You can specify an encryption key with a 64-digit hexadecimal value with the HEXKEY= option.

```
/* SSE-C (user-specified) keys as hex data */
enckey replace name="foo2"

hexkey="e1d2e3bb4a5f6079889706b3c4d1e2f999f2c3bb4c5f6079888706f6a4b3e2d
a";
enckey replace name="bar2"

hexkey="989294020345689209375802937572839401092837467483920183758392019
2";

run;
```

```
1      /* keys.sas */
2      proc s3;
3      /* IAM:KMS keys */
4      enckey add name="foo" id="98v27390-1si1-8sc9-38k0-k893j354nw5g";
5      enckey add name="bar" id="22f87852-0ak1-5xy2-01j1-l852g809gx4q";
6      /* user-defined keys as character string */
7      enckey add name="foo2" key="ke-sj-eb-ok-qk-zi-nb-lu-fh-wi-zi";
8      enckey add name="bar2" key="wl-tk-64-rk-i0-zn-wk-7c-s8-a8-jk";
9      /* user-defined keys as hex data */
10     enckey replace name="foo2"
11
12     hexkey="e1d2e3bb4a5f6079889706b3c4d1e2f999f2c3bb4c5f6079888706f6a4b3e2da";
13     enckey replace name="bar2"
14
15     hexkey="9892940203456892093758029375728394010928374674839201837583920192";
16
17     run;
```

NOTE: Created the S3 encryption key "foo".

NOTE: Created the S3 encryption key "bar".

NOTE: Created the S3 encryption key "foo2".

NOTE: Created the S3 encryption key "bar2".

NOTE: The value for S3 encryption key "foo2" was redefined.

NOTE: The value for S3 encryption key "bar2" was redefined.

NOTE: PROCEDURE S3 used (Total process time):

real time	3.02 seconds
cpu time	0.01 seconds

NOTE: SAS Institute Inc., SAS Campus Drive, Cary, NC USA 27513-2414

NOTE: The SAS System used:

real time	3.34 seconds
cpu time	0.11 seconds

Program: S3data.sas

```

/* s3data.sas */
%include "keys.sas";

proc s3;
  enckey list;

  put  enckey="foo"  "/home/demouser/s3/foo.txt"  "/mybucket/foo.txt";
  copy srckey="foo"  "/mybucket/foo.txt"  enckey="bar"  "/mybucket/
bar.txt";
  get  enckey="foo"  "/mybucket/foo.txt"  "/home/demouser/foo-get.txt";

  putdir enckey="foo2"  "/home/demouser/s3"  "/mybucket/enctest";
  getdir enckey="foo2"  "/mybucket/enctest"  "/home/demouser/enctest";

run;

```

Program Description

In a separate program, `s3data.sas`, you can specify the encryptions to use while you work with data from an S3 environment.

Include encryption key definitions so that they are accessible to the `s3data.sas` program. Use the `%INCLUDE` statement to retrieve encryption key definitions and their associated names.

```

/* s3data.sas */
%include "keys.sas";

```

List available encryption keys. Use the `LIST` option with the `ENCKEY` statement to see the currently defined encryption keys.

```

proc s3;
  enckey list;

```

Manipulate the encryptions for files. The `PUT` statement uses the `foo` encryption key from `keys.sas` to encrypt the local `foo.txt` file that is copied to an S3 bucket called `mybucket`. The `COPY` statement copies the `foo.txt` file in `mybucket` to a file called `bar.txt` that is encrypted using the `bar` encryption key. The `GET` statement copies the file `foo.txt` from `mybucket` to your local directory and renames the file `foo-get.txt`. The file is encrypted using the `foo` encryption key.

```

put  enckey="foo"  "/home/demouser/s3/foo.txt"  "/mybucket/foo.txt";
copy srckey="foo"  "/mybucket/foo.txt"  enckey="bar"  "/mybucket/
bar.txt";
get  enckey="foo"  "/mybucket/foo.txt"  "/home/demouser/foo-get.txt";

```

Create a subdirectory in an S3 bucket and encrypt it. Use the `PUTDIR` statement to create a subdirectory, `Enctest`, in `mybucket`. This location is a copy of your local

directory /home/demouser/s3. The Enctest location is encrypted using the foo2 encryption key. In keys.sas, the foo2 encryption key is represented by a hexadecimal value, because that encryption key replaced the original encryption key for foo2.

```
putdir enckey="foo2" "/home/demouser/s3" "/mybucket/enctest";
```

Create a decrypted local directory that is a copy of an encrypted S3 location. Use the GETDIR statement to create a decrypted copy of an S3 location, Enctest. The encryption key name is provided to enable decryption of the data. Data is encrypted by default during data transfer. The new local directory is /home/demouser/enctest.

```
getdir enckey="foo2" "/mybucket/enctest" "/home/demouser/enctest";
```

```
run;
```

```
1      /* s3data.sas */
2      %include "keys.sas";

NOTE: Created the S3 encryption key "foo".
NOTE: Created the S3 encryption key "bar".
NOTE: Created the S3 encryption key "foo2".
NOTE: Created the S3 encryption key "bar2".
NOTE: The value for S3 encryption key "foo2" was redefined.
NOTE: The value for S3 encryption key "bar2" was redefined.
NOTE: PROCEDURE S3 used (Total process time):
      real time          3.02 seconds
      cpu time           0.01 seconds

17      proc s3;
18      enckey list;
19      put enckey="foo" "/home/demouser/foo.txt" "/mybucket/foo.txt";
20      copy srckey="foo" "/mybucket/foo.txt" enckey="bar" "/mybucket/bar.txt";
21      get enckey="foo" "/mybucket/foo.txt" "/home/demouser/foo-get.txt";
22      putdir enckey="foo2" "/home/demouser/s3" "/mybucket/enctest";
23      getdir enckey="foo2" "/mybucket/enctest" "/home/demouser";
24      run;

Defined S3 Encryption Keys
Name      Type      Details
_DEFAULT_ KMS
bar       KMS        22f87852-0ak1-5xy2-01j1-l852g809gx4q
bar2      SSE-C      HEX AES256
foo       KMS        98v27390-1sil-8sc9-38k0-k893j354nw5g
foo2      SSE-C      HEX AES256
NOTE: Writing directory /home/demouser/s3 to S3 path /mybucket/enctest
NOTE: Put: /home/demouser/s3/s3data.sas -> /mybucket/enctest/s3data.sas
NOTE: Put: /home/demouser/s3/foo.txt -> /mybucket/enctest/foo.txt
NOTE: Put: /home/demouser/s3/keys.log -> /mybucket/enctest/keys.doc.log
NOTE: Put: /home/demouser/s3/keys.sas -> /mybucket/enctest/keys.sas
NOTE: Writing contents of S3 path /mybucket/enctest to local path /home/demouser
NOTE: PROCEDURE S3 used (Total process time):
      real time          5.53 seconds
      cpu time           2.35 seconds

NOTE: SAS Institute Inc., SAS Campus Drive, Cary, NC USA 27513-2414
NOTE: The SAS System used:
      real time          8.82 seconds
      cpu time           2.47 seconds
```

Example 4: Define a Custom Region with PROC S3

Features: REGION ADD statement

Details

This example shows how to define a custom region using PROC S3.

Program

```
proc s3;
    region add host="s3-us-west-1.amazonaws.com"
             name="south-west"
             sslrequired replace;
run;
```

Program Description

Connect to S3 using the default configuration file. Use the PROC S3 statement to connect to S3 with connection options that are specified in the .tk3.conf file in your home directory. This is the default configuration file location.

```
proc s3;
```

Define a new custom region. Use the REGION ADD statement to create a custom region, south-west. The SSLREQUIRED argument indicates that SSL is required when communicating with the S3 environment. The REPLACE argument specifies that if the custom region is already defined, then the definition should be overwritten.

```
    region add host="s3-us-west-1.amazonaws.com"
             name="south-west"
             sslrequired replace;
run;
```

Program

```
proc s3;
    region add host="s3.us-west-004.backblazeb2.com"
             name="backblaze-west"
```

```
        region="us-west-004"  
        sslrequired replace;  
run;
```

Program Description

Connect to S3 using the default configuration file. The PROC S3 statement connects to S3 with connection options that are specified in the .tk3.conf file in your home directory.

```
proc s3;
```

Define a new custom region. Use the REGION ADD statement to create a custom region, us-west-004. As in the previous program, the SSLREQUIRED argument indicates that SSL is required when communicating with the S3 environment. The REPLACE argument specifies that if the custom region is already defined, then the definition should be overwritten. Use the optional REGION= argument to provide your region code, typically if using the same value for NAME= would cause a conflict.

```
    region add host="s3.us-west-004.backblazeb2.com"  
            name="backblaze-west"  
            region="us-west-004"  
            sslrequired replace;  
run;
```

SCAPROC Procedure

Overview: SCAPROC Procedure	1957
What Does the SCAPROC Procedure Do?	1957
Special Considerations	1958
Concepts: SCAPROC Procedure	1958
Handling Global Statements	1958
Internal Statements That Appear in Output	1959
Syntax: SCAPROC Procedure	1959
PROC SCAPROC Statement	1960
RECORD Statement	1960
WRITE Statement	1961
Results: SCAPROC Procedure	1962
Results	1962
Examples: SCAPROC Procedure	1966
Example 1: Specifying a Record File	1966
Example 2: Specifying the Grid Job Generator	1968

Overview: SCAPROC Procedure

What Does the SCAPROC Procedure Do?

The SCAPROC procedure implements the SAS Code Analyzer, which captures information about input, output, and the use of macro symbols from a SAS job while it is running. The SAS Code Analyzer can write this information and the information that is in the original SAS file to a file that you specify. The SCAPROC procedure can also generate a grid-enabled job that can concurrently run

independent pieces of the job. You can issue the SCAPROC procedure on your operating system's command line or in SAS code in the SAS Editor window.

The following command runs your SAS job with the SAS Code Analyzer from your operating system's command line:

```
sas yourjob.sas -initstmt "proc scaproc; record 'yourjob.txt' ; run;"
```

sas

is the command used at your site to start SAS.

yourjob.sas

is the name of the SAS job that you want to analyze.

yourjob.txt

is the name of the file that will contain a copy of your SAS code. The file will also contain the comments that are inserted to show input and output information, macro symbol usage, and other aspects of your job. For information about issuing PROC SCAPROC in SAS code, see the examples.

Special Considerations

Some tasks of grid-enabled jobs can have dependencies on previous tasks. PROC SCAPROC combines and reorders these tasks based on their dependencies to the preceding tasks. Combining the tasks and submitting them in the same work unit enables faster processing of the tasks. The NOOPTIMIZE argument of the GRID option disables the combining and reordering of tasks of grid-enabled jobs.

For the GRID statement to work, your site has to license SAS Grid Manager or SAS/CONNECT. SAS Grid Manager enables your generated grid job to run on a grid of distributed machines. SAS/CONNECT enables your generated grid job to run on parallel SAS sessions on one symmetric multiprocessing (SMP) machine.

Concepts: SCAPROC Procedure

Handling Global Statements

You can mark a set of statements that you want to submit to each remote session in the generated grid job. Include an `/* SCAPROC GLOBAL BEGIN */` comment on the line before the set of statements, and include an `/* SCAPROC GLOBAL END */` comment on the line after the statements. PROC SCAPROC submits the statements in the marked session to each of the remote sessions before any of the job's steps are submitted.

In the following example, PROC SCAPROC submits the statements between the `/* SCAPROC GLOBAL BEGIN */` and `/* SCAPROC GLOBAL END */` comments with each of the PROC SUMMARY statements.

```
/* SCAPROC GLOBAL BEGIN */;
    libname one "SAS-library-1";
    libname two "SAS-library-1";
    %let year=2018;
/* SCAPROC GLOBAL END */;

Proc summary data=one.sales;
    where year=&year
    class month;
    var sales;
    output out=sasuser.sales;
run;
Proc summary data=two.expenses;
    where year=&year
    class month;
    var expenses;
    output out=sasuser.expenses;
run;
```

You can include any SAS statement that is valid with the generated grid job. You can use text that is all uppercase, all lowercase, or mixed case.

Internal Statements That Appear in Output

The statements from your SAS program are included in the PROC SCAPROC output file. There are additional statements that are called internally by the SCAPROC procedure that also are listed in the output file. You cannot call these statements directly. However, here is a brief description of the function of these internal statements:

- GETSESSION. This statement gets an available session or parallel processing thread.
- SHUTDOWN. This statement stops processing.
- STARTUP. This statement starts a SAS session or starts a thread for parallel processing.
- TASKWAIT. This statement instructs SAS to wait until processing for a session or parallel processing threads is complete.

Syntax: SCAPROC Procedure

PROC SCAPROC;

```
RECORD filespec <ATTR> <OPENTIMES> <INTCON> <EXPANDMACROS>
  <GRID filespec <RESOURCE "resource name"> <INHERITLIB> <NOOPTIMIZE>
  >;
WRITE;
```

Statement	Task	Example
PROC SCAPROC	Implement the SAS Code Analyzer	Ex. 1 , Ex. 2
RECORD	Specify a filename or a fileref to contain the output of the SAS Code Analyzer	Ex. 1 , Ex. 2
WRITE	Output information to the record file	Ex. 1

PROC SCAPROC Statement

Specifies that SAS will run the SAS Code Analyzer with your SAS job.

- Examples:
- [“Example 1: Specifying a Record File” on page 1966](#)
 - [“Example 2: Specifying the Grid Job Generator” on page 1968](#)

Syntax

```
PROC SCAPROC;
```

RECORD Statement

Specifies a filename or a fileref to contain the output of the SAS Code Analyzer.

- Examples:
- [“Example 1: Specifying a Record File” on page 1966](#)
 - [“Example 2: Specifying the Grid Job Generator” on page 1968](#)

Syntax

```
RECORD filespec <ATTR> <OPENTIMES> <INTCON> <EXPANDMACROS>
  <GRID filespec ><RESOURCE "resource name"><INHERITLIB><NOOPTIMIZE> ;
```

Required Argument

filespec

specifies a physical filename in quotation marks, or a fileref, that indicates a file to contain the output of the SAS Code Analyzer. The output is the original SAS source and comments that contain information about the job. For more information about the output comments, see [“Results” on page 1962](#).

Optional Arguments

ATTR

writes additional information about the variables in the input data sets and views.

OPENTIMES

writes the open time, size, and physical filename of the input data sets.

INTCON

outputs information about table integrity constraints.

EXPANDMACROS

expands macro invocations into separate tasks.

GRID

filespec

specifies a physical filename in quotation marks, or a fileref, that points to a file that will contain the output of the Grid Job Generator.

RESOURCE “resource name”

specifies the resource to use in the `grdsvs_enable` function call. The default is SASApp.

INHERITLIB

adds INHERITLIB=(USER) to the SIGNON statements in the grid-enabled job. By default, the user library (if one is specified) is assigned in each remote session.

.....
Note: INHERITLIB should be used only when the USER library is not globally accessible to all remote sessions.

NOOPTIMIZE

disables the combining and reordering of tasks for grid-enabled jobs.

WRITE Statement

Specifies output information to the record file.

Example: [“Example 1: Specifying a Record File” on page 1966](#)

Syntax

WRITE;

Without Arguments

The WRITE statement specifies that the SAS Code Analyzer writes information to the record file, if a file has been specified with the RECORD statement. The Grid Job Generator will also run at this time if it has been specified. Termination of SAS also causes the SAS Code Analyzer to write information to the specified record file.

Results: SCAPROC Procedure

Results

The following list contains explanations of the comments that the SAS Code Analyzer writes to the record file that you specify with PROC SCAPROC. The output comments are bounded by /* and */ comment tags in the record file. That format is represented here to enhance clarity when the user reads a record file.

```
/* JOBSPLIT: DATASET INPUT|OUTPUT|UPDATE SEQ|MULTI name */  
specifies that a data set was opened for reading, writing, or updating.
```

INPUT

specifies that SAS read the data set.

OUTPUT

specifies that SAS wrote the data set.

UPDATE

specifies that SAS updated the data set.

SEQ

specifies that SAS opened the data set for sequential access.

MULTI

specifies that SAS opened the data set for multipass access.

name

specifies the name of the data set.

```
/* JOBSPLIT: CATALOG INPUT|OUTPUT|UPDATE name */  
specifies that a catalog was opened for reading, writing, or updating.
```

INPUT

specifies that SAS read the catalog.

OUTPUT

specifies that SAS wrote the catalog.

UPDATE

specifies that SAS updated the catalog.

name

specifies the name of the catalog.

```
/* JOBSPLIT: FILE INPUT|OUTPUT|UPDATE name */
```

specifies that an external file was opened for reading, writing, or updating.

INPUT

specifies that SAS read the file.

OUTPUT

specifies that SAS wrote the file.

UPDATE

specifies that SAS updated the file.

name

specifies the name of the file.

```
/* JOBSPLIT: ITEMSTOR INPUT|OUTPUT|UPDATE name */
```

specifies that an ITEMSTOR was opened for reading, writing, or updating.

INPUT

specifies that SAS read the ITEMSTOR.

OUTPUT

specifies that SAS wrote the ITEMSTOR.

UPDATE

specifies that SAS updated the ITEMSTOR.

name

specifies the name of the ITEMSTOR.

```
/* JOBSPLIT: OPENTIME name DATE:date PHYS:phys SIZE:size */
```

specifies that a data set was opened for input. SAS writes the OPENTIME and the SIZE of the file.

name

specifies the name of the data set.

DATE

specifies the date and time that the data set was opened. The value that is returned for DATE is not the creation time of the file.

PHYS

specifies the complete physical name of the data set that was opened.

SIZE

specifies the size of the data set in bytes.

```
/* JOBSPLIT: ATTR name INPUT|OUTPUT VARIABLE:variable name
TYPE:CHARACTER|NUMERIC LENGTH:length LABEL:label FORMAT:format
INFORMAT:informat */
```

specifies that when a data set is closed, SAS reopens it and writes the attributes of each variable. One ATTR line is produced for each variable.

name
specifies the name of the data set.

INPUT
specifies that SAS read the data set.

OUTPUT
specifies that SAS wrote the data set.

VARIABLE
specifies the name of the current variable.

TYPE
specifies whether the variable is character or numeric.

LENGTH
specifies the length of the variable in bytes.

LABEL
specifies the variable label if it has one.

FORMAT
specifies the variable format if it has one.

INFORMAT
specifies the variable informat if it has one.

/ JOBSPLIT: SYMBOL SET|GET name */*
specifies that a macro symbol was accessed.

SET
specifies that SAS set the symbol. For example, SAS set the symbol **sym1** in the following code: %let sym1=sym2

GET
specifies that SAS retrieved the symbol. For example, SAS retrieved the symbol **sym** in the following code: a="&sym"

name
specifies the name of the symbol.

/ JOBSPLIT: ELAPSED number */*
specifies a number for you to use to determine the relative run times of tasks.

number
specifies a number for you to use to determine the relative run times of tasks.

/ JOBSPLIT: USER useroption */*
specifies that SAS uses the USER option with the grid job code to enable single-level data set names to reside in the Work library.

useroption
specifies the value that is to be used while the code is running.

/ JOBSPLIT: _DATA_ */*
specifies that SAS is to use the reserved data set name **_DATA_**.

/ JOBSPLIT: _LAST_ */*
specifies that SAS is to use the reserved data set name **_LAST_**.

```

/* JOBSPLIT: PROCNAME procname|DATASTEP */
    specifies the name of the SAS procedure or DATA step for this step.

/* JOBSPLIT: LIBNAME <libname options> */
    specifies the LIBNAME options that were provided in a LIBNAME statement or
    were set internally.

/* JOBSPLIT: SYSSCP <sysscp> */
    specifies the value of the SYSSCP automatic macro variable when the SAS job
    was run.

/* JOBSPLIT: JOBSTARTTIME <datetime> */
    records the date and time that a job started.

/* JOBSPLIT: JOBENDTIME <datetime> */
    records the date and time that a job ended.

/* JOBSPLIT: TASKSTARTTIME <datetime> */
    records the date and time that a task started.

/* JOBSPLIT: INTCON <table name> <INPUT | OUTPUT> NAME:<name>
TYPE:<UNIQUE|CHECK|NOTNULL|PRIMARY|FOREIGN|REFERENTIAL>
VARIABLES:<names> WHERE:<where clause> REFERENCE: ONDELETE: ONUPDATE:
MESSAGE:<msg> MESSAGETYPE:<USER> */
    table name
        specifies the name of the table.

    INPUT
        specifies that the table was opened for input.

    OUTPUT
        specifies that the table was opened for output.

    NAME
        specifies the name of the integrity constraint.

    TYPE
        specifies the type of integrity constraint:

```

Table 53.1 Integrity Constraint for Each Type

Type	Integrity Constraint
UNIQUE	unique
CHECK	check
NOTNULL	not null
PRIMARY	primary key
FOREIGN	foreign key
REFERENTIAL	a FOREIGN integrity constraint in another table that references this table

VARIABLES

specifies the variables involved with this integrity constraint. Otherwise, it is empty.

WHERE

specifies the WHERE expression for CHECK integrity constraints. Otherwise, it is empty.

REFERENCE

specifies one of the following integrity constraints:

- For FOREIGN integrity constraints, it specifies the table the foreign key references.
- For REFERENTIAL integrity constraints, it specifies the table containing the foreign key.
- Otherwise, it is empty.

ONDELETE

specifies the ON DELETE referential action for FOREIGN and REFERENTIAL integrity constraints. Otherwise, it is empty.

ONUPDATE

specifies the ON UPDATE referential action for FOREIGN and REFERENTIAL integrity constraints. Otherwise, it is empty.

MESSAGE

specifies the text of the error message, if there is any.

MESSAGETYPE

specifies USER if MESSAGETYPE=USER was specified. Otherwise, it is empty.

Examples: SCAPROC Procedure

Example 1: Specifying a Record File

Features:

RECORD statement
WRITE statement

This example specifies the record file '**record.txt**' and writes information from the SAS Code Analyzer to the file.

Program

```
proc scaproc;
    record 'record.txt';
run;

data a;
    do i = 1 to 100000;
        j = cos(i);
        output;
    end;
run;

proc print data=a(obs=25);
run;

proc means data=a;
run;

proc scaproc;
    write;
run;
```

Output

Output 53.1 Contents of the record.txt File

```

/* JOBSPLIT: DATASET OUTPUT SEQ WORK.A.DATA */
/* JOBSPLIT: LIBNAME WORK ENGINE V9 PHYS C:\DOCUME~1\userid\LOCALS~1\Temp\SAS
Temporary Files\_TD1252 */
/* JOBSPLIT: ELAPSED 3984 */
/* JOBSPLIT: PROCNAME DATASTEP */
/* JOBSPLIT: STEP SOURCE FOLLOWS */

data a;
  do i = 1 to 1000000;
    j = cos(i);
    output;
  end;
run;

/* JOBSPLIT: ITEMSTOR INPUT SASUSER.TEMPLAT */
/* JOBSPLIT: ITEMSTOR INPUT SASHELP.TMPLMST */
/* JOBSPLIT: DATASET INPUT SEQ WORK.A.DATA */
/* JOBSPLIT: LIBNAME WORK ENGINE V9 PHYS C:\DOCUME~1\userid\LOCALS~1\Temp\SAS
Temporary Files\_TD1252 */
/* JOBSPLIT: ELAPSED 5187 */
/* JOBSPLIT: PROCNAME PRINT */
/* JOBSPLIT: STEP SOURCE FOLLOWS */
proc print data=a(obs=25);
run;

/* JOBSPLIT: DATASET INPUT SEQ WORK.A.DATA */
/* JOBSPLIT: LIBNAME WORK ENGINE V9 PHYS C:\DOCUME~1\userid\LOCALS~1\Temp\SAS
Temporary Files\_TD1252 */
/* JOBSPLIT: FILE OUTPUT C:\winnt\profiles\userid\record.txt */
/* JOBSPLIT: SYMBOL GET SYSSUMTRACE */
/* JOBSPLIT: ELAPSED 2750 */
/* JOBSPLIT: PROCNAME MEANS */
/* JOBSPLIT: STEP SOURCE FOLLOWS */
proc means data=a;
run;

/* JOBSPLIT: END */

```

Example 2: Specifying the Grid Job Generator

Features: RECORD statement
 GRID statement

Details

This example writes information from the SAS Code Analyzer to the file that is named `1.txt`. The example code also runs the Grid Job Generator, and writes that information to the file that is named `1.grid`. Notice that this example does not have an ending statement that contains this code:

```
proc scaproc;  
    write;  
run;
```

When SAS terminates, PROC SCAPROC automatically runs any pending RECORD or GRID statements.

For the GRID statement to work, your site has to license SAS Grid Manager or SAS/CONNECT. SAS Grid Manager enables your generated grid job to run on a grid of distributed machines. SAS/CONNECT enables your generated grid job to run on parallel SAS sessions on one symmetric multiprocessing (SMP) machine.

Program

```
proc scaproc;  
    record '1.txt' grid '1.grid';  
run;  
  
data a;  
    do i = 1 to 100000;  
        j = cos(i);  
        output;  
    end;  
run;  
  
proc print data=a(obs=25);  
run;  
  
proc means data=a;  
run;
```

Output

Output 53.2 Contents of the 1.txt File

```
/* JOBSPLIT: DATASET OUTPUT SEQ WORK.A.DATA */
/* JOBSPLIT: LIBNAME WORK ENGINE V9 PHYS C:\DOCUME~1\userid\LOCALS~1\Temp\SAS
Temporary Files\_TD1252 */
/* JOBSPLIT: ELAPSED 375 */
/* JOBSPLIT: PROCNAME DATASTEP */
/* JOBSPLIT: STEP SOURCE FOLLOWS */

data a;
  do i = 1 to 1000000;
    j = cos(i);
    output;
  end;
run;

/* JOBSPLIT: DATASET INPUT SEQ WORK.A.DATA */
/* JOBSPLIT: LIBNAME WORK ENGINE V9 PHYS C:\DOCUME~1\userid\LOCALS~1\Temp\SAS
Temporary Files\_TD1252 */
/* JOBSPLIT: ELAPSED 46 */
/* JOBSPLIT: PROCNAME PRINT */
/* JOBSPLIT: STEP SOURCE FOLLOWS */
proc print data=a(obs=25);
run;

/* JOBSPLIT: DATASET INPUT SEQ WORK.A.DATA */
/* JOBSPLIT: LIBNAME WORK ENGINE V9 PHYS C:\DOCUME~1\userid\LOCALS~1\Temp\SAS
Temporary Files\_TD1252 */
/* JOBSPLIT: FILE OUTPUT C:\WINNT\Profiles\userid\1.txt */
/* JOBSPLIT: SYMBOL GET SYSSUMTRACE */
/* JOBSPLIT: ELAPSED 81453 */
/* JOBSPLIT: PROCNAME MEANS */
/* JOBSPLIT: STEP SOURCE FOLLOWS */
proc means data=a;
run;

/* JOBSPLIT: END */
```


SCOREACCEL Procedure

Overview: SCOREACCEL Procedure	1971
What Does the SCOREACCEL Procedure Do?	1972
Concepts: SCOREACCEL Procedure	1973
Supported Platforms for In-Database Model Scoring	1973
Arguments Listed by Data Source and Statement	1974
Publishing Models to a Model Table on the CAS Server	1977
Naming Requirements	1977
Syntax: SCOREACCEL Procedure	1977
PROC SCOREACCEL Statement	1978
DELETEMODEL Statement	1979
PUBLISHMODEL Statement	1987
RUNMODEL Statement	2000
Examples: SCOREACCEL Procedure	2013
Example 1: Publish, Run, and Delete a DATA Step Model in CAS	2013
Example 2: Publish, Run, and Delete a Model in Teradata	2015
Example 3: Publish a Model to a Hive Table and Run in Cloudera	2018
Example 4: Publish a Model to a Spark Table and Run in Databricks on AWS ...	2021
Example 5: Publish a Model to a Spark Table and Run in Azure Databricks	2023
Example 6: Publish a Model to Amazon S3 and Run in Databricks on AWS	2026
Example 7: Publish a Model to ADLS and Run in Azure Databricks	2028
Example 8: Publish a Model to an SQL Server Table and Run in Azure Synapse	2030
Example 9: Publish a Model to ADLS and Run in Azure Synapse	2032
Example 10: Publish, Run, and Delete a Model in SingleStore	2036

Overview: SCOREACCEL Procedure

What Does the SCOREACCEL Procedure Do?

PROC SCOREACCEL is an interactive procedure that provides an interface to the CAS server and external data sources for model publishing and scoring.

- Models are supported in the form of DATA step code, DS2 code, or analytic stores.
- Model code can be published to a session-local or global CAS table and executed in CAS. Session-local means it is accessible only to the current CAS session that is executing PROC SCOREACCEL.
- Alternatively, models can be published to external data sources. See [“Supported Platforms for In-Database Model Scoring” on page 1973](#). After a model is published to an external data source, running the model invokes the SAS Embedded Process.
- Models can be removed from CAS or from an external data source by using the DELETEMODEL statement.
- PROC SCOREACCEL calls CAS actions to complete the tasks. You can use the Model Publishing and Scoring action set directly, instead of using PROC SCOREACCEL.
- For a list of the supported scoring models and other information, see [“In-Database Model Scoring” in SAS In-Database Products: User’s Guide](#).
- For information about PROC ACCELERATOR, an alternative to PROC SCOREACCEL, see [“Feature Comparison for PROC ACCELERATOR and PROC SCOREACCEL” on page 96](#). PROC ACCELERATOR is supported for Databricks and SingleStore.

Note: Starting with 2026.04 for Azure Synapse, PROC SCOREACCEL is deprecated. The Model Publishing and Scoring action set and the SAS Embedded Process for Spark action set are also deprecated. The change is a result of a Microsoft announcement to end support for Azure Synapse Runtime for Apache Spark 3.4. You can no longer deploy the sepcorespark RPM in Synapse. You are encouraged to use PROC ACCELERATOR instead. To use PROC ACCELERATOR, you must deploy the SAS Embedded Process from the accelserver RPM. See [“Deployment Details for Databricks and Synapse” on page 98](#).

Note: Starting with 2026.06 for Databricks, PROC SCOREACCEL is deprecated. The Model Publishing and Scoring action set and the SAS Embedded Process for

Spark action set are also deprecated. You are encouraged to use PROC ACCELERATOR instead. To use PROC ACCELERATOR, you must deploy the SAS Embedded Process from the accelserver RPM. See “[Deployment Details for Databricks and Synapse](#)” on page 98.

Concepts: SCOREACCEL Procedure

Supported Platforms for In-Database Model Scoring

The following table shows processing environments other than CAS where you can run model scoring.

Table 54.1 Platforms for Running a Model

RUNMODEL Processing	RUNMODEL EXTTYPE= Value CASLIB Values	PUBLISHMODEL Model Storage	PUBLISHMODEL and DELETEMODEL EXTTYPE= Value CASLIB Values	Documentation
AWS Databricks	EXTTYPE=SPARK In the CASLIB, specify srcType=spark and platform=databricks.	Spark table ¹	EXTTYPE=SPARK In the CASLIB, specify srcType=spark and platform=databricks.	Overview Example
		Amazon S3 (deprecated) ²	EXTTYPE=FILESYSTEM In the CASLIB, specify srcType=spark and platform=databricks.	Overview Example
Azure Databricks	EXTTYPE=SPARK In the CASLIB, specify srcType=spark and platform=databricks.	Spark table ¹	EXTTYPE=SPARK In the CASLIB, specify srcType=spark and platform=databricks.	Overview Example
		ADLS Gen2 (deprecated) ^{2, 3}	EXTTYPE=FILESYSTEM In the CASLIB, specify srcType=adls.	Overview Example

RUNMODEL Processing	RUNMODEL EXTTYPE= Value CASLIB Values	PUBLISHMODEL Model Storage	PUBLISHMODEL and DELETEMODEL EXTTYPE= Value CASLIB Values	Documentation
Azure Synapse ⁴	EXTTYPE=SPARK In the CASLIB, specify srcType=spark and platform=synapse.	SQL Server table	EXTTYPE=SYNAPSE In the CASLIB, specify srcType=jdbc and other required arguments.	Overview Example
		ADLS Gen2 (deprecated) ³	EXTTYPE=FILESYSTEM In the CASLIB, specify srcType=adls.	Overview Example
Hadoop	EXTTYPE=HADOOP In the CASLIB, specify srcType=hadoop.	Hive table ⁵	EXTTYPE=HADOOP In the CASLIB, specify srcType=hadoop.	Overview Example
		Hadoop (deprecated)	EXTTYPE=HADOOP In the CASLIB, specify srcType=hadoop.	Overview
SingleStore (SAS SpeedyStore) ⁶	EXTTYPE=SINGLESTORE In the CASLIB, specify srcType=singlestore.	SingleStore	EXTTYPE=SINGLESTORE In the CASLIB, specify srcType=singlestore.	Overview Example
Teradata	EXTTYPE=TERADATA In the CASLIB, specify srcType=teradata.	Teradata	EXTTYPE=TERADATA In the CASLIB, specify srcType=teradata.	Overview Example

1 As of 2023.10 for Databricks, you can publish a model to a Spark table, and this method is recommended.

2 Publishing a model to Databricks by using the DBFS REST API is not recommended.

3 ADLS Gen1 and Azure blob storage are not supported.

4 As of 2026.04 for Synapse, PROC SCOREACCEL is deprecated. The change is a result of a Microsoft announcement to end support for Azure Synapse Runtime for Apache Spark 3.4. You can no longer deploy the sepcorespark RPM in Synapse. You are encouraged to use PROC ACCELERATOR instead. To use PROC ACCELERATOR, you must deploy the SAS Embedded Process from the accelserver RPM. See “[Deployment Details for Databricks and Synapse](#)” on page 98.

5 As of 2023.09 for Hadoop, you can publish a model to a Hive table, and this method is recommended. Cloudera and Amazon EMR are supported.

6 In-database model scoring is supported if you license and deploy the product SAS SpeedyStore. For more information, see “[SAS SpeedyStore Data Connector](#)” in *SAS Cloud Analytic Services: User's Guide*. In-database model scoring is not supported by the SingleStore standard data connector.

Arguments Listed by Data Source and Statement

EXTTYPE=CAS

- [DELETEMODEL statement on page 1979](#)

CASLIB=, DATABASE=, DELETGLOBAL=, EXTTYPE=, MODELNAME=,
 MODELTABLE=, PERSISTTABLE=, PROMOTETABLE=, REPLACETABLE=,
 SCHEMA=

■ [PUBLISHMODEL statement on page 1987](#)

CASLIB=, DATABASE=, EXTTYPE=, FORMATFILE=,
 FORMATITEMSTOREFILE=, KEEPLIST=, MODELNAME=, MODELNOTES=,
 MODELTABLE=, MODELTYPE=, MODELUUID=, OUTDIR=, PERSISTTABLE=,
 PROGRAMFILE=, PROMOTETABLE=, PUBLISHGLOBAL=, REPLACEMODEL=,
 REPLACETABLE=, SCHEMA=, STORETABLES=, VARXMLFILE=

■ [RUNMODEL statement on page 2000](#)

CASLIB=, DATABASE=, EXTTYPE=, FORMATFILE=,
 FORMATITEMSTOREFILE=, KEEPLIST=, MODELNAME=, MODELNOTES=,
 MODELTABLE=, MODELTYPE=, MODELUUID=, OUTDIR=, PERSISTTABLE=,
 PROGRAMFILE=, PROMOTETABLE=, PUBLISHGLOBAL=, REPLACEMODEL=,
 REPLACETABLE=, SCHEMA=, STORETABLES=, VARXMLFILE=

EXTTYPE=DATABRICKS

■ [DELETEMODEL statement on page 1979](#)

CASLIB=, DATABASE=, EXTTYPE=, MODELDATABASE=, MODELNAME=,
 SCHEMA=

■ [PUBLISHMODEL statement on page 1987](#)

CASLIB=, DATABASE=, EXTTYPE=, FORMATFILE=,
 FORMATITEMSTOREFILE=, KEEPLIST=, MODELDATABASE=, MODELNAME=,
 MODELTYPE=, OUTDIR=, PROGRAMFILE=, REPLACEMODEL=, SCHEMA=,
 STOREFILES=, STORETABLES=, VARXMLFILE=

EXTTYPE=FILESYSTEM (deprecated)

■ [DELETEMODEL statement on page 1979](#)

CASLIB=, DATABASE=, EXTTYPE=, MODELDIR=, MODELNAME=,
 PASSWORD=, SCHEMA=, TIMEOUT=

■ [PUBLISHMODEL statement on page 1987](#)

CASLIB=, DATABASE=, EXTTYPE=, FORMATFILE=,
 FORMATITEMSTOREFILE=, KEEPLIST=, MODELDIR=, MODELNAME=,
 MODELTYPE=, OUTDIR=, PASSWORD=, PROGRAMFILE=, REPLACEMODEL=,
 SCHEMA=, STOREFILES=, STORETABLES=, TIMEOUT=, VARXMLFILE=

EXTTYPE=HADOOP

■ [DELETEMODEL statement on page 1979](#)

CASLIB=, DATABASE=, EXTTYPE=, MODELDATABASE=, MODELDIR=,
 MODELNAME=, SCHEMA=

■ [PUBLISHMODEL statement on page 1987](#)

CASLIB=, DATABASE=, EXTTYPE=, FORMATFILE=,
 FORMATITEMSTOREFILE=, KEEPLIST=, MODELDATABASE=, MODELDIR=,
 MODELNAME=, MODELTYPE=, OUTDIR=, PROGRAMFILE=,
 REPLACEMODEL=, SCHEMA=, STOREFILES=, STORETABLES=, VARXMLFILE=

■ [RUNMODEL statement on page 2000](#)

CASLIB=, DATABASE=, DBMAXTEXT=, EXTTYPE=, FORCEOVERWRITE=,
 INDATASET=, INTABLE=, KEEPLISTCOLUMNS=, KEEPLISTFILE=,
 MODELDATABASE=, MODELDIR=, MODELNAME=, OUTDATASET=,
 OUTSCHEMA=, OUTTABLE=, PROPERTIES=, SCHEMA=, TRACE

EXTTYPE=SINGLESTORE

- [DELETEMODEL statement on page 1979](#)

CASLIB=, DATABASE=, EXTTYPE=, MODELNAME=, SCHEMA=

- [PUBLISHMODEL statement on page 1987](#)

CASLIB=, DATABASE=, EXTTYPE=, FORMATFILE=,
 FORMATITEMSTOREFILE=, KEEPLIST=, MODELNAME=, MODELNOTES=,
 MODELTYPE=, MODELUUID=, OUTDIR=, PROGRAMFILE=, REPLACEMODEL=,
 SCHEMA=, STOREFILES=, STORETABLES=, VARXMLFILE=

- [RUNMODEL statement on page 2000](#)

CASLIB=, DATABASE=, EOPTIONS=, EXTTYPE=, INQUERY=, INTABLE=,
 MODELNAME=, OUTTABLE=, OUTTABLEOPTIONS=, SCHEMA=

EXTTYPE=SPARK

- [RUNMODEL statement on page 2000](#)

CASLIB=, DATABASE=, EXTTYPE=, FORCEOVERWRITE=, INDATASET=,
 INTABLE=, KEEPLISTCOLUMNS=, KEEPLISTFILE=, MODELDATABASE=,
 MODELDIR=, MODELNAME=, OUTDATASET=, OUTSCHEMA=, OUTTABLE=,
 PROPERTIES=, SCHEMA=, TRACE

EXTTYPE=SYNAPSE

- [DELETEMODEL statement on page 1979](#)

CASLIB=, DATABASE=, EXTTYPE=, MODELNAME=, MODELSHEMA=,
 SCHEMA=

- [PUBLISHMODEL statement on page 1987](#)

CASLIB=, DATABASE=, EXTTYPE=, FORMATFILE=,
 FORMATITEMSTOREFILE=, KEEPLIST=, MODELNAME=, MODELSHEMA=,
 MODELTYPE=, OUTDIR=, PROGRAMFILE=, REPLACEMODEL=, SCHEMA=,
 STOREFILES=, STORETABLES=, VARXMLFILE=

EXTTYPE=TERADATA

- [DELETEMODEL statement on page 1979](#)

CASLIB=, DATABASE=, EXTTYPE=, MODELNAME=, MODELTABLE=,
 SCHEMA=

- [PUBLISHMODEL statement on page 1987](#)

CASLIB=, DATABASE=, EXTTYPE=, FORMATFILE=,
 FORMATITEMSTOREFILE=, KEEPLIST=, MODELNAME=, MODELNOTES=,
 MODELTABLE=, MODELTYPE=, MODELUUID=, OUTDIR=, PROGRAMFILE=,
 REPLACEMODEL=, SCHEMA=, STOREFILES=, STORETABLES=, VARXMLFILE=

- [RUNMODEL statement on page 2000](#)

CASLIB=, DATABASE=, EOPTIONS=, EXTTYPE=, INQUERY=, INTABLE=,
 MODELNAME=, MODELTABLE=, OUTTABLE=, OUTTABLEOPTIONS=,
 SCHEMA=,

Publishing Models to a Model Table on the CAS Server

Publishing a model to CAS involves adding or replacing a row in a model table. In order to publish a model to an existing model table, the model table must first be loaded on the CAS server. If the specified model table has been loaded on the CAS server, then the model table is updated. Otherwise, a new model table is created.

A model can be published directly to a global model table in CAS. A model is published to a global model table in CAS when the PUBLISHMODEL PUBLISHGLOBAL= parameter is set to TRUE.

Naming Requirements

Data source column and table names must follow the vendor's naming conventions.

Syntax: SCOREACCEL Procedure

PROC SCOREACCEL *<optional-argument>;*

PUBLISHMODEL *required-arguments <publish-model-optional-arguments>;*

RUNMODEL *required-arguments <run-model-optional-arguments>;*

DELETEMODEL *required-arguments <delete-model-optional-arguments>;*

Statement	Task	Example
PROC SCOREACCEL	Publish, execute, or delete a model in CAS or an external data source	Ex. 1, Ex. 2, Ex. 3, Ex. 4, Ex. 5, Ex. 6, Ex. 7, Ex. 8, Ex. 9, Ex. 10
PUBLISHMODEL	Publish a model in CAS or an external data source	Ex. 1, Ex. 2, Ex. 3, Ex. 4, Ex. 5, Ex. 6, Ex. 7, Ex. 8, Ex. 9, Ex. 10

Statement	Task	Example
RUNMODEL	Run a model in CAS or an external data source	Ex. 1 , Ex. 2 , Ex. 3 , Ex. 4 , Ex. 5 , Ex. 6 , Ex. 7 , Ex. 8 , Ex. 9 , Ex. 10
DELETEMODEL	Delete a model from CAS or an external data source	Ex. 1 , Ex. 2 , Ex. 3 , Ex. 4 , Ex. 5 , Ex. 6 , Ex. 7 , Ex. 8 , Ex. 9 , Ex. 10

PROC SCOREACCEL Statement

Publishes, executes, or deletes a model in CAS or an external data source.

Examples:

[“Example 1: Publish, Run, and Delete a DATA Step Model in CAS” on page 2013](#)

[“Example 2: Publish, Run, and Delete a Model in Teradata” on page 2015](#)

[“Example 3: Publish a Model to a Hive Table and Run in Cloudera” on page 2018](#)

[“Example 4: Publish a Model to a Spark Table and Run in Databricks on AWS” on page 2021](#)

[“Example 5: Publish a Model to a Spark Table and Run in Azure Databricks” on page 2023](#)

[“Example 6: Publish a Model to Amazon S3 and Run in Databricks on AWS” on page 2026](#)

[“Example 7: Publish a Model to ADLS and Run in Azure Databricks” on page 2028](#)

[“Example 8: Publish a Model to an SQL Server Table and Run in Azure Synapse” on page 2030](#)

[“Example 9: Publish a Model to ADLS and Run in Azure Synapse” on page 2032](#)

[“Example 10: Publish, Run, and Delete a Model in SingleStore” on page 2036](#)

Syntax

```
PROC SCOREACCEL <SESSREF=session-reference | SESSUUID="session-uuid">;
```

Arguments

SESSREF=*session-reference* | **SESSUUID**=*"session-uuid"*

specifies the name of a CAS session or the universally unique identifier (UUID) of an existing CAS session to which you want to connect. If no option is specified, the automatic CAS session, CASAUTO, is used.

SESSREF=*session-reference*

specifies the name of a CAS session to which you want to connect.

SESSUUID="session-uuid"

specifies the universally unique identifier (UUID) of an existing CAS session. You must obtain the SESSUUID from the existing session before you can specify it in this option. The engine connects to the session that is identified in the UUID.

DELETEMODEL Statement

Deletes a model from CAS or an external data source.

Requirement: For Hadoop, beginning in 2023.09, security updates have been introduced by a newer version of the Java Runtime Environment. For a summary of changes in deployment, configuration, and usage, see [“Publishing and Running Models in Amazon EMR or Cloudera” in SAS In-Database Products: User’s Guide](#).

Syntax

DELETEMODEL**CASLIB="caslib"****MODELNAME="model-name"****MODELTABLE="model-table" | "caslib.model-table" | "schema.model-table"****<delete-model-optional-arguments>;**

Required Arguments

CASLIB="casref"

specifies the name of the caslib that is associated with the data source where the model is published. PROC SCOREACCEL uses the caslib to obtain options that are specific to the data source. For example, the caslib can specify user credentials and other connection options.

Applies to **CAS**

Databricks

Filesystem

Hadoop

SingleStore (SAS SpeedyStore)

Synapse

Teradata

Interactions Specify connection options in a caslib assignment. Specifying connection options in the procedure is deprecated. The CASLIB= option is recommended and will be required in the future for all EXTTYPE= data sources. Currently, the CASLIB= option is required

for publishing and deleting models with EXTTYPE= values of FILESYSTEM, SINGLESTORE, SPARK, and SYNAPSE.

For CAS or Teradata, a caslib that is specified as the first part of a two-part model table name in MODELTABLE= takes precedence over the value in this option.

MODELNAME="*model-name*"

specifies the name of the model to be deleted.

Alias MODEL=

Applies to CAS

Filesystem

Databricks

Hadoop

SingleStore (SAS SpeedyStore)

Synapse

Teradata

Note For SingleStore, SAS publishes each model as a table. For Hadoop, Databricks, and Synapse, if you omit the MODELDIR= option, SAS publishes each model as a table (Hive table, Spark table, or SQL Server table, depending on the data source). In all of these cases, SAS uses the model name as the table name and adds a prefix of **sasmodel_** in order to differentiate from other tables in the database. When you reference the model to publish it, delete it, or run it, do not include the **sasmodel_** prefix. Do not use the MODELTABLE= option.

MODELTABLE="*model-table*" | "*caslib.model-table*" | "*schema.model-table*"

specifies the name of the table that contains the model to be deleted when deleting a model from CAS or Teradata.

For CAS, the model table can be specified as a one-part or a two-part name. The first part of a two-part name is interpreted as the name of the caslib that contains the model table.

Applies to CAS

Teradata

Requirements This parameter is required when deleting a model from CAS or Teradata.

Deleting a model from CAS involves removing a row from a model table. In order to remove the model, the model table must first be loaded to the CAS server.

Optional Arguments

AUTHDOMAIN="authentication-domain"

specifies the name of the authentication domain that contains the credentials (user name and password) that are used to access Teradata.

Applies to Teradata

CAUTION Connection options in the procedure are deprecated. Specify this option in the caslib assignment instead.

AUTHTOKEN="authentication-token"

specifies the authentication token that is used to establish a connection with a Spark cluster.

Applies to Hadoop

CAUTION Connection options in the procedure are deprecated. Specify this option in the caslib assignment instead.

CLASSPATH="class_path"

specifies the class path used in the Hadoop call context. The class path can be a folder or individual JAR files. The Hadoop configuration folder must be included in the class path.

Applies to Hadoop

CAUTION This option is no longer supported. Beginning with 2023.09, modify your code to remove this option.

CLUSTERID="cluster-id"

specifies the identifier of the Spark cluster.

Applies to Hadoop

CAUTION Connection options in the procedure are deprecated. Specify this option in the caslib assignment instead.

DATABASE="database-name"

specifies the name of the database.

Default "" (a zero-length string)

Applies to Databricks

Filesystem

Hadoop

SingleStore (SAS SpeedyStore)

Synapse

Teradata

Notes For Teradata, this option provides the connection option that names the Teradata database to be accessed.

For Filesystem, Hadoop, SingleStore, Spark, or Synapse, the DATABASE= option is an alias for the SCHEMA= option.

DELETEGLOBAL=TRUE | FALSE

specifies whether the model is deleted from a global model table in CAS.

TRUE

deletes the model from the global model table.

Alias YES

FALSE

deletes the model from the model table in the local CAS session, leaving the global table unaltered.

Alias NO

Default FALSE

Applies to CAS

EXTTYPE=CAS | DATABRICKS | FILESYSTEM | HADOOP | SINGLESTORE | SYNAPSE | TERADATA

specifies the target environment to which the model is to be deleted.

CAS

specifies to delete the model from a model table in CAS.

DATABRICKS

specifies to delete the model from Databricks.

FILESYSTEM

specifies to delete the model from a cloud-based file system. FILESYSTEM supports Microsoft Azure Data Lake Storage Gen2 (ADLS) and Amazon S3.

Note: The FILESYSTEM value is deprecated as of 2023.12. For Databricks and Synapse, you are no longer required to publish a model to a file system.

HADOOP

specifies to delete the model from the Hadoop server.

SINGLESTORE

specifies to delete the model from the SingleStore cluster. This value is supported as of [2023.05](#).

SYNAPSE

specifies to delete the model from Azure Synapse. This value is supported as of [2023.12](#) to delete a model that was published to an SQL Server table.

TERADATA

specifies to delete the model from a model table in the Teradata database.

Alias	TARGET=
Default	CAS
Applies to	CAS
	Databricks
	Filesystem
	Hadoop
	SingleStore (SAS SpeedyStore)
	Synapse
	Teradata

MODELDATABASE="model-database"

specifies the database where the model is stored. If this option is not specified, then the database that is specified in the SCHEMA= option of the caslib is used. This option is supported for storing a model in a Hive table or in a Spark table. If you specify the MODELDIR= option, then the MODELDATABASE= option is ignored.

Applies to	Databricks
	Hadoop
Requirement	When you publish a model, if MODELDATABASE= is specified, the database must exist already.
Note	This option is supported as of 2023.09 .

MODELDIR="model-directory"

specifies the root folder where the model directory is stored.

For Hadoop, in releases prior to [2023.09](#), models were always published to HDFS. Now SAS uses WebHDFS to access models that are stored in HDFS. If you want to access models in HDFS, you must specify the webHdfsUrl= option in the Hadoop caslib. However, this method is not recommended. In a future release, publishing to HDFS will not be supported.

For Hadoop, as of [2023.09](#) you can publish a model to a Hive table, and this method is recommended. In order to publish, delete, or run a model that is stored in a Hive table, omit the MODELDIR= option from PROC SCOREACCEL. When you omit the MODELDIR= option, the SERVER= parameter in the Hadoop caslib must specify the Hive server.

Applies to	Filesystem
	Hadoop

MODELSHEMA="model-schema"

specifies the schema for the stored model. This option is supported for storing a model in an SQL Server table. If you specify the MODELDIR= option, then the MODELSHEMA= option is ignored.

Applies to Synapse

Requirement For Synapse, when deleting, publishing, or running a model that is stored in an SQL Server table, the MODELSHEMA= option must be specified.

Note This option is supported as of [2023.12](#).

PASSWORD="password"

is the password for the user ID on ADLS, the Hadoop server, or the Teradata server. On ADLS this is value known as the secret.

Aliases PASS=
PASSWD=
PWD=

Applies to Filesystem
Hadoop
Teradata

CAUTION Connection options in the procedure are deprecated. Specify this option in the caslib assignment instead.

PERSISTTABLE=TRUE | FALSE

specifies whether the updated model table, that results from deleting a model, should be saved to the caslib data source associated with the table.

TRUE

saves the updated model table to the data source.

Alias YES

FALSE

does not save the updated model table to the data source.

Alias NO

Default FALSE

Applies to CAS

Requirement If the model table already exists in the caslib data source, you must also specify REPLACETABLE=TRUE to save the table.

See [REPLACETABLE= option](#)

PROMOTETABLE=TRUE | FALSE

specifies whether the updated model table, that results from deleting a model, should be promoted to global scope on the CAS server.

TRUE

promotes the updated model table to global scope.

Alias YES

FALSE

does not promote the updated model table to global scope.

Alias NO

Default FALSE

Applies to CAS

REPLACETABLE=TRUE | FALSE

specifies whether to allow an existing model table, that results from deleting a model, to be replaced when the updated model table is saved to the caslib data source.

TRUE

replaces the model table.

Alias YES

FALSE

does not replace the model table.

Alias NO

Default TRUE

Applies to CAS

Requirement You must also specify PERSISTTABLE=TRUE to save the model table to the caslib data source.

See [PERSISTTABLE= option](#)

SCHEMA="*schema-name*"

specifies the name of the database or schema.

Applies to Databricks

Filesystem

Hadoop

SingleStore (SAS SpeedyStore)

Synapse

Teradata

Notes For Teradata, this option provides the connection option that the Teradata database uses to qualify the Teradata tables.

For Filesystem, Hadoop, SingleStore, Spark, or Synapse, the DATABASE= option is an alias for the SCHEMA= option.

SERVER="server"

specifies the name of the Teradata server.

Applies to Teradata

CAUTION **Connection options in the procedure are deprecated.** Specify this option in the caslib assignment instead.

TIMEOUT="n"

specifies the number of seconds for the procedure to attempt an operation. If the operation does not complete by *n* seconds, the operation is terminated.

Applies to Filesystem

Restriction This option is supported for ADLS only.

USERNAME="id"

is an authorized user ID on the Hadoop or Teradata server.

Aliases USER=
USERID=
UID=

Applies to Hadoop
Teradata

CAUTION **Connection options in the procedure are deprecated.** Specify this option in the caslib assignment instead.

WEBHDFSURL="webhdfs-url"

specifies the URL used to access the Hadoop distributed file system through the REST API.

Applies to Hadoop

Note Use this argument when the platform is configured to access the distributed file system through the REST API.

CAUTION **Connection options in the procedure are deprecated.** Specify this option in the caslib assignment instead.

PUBLISHMODEL Statement

Publishes a model in CAS or an external data source.

Requirement: For Hadoop, beginning in 2023.09, security updates have been introduced by a newer version of the Java Runtime Environment. For a summary of changes in deployment, configuration, and usage, see [“Publishing and Running Models in Amazon EMR or Cloudera” in SAS In-Database Products: User’s Guide](#).

Syntax

PUBLISHMODEL

```
CASLIB="caslib"
MODELNAME="model-name"
MODELTYPE=DATASTEP | DS2
MODELTABLE="model-table" | "caslib.model-table" | "schema.model-table"
STOREFILES=file-specification | PROGRAMFILE=file-specification
VARXMLFILE=file-path
< publish-model-optional-arguments> ;
```

Required Arguments

CASLIB="casref"

specifies the name of the caslib that is associated with the data source where the model is published. PROC SCOREACCEL uses the caslib to obtain options that are specific to the data source. For example, the caslib can specify user credentials and other connection options.

Applies to CAS

Databricks

Filesystem

Hadoop

SingleStore (SAS SpeedyStore)

Synapse

Teradata

Interactions Specify connection options in a caslib assignment. Specifying connection options in the procedure is deprecated. The CASLIB= option is recommended and will be required in the future for all EXTTYPE= data sources. Currently, the CASLIB= option is required for publishing and deleting models with EXTTYPE= values of FILESYSTEM, SINGLESTORE, SPARK, and SYNAPSE.

For CAS or Teradata, a caslib that is specified as the first part of a two-part model table name in MODELTABLE= takes precedence over the value in this option.

MODELNAME="model-name"

specifies the name of the model to be published.

Alias MODEL=

Applies to CAS

Databricks

Filesystem

Hadoop

SingleStore (SAS SpeedyStore)

Synapse

Teradata

Note For SingleStore, SAS publishes each model as a table. For Hadoop, Databricks, and Synapse, if you omit the MODELDIR= option, SAS publishes each model as a table (Hive table, Spark table, or SQL Server table, depending on the data source). In all of these cases, SAS uses the model name as the table name and adds a prefix of **sasmodel_** in order to differentiate from other tables in the database. When you reference the model to publish it, delete it, or run it, do not include the **sasmodel_** prefix. Do not use the MODELTABLE= option.

MODELTABLE="model-table" | "caslib.model-table" | "schema.model-table"

specifies the name of the table to which the model is published in CAS or Teradata.

For CAS, the model table can be specified as a one-part or a two-part name. The first part of a two-part name is interpreted as the name of the caslib that contains the model table.

Applies to CAS

Teradata

Interaction The caslib that is specified as the first part of a two-part model table name takes precedence over the caslib that is specified with the CASLIB= option.

Note Publishing a model to CAS involves adding or replacing a row in a model table. In order to publish a model to an existing model table, the model table must first be loaded onto the CAS server. If the specified model table has been loaded onto the CAS server, then the model table is updated. Otherwise, a new model table is created.

MODELTYPE=DATASTEP | DS2

specifies the type of the input model program.

DATASTEP

specifies that the input model program is DATA step code.

Alias DS

DS2

specifies that the input model program is DS2 code.

Default DS2

Applies to CAS

Databricks

Filesystem

Hadoop

SingleStore (SAS SpeedyStore)

Synapse

Teradata

Note The MODELTYPE= option is required only for DATA step models. The DATA step code is converted to DS2 code before being bundled into an item store

STOREFILES=("*file-path-1*" | *fileref-1* <,"*file-path-2*" | *fileref-2*, ...>)

specifies a list of analytic store .sasast files, one for each analytic store to be published. The list of files can be a mixture of file path names and file references.

You must specify either STOREFILES= or PROGRAMFILE= or both options. Some analytic store models require only the STOREFILES= option. Some DATA step and DS2 models require only the PROGRAMFILE= option. Some models consist of DATA step code, DS2 code, and one or more ASTORE files. In that case, you specify both STOREFILES= and PROGRAMFILE= options.

If you publish an analytic store without a model program, then a DS2 model program is generated from the analytic store.

To specify an associated model program, use the PROGRAMFILE= option. When an analytic store is published, the associated model program is usually DS2 code.

Alias STOREFILE=

Applies to CAS

Databricks

Filesystem

Hadoop

SingleStore (SAS SpeedyStore)

Synapse

Teradata

Interactions If a single analytic store model is specified without an accompanying DS2 program, the KEEPLIST option can also be specified.

You must specify either STOREFILES= or PROGRAMFILE= or both options.

The STOREFILES= option is used only when MODELTYPE=DS2.
The option is not used when MODELTYPE=DATASTEP.

PROGRAMFILE="file-path" | fileref

specifies the file that contains the model program to be published. You must specify either STOREFILES= or PROGRAMFILE= or both options. Some analytic store models require only the STOREFILES= option. Some DATA step and DS2 models require only the PROGRAMFILE= option. Some models consist of DATA step code, DS2 code, and one or more ASTORE files. In that case, you specify both STOREFILES= and PROGRAMFILE= options.

Applies to CAS

Databricks

Filesystem

Hadoop

SingleStore (SAS SpeedyStore)

Synapse

Teradata

Interaction You must specify either STOREFILES= or PROGRAMFILE= or both options.

VARXMLFILE="file-path" | fileref

specifies the file that contains the variable metadata XML to be used during translation of an input DATA step model program to DS2.

Alias XMLFILE=

Applies to CAS

Databricks

Filesystem

Hadoop

SingleStore (SAS SpeedyStore)

Synapse

Teradata

Interaction The VARXMLFILE= option is required by some models when MODELTYPE=DATASTEP. The option is not used when MODELTYPE=DS2.

Optional Arguments

AUTHDOMAIN="authentication-domain"

specifies the name of the authentication domain that contains the credentials (user name and password) that are used to access Teradata.

Applies to Teradata

CAUTION **Connection options in the procedure are deprecated.** Specify this option in the caslib assignment instead.

AUTHTOKEN="authentication-token"

specifies the authentication token that is used to establish a connection with a Spark cluster.

Applies to Hadoop

CAUTION **Connection options in the procedure are deprecated.** Specify this option in the caslib assignment instead.

CLASSPATH="class_path"

specifies the class path used in the Hadoop call context. The class path can be a folder or individual JAR files. The Hadoop configuration folder must be included in the class path.

Applies to Hadoop

CAUTION **This option is no longer supported.** Beginning with 2023.09, modify your code to remove this option.

CLUSTERID="cluster-id"

specifies the identifier of the Spark cluster.

Applies to Hadoop

CAUTION **Connection options in the procedure are deprecated.** Specify this option in the caslib assignment instead.

DATABASE="database-name"

specifies the name of the database.

Default "" (a zero-length string)

Applies to Databricks

	Filesystem
	Hadoop
	SingleStore (SAS SpeedyStore)
	Synapse
	Teradata
Notes	For Teradata, this option provides the connection option that names the Teradata database to be accessed.
	For Filesystem, Hadoop, SingleStore, Spark, or Synapse, the DATABASE= option is an alias for the SCHEMA= option.

EXTTYPE=CAS | DATABRICKS | FILESYSTEM | HADOOP | SINGLESTORE | SYNAPSE | TERADATA

specifies the target environment to which the model is published.

CAS

specifies to publish to a model table in CAS.

DATABRICKS

specifies to publish the model to Databricks. This value is supported as of 2023.10 to publish a model to a Spark table.

FILESYSTEM

specifies to publish the model to a cloud-based file system. FILESYSTEM supports Microsoft Azure Data Lake Storage Gen2 (ADLS) and Amazon S3.

Note: The FILESYSTEM value is deprecated as of 2023.12. For Databricks and Synapse, you are no longer required to publish a model to a file system.

HADOOP

specifies to publish the model to the Hadoop server.

SINGLESTORE

specifies to publish the model to the SingleStore cluster. This value is supported as of 2023.05.

SYNAPSE

specifies to publish the model to Azure Synapse. This value is supported as of 2023.12 to publish a model to an SQL Server table.

TERADATA

specifies to publish to a model table in the Teradata database.

Alias TARGET=

Default CAS

Applies to CAS

Databricks

Filesystem

Hadoop

SingleStore (SAS SpeedyStore)

Synapse

Teradata

FORMATFILE="*file-path*" | *fileref*

specifies the file that contains the user-defined format XML definition to be published.

Applies to CAS

Databricks

Filesystem

Hadoop

SingleStore (SAS SpeedyStore)

Synapse

Teradata

Note This option is valid for DS2 models or DATA step models.

FORMATITEMSTOREFILE="*file-path*" | *fileref*

specifies the file that contains the format item store to be published.

Applies to CAS

Databricks

Filesystem

Hadoop

SingleStore (SAS SpeedyStore)

Synapse

Teradata

KEEPLIST=TRUE | FALSE

specifies whether to include a KEEP statement in the DS2 model program that is automatically generated from an analytic store model.

TRUE

includes a KEEP statement.

Alias YES

FALSE

does not include a KEEP statement.

Alias NO

Default FALSE

Applies to CAS

Databricks

Filesystem

Hadoop

SingleStore (SAS SpeedyStore)

Synapse

Teradata

Interaction KEEPLIST= can be specified when the STOREFILES= or STORETABLES= option contains a single analytic store model, and an accompanying DS2 model program is not specified.

MODELDATABASE="model-database"

specifies the database where the model is stored. If this option is not specified, then the database that is specified in the SCHEMA= option of the caslib is used. This option is supported for storing a model in a Hive table or in a Spark table. If you specify the MODELDIR= option, then the MODELDATABASE= option is ignored.

Applies to Databricks

Hadoop

Requirement When you publish a model, if MODELDATABASE= is specified, the database must exist already.

Note This option is supported as of [2023.09](#).

MODELDIR="model-directory"

specifies the root folder where the model directory is stored.

For Hadoop, in releases prior to [2023.09](#), models were always published to HDFS. Now SAS uses WebHDFS to access models that are stored in HDFS. If you want to access models in HDFS, you must specify the webHdfsUrl= option in the Hadoop caslib. However, this method is not recommended. In a future release, publishing to HDFS will not be supported.

For Hadoop, as of [2023.09](#) you can publish a model to a Hive table, and this method is recommended. In order to publish, delete, or run a model that is stored in a Hive table, omit the MODELDIR= option from PROC SCOREACCEL. When you omit the MODELDIR= option, the SERVER= parameter in the Hadoop caslib must specify the Hive server.

Applies to Filesystem

Hadoop

MODELNOTES="model-notes"

specifies the model notes to be written to the model table.

Applies to CAS

SingleStore (SAS SpeedyStore)

Teradata

MODELSHEMA="model-schema"

specifies the schema for the stored model. This option is supported for storing a model in an SQL Server table. If you specify the MODELDIR= option, then the MODELSHEMA= option is ignored.

Applies to Synapse

Requirement For Synapse, when deleting, publishing, or running a model that is stored in an SQL Server table, the MODELSHEMA= option must be specified.

Note This option is supported as of [2023.12](#).

MODELUUID="model-uuid"

specifies that the Model UUID is written to the model table.

Applies to CAS

SingleStore (SAS SpeedyStore)

Teradata

OUTDIR="work-directory"

specifies the local output directory that contains the program file that was converted from DATA step to DS2.

Applies to CAS

Databricks

Filesystem

Hadoop

SingleStore (SAS SpeedyStore)

Synapse

Teradata

PASSWORD="password"

is the password for the user ID on ADLS, the Hadoop server, or the Teradata server. On ADLS this is value known as the secret.

Aliases PASS=
 PASSWD=
 PWD=

Applies to Filesystem
 Hadoop
 Teradata

CAUTION **Connection options in the procedure are deprecated.** Specify this option in the caslib assignment instead.

PERSISTTABLE=TRUE | FALSE

specifies whether the updated model table should be saved to the caslib data source associated with the table.

TRUE

saves the updated model table to the data source.

Alias YES

FALSE

does not save the updated model table to the data source.

Alias NO

Default FALSE

Applies to CAS

Requirement If the model table already exists in the caslib data source, you must also specify REPLACETABLE=TRUE to save the table.

See [REPLACETABLE= option](#)

PROMOTETABLE=TRUE | FALSE

specifies whether the updated model table should be promoted to global scope on the CAS server.

TRUE

promotes the updated model table to global scope.

Alias YES

FALSE

does not promote the updated model table to global scope.

Alias NO

Default FALSE

Applies to CAS

PUBLISHGLOBAL=TRUE | FALSE

specifies whether the model is published to a global model table in CAS.

TRUE

Alias YES

FALSE

publishes the model to the model table in the local CAS session, leaving the global model table unaltered.

Alias NO

Default FALSE

Applies to CAS

REPLACEMODEL=TRUE | FALSE

specifies whether to allow an existing model in the model table to be replaced by the model being published.

TRUE

replaces the model in the model table.

Alias YES

FALSE

does not replace the model in the model table.

Alias NO

Default TRUE

Applies to CAS

Databricks

Filesystem

Hadoop

SingleStore (SAS SpeedyStore)

Synapse

Teradata

REPLACETABLE=TRUE | FALSE

specifies whether to allow an existing model table to be replaced when the updated model table is saved to the caslib data source.

TRUE

replaces the model table.

Alias YES

FALSE

does not replace the model table.

Alias NO

Default TRUE

Applies to CAS

Requirement You must also specify PERSISTTABLE=TRUE to save the model table to the caslib data source.

See [PERSISTTABLE= option](#)

SCHEMA="schema-name"

specifies the name of the database or schema.

Applies to Databricks

Filesystem

Hadoop

SingleStore (SAS SpeedyStore)

Synapse

Teradata

Notes For Teradata, this option provides the connection option that the Teradata database uses to qualify the Teradata tables.

For Filesystem, Hadoop, SingleStore, Spark, or Synapse, the DATABASE= option is an alias for the SCHEMA= option.

SERVER="server"

specifies the name of the Teradata server.

Applies to Teradata

CAUTION Connection options in the procedure are deprecated. Specify this option in the caslib assignment instead.

STORETABLES=("store-table-1" | "caslib.store-table-1" <,"store-table-2" | "caslib.store-table-2", ...>)

specifies one or more CAS blob table names that contain the analytic stores to be published. Each table must include a VARBINARY column named "_state_" that contains the analytic store blob. If you publish an analytic store without a model program, then a DS2 model program is generated from the analytic store.

To specify an associated model program, use the PROGRAMFILE= option. When an analytic store is published, the associated model program is usually DS2 code.

Applies to CAS

Databricks

Filesystem

Hadoop

SingleStore (SAS SpeedyStore)

Synapse

Teradata

Interactions If a single analytic store model is specified without an accompanying DS2 program, then the KEEPLIST option can also be specified.

If a single analytic store is specified using the STORETABLES= option, then the PROGRAMFILE= option is not required.

TIMEOUT="*n*"

specifies the number of seconds for the procedure to attempt an operation. If the operation does not complete by *n* seconds, the operation is terminated.

Applies to Filesystem

Restriction This option is supported for ADLS only.

USERNAME="*id*"

is an authorized user ID on the Hadoop or Teradata server.

Aliases USER=

USERID=

UID=

Applies to Hadoop

Teradata

CAUTION **Connection options in the procedure are deprecated.** Specify this option in the caslib assignment instead.

WEBHDFSURL="*webhdfs-url*"

specifies the URL used to access the Hadoop distributed file system through the REST API.

Applies to Hadoop

Note Use this argument when the platform is configured to access the distributed file system through the REST API.

CAUTION **Connection options in the procedure are deprecated.** Specify this option in the caslib assignment instead.

RUNMODEL Statement

Runs a model in CAS or an external data source.

Requirement: For Hadoop, beginning in 2023.09, security updates have been introduced by a newer version of the Java Runtime Environment. For a summary of changes in deployment, configuration, and usage, see [“Publishing and Running Models in Amazon EMR or Cloudera” in SAS In-Database Products: User’s Guide](#).

Syntax

RUNMODEL

CASLIB=*"caslib"*

MODELNAME=*"model-name"*

MODELTABLE=*"model-table" | "caslib.model-table" | "schema.model-table"*

<run-model-optional-arguments>;

Required Arguments

CASLIB=*"casref"*

specifies the name of the caslib that is associated with the data source where the model is executed. PROC SCOREACCEL uses the caslib to obtain options that are specific to the data source. For example, the caslib can specify user credentials and other connection options.

Applies to CAS

Hadoop

SingleStore (SAS SpeedyStore)

Spark

Teradata

Requirement For EXTTYPE=SPARK, use the following parameters in the Spark caslib: The platform= option specifies the external target. The schema= is either the Synapse SQL database name or Spark database name. The username= is the application ID, and the password= is the application ID’s secret. You must specify all connection options in the caslib.

Interactions Specify connection options in a caslib assignment. Specifying connection options in the procedure is deprecated. The CASLIB= option is recommended and will be required in the future for all EXTTYPE= data sources. Currently, the CASLIB= option is required for publishing and deleting models with EXTTYPE= values of SINGLESTORE and SPARK.

For CAS or Teradata, a caslib that is specified as the first part of a two-part model table name in MODELTABLE= takes precedence over the value in this option.

MODELNAME="*model-name*"

specifies the name of the model to run.

Alias MODEL=

Applies to CAS

Hadoop

SingleStore (SAS SpeedyStore)

Spark

Teradata

Note For SingleStore, SAS publishes each model as a table. For Hadoop, Databricks, and Synapse, if you omit the MODELDIR= option, SAS publishes each model as a table (Hive table, Spark table, or SQL Server table, depending on the data source). In all of these cases, SAS uses the model name as the table name and adds a prefix of **sasmodel_** in order to differentiate from other tables in the database. When you reference the model to publish it, delete it, or run it, do not include the **sasmodel_** prefix. Do not use the MODELTABLE= option.

MODELTABLE="*model-table*" | "*caslib.model-table*" | "*schema.model-table*"

specifies the name of the table that contains the model to run. When running a model in CAS, the model table can be specified as a one-part or a two-part name. The first part of a two-part name is interpreted as the name of the caslib that contains the model table.

Applies to CAS

Teradata

Interaction The caslib that is specified as the first part of a two-part model table name takes precedence over the caslib that is specified with the CASLIB= option.

Optional Arguments

AUTHDOMAIN="*authentication-domain*"

specifies the name of the authentication domain that contains the credentials (user name and password) that are used to access Teradata.

Applies to Teradata

CAUTION **Connection options in the procedure are deprecated.** Specify this option in the caslib assignment instead.

AUTHTOKEN="authentication-token"

specifies the authentication token that is used to establish a connection with a Spark cluster.

Applies to Hadoop

CAUTION **Connection options in the procedure are deprecated.** Specify this option in the caslib assignment instead.

CLASSPATH="class_path"

specifies the class path used in the Hadoop call context. The class path can be a folder or individual JAR files. The Hadoop configuration folder must be included in the class path if you are not specifying the CONFIGPATH= option. If you are using Spark, Hadoop JAR files and Spark JAR files must be located in separate folders, and the Spark JAR folder must be specified as the last entry in the class path.

Applies to Hadoop

CAUTION **This option is no longer supported.** Beginning with 2023.09, modify your code to remove this option.

CLUSTERID="cluster-id"

specifies the identifier of the Spark cluster.

Applies to Hadoop

CAUTION **Connection options in the procedure are deprecated.** Specify this option in the caslib assignment instead.

CONFIGPATH="configuration-path"

specifies a single folder where all the Hadoop and Spark configuration files reside.

Applies to Hadoop

Spark

CAUTION **Connection options in the procedure are deprecated.** Specify this option in the caslib assignment instead.

CUSTOMJAR="file-path"

specifies the local JAR file that contains the user-provided custom reader. The custom JAR file is automatically copied to the Hadoop cluster during job submission.

Applies to Hadoop

CAUTION **This option is no longer supported.** Beginning with 2023.09, modify your code to remove this option.

DATABASE="database-name"

specifies the name of the database.

Default "" (a zero-length string)

Applies to Hadoop

Spark

SingleStore (SAS SpeedyStore)

Teradata

Notes For Teradata, this option provides the connection option that names the Teradata database to be accessed.

For Hadoop, SingleStore, or Spark, the DATABASE= option is an alias for the SCHEMA= option.

DBMAXTEXT=number-of-bytes

specifies the maximum number of bytes to allocate for STRING data type columns. This option does not apply to CHAR or VARCHAR columns.

Default 32767

Range 1–32767

Applies to Hadoop

EOPTIONS="options-string"

specifies properties that are passed to the SAS Embedded Process. For SingleStore, the properties `ds2.dbMaxText=number` and `logger.level=value` are valid. For information about these properties, see [“epProperties= Data Connector Option” in SAS Cloud Analytic Services: User’s Guide](#).

For Teradata, this option is supported in legacy programs only. The properties are passed to the SAS_SCORE_EP stored procedure in its OPTIONS parameter. For more information, see *SAS In-Database Products: User’s Guide*.

Applies to SingleStore (SAS SpeedyStore)

Teradata

Example `epOptions="ds2.dbMaxText=2048,logger.level=allon"`

EXTTYPE=CAS | HADOOP | SINGLESTORE | SPARK | TERADATA

specifies the target environment in which the model is to be run.

CAS

specifies to run the model in CAS.

HADOOP

specifies to run the model using the SAS Embedded Process for Hadoop.

SINGLESTORE

specifies to run the model using the SAS Embedded Process for SingleStore. This value is supported as of [2023.05](#).

SPARK

specifies to run the model using the SAS Embedded Process for Spark. The CASLIB= option is required. In the caslib use the PLATFORM= option to

specify the external target. For more information, see [“Interaction with SAS Data Connector to Spark” in SAS In-Database Products: User’s Guide](#).

Note: The DATABRICKS and SYNAPSE values have been deprecated. Use EXTTYPE=SPARK instead.

TERADATA

specifies to run the model using the SAS Embedded Process for Teradata.

Alias TARGET=

Default CAS

Applies to CAS

Hadoop

SingleStore (SAS SpeedyStore)

Spark

Teradata

FORCEOVERWRITE=TRUE | FALSE

specifies whether to force deletion of the output data directory before running the Hadoop MapReduce job.

TRUE

forces deletion of the output data directory.

Alias YES

FALSE

does not force deletion of the output data directory.

Alias NO

Default FALSE

Applies to Hadoop

Spark

Restriction For running models against SQL tables in Synapse, FORCEOVERWRITE=FALSE is not supported. The output table must not exist.

HIVEPORT=*integer*

specifies the Hive server port number.

Applies to Hadoop

CAUTION **Connection options in the procedure are deprecated.** Specify this option in the caslib assignment instead.

INDATASET="input-dataset"

specifies the name of the Spark dataset passed as input to the SAS Embedded Process.

Applies to Hadoop

Spark

Requirements When running a model in Hadoop, either INDATASET=, INHDMD=, or INTABLE= must be specified.

When running a model in Databricks or Synapse, either INDATASET= or INTABLE= must be specified.

See [INHDMD= option](#)

[INTABLE= option](#)

INHDMD="file-path"

specifies the name of the input Spark dataset file on HDFS.

Applies to Hadoop

Requirement When running a model in Hadoop, either INDATASET=, INHDMD=, or INTABLE= must be specified.

See [INDATASET= option](#)

[INTABLE= option](#)

CAUTION **This option is deprecated.** In the future, it will not be supported. Modify your code to remove this option.

INQUERY="sql-query"

specifies an SQL SELECT statement that defines the inputs to the SAS Embedded Process. The SQL query must be supported by the data source.

Applies to SingleStore (SAS SpeedyStore)

Teradata

Requirement When running a model in Teradata or SingleStore, either INTABLE= or INQUERY= must be specified.

See [INTABLE= option](#)

INTABLE="input-table" | "caslib.input-table" | "schema.input-table"

specifies the name of the input table. The input table can be specified as a one-part or a two-part name:

- When running a model in CAS, the first part of a two-part name is interpreted as the name of the caslib that contains the input table.
- When running in SingleStore, you must use a one-part name. Use the DATABASE= argument to specify the database name.

- When running in Synapse against an SQL Server table, use a two-part name. Specify the schema name followed by a dot and then the table name.
- When running a model in Teradata, the first part of a two-part name is interpreted as the name of the database schema that contains the input table.

Alias INPUTTABLE=

Applies to CAS
Hadoop
Spark
Teradata

Requirements When running a model in Hadoop, either INDATASET=, INHDMD=, or INTABLE= must be specified.

When running a model in Databricks or Synapse, either INDATASET= or INTABLE= must be specified.

When running a model in Teradata or SingleStore, either INTABLE= or INQUERY= must be specified.

See [INDATASET= option](#)
[INHDMD= option](#)
[INQUERY= option](#)

JOBMANAGEMENTURL="rest-url"

specifies the URL used to submit execution requests over a REST interface to services such as Apache Livy or Databricks Notebooks.

Alias RESTURL=

Applies to Hadoop

CAUTION Connection options in the procedure are deprecated. Specify this option in the caslib assignment instead.

KEEPLISTCOLUMNS="column-1 <column-2 ... >"

KEEPLISTCOLUMNS=(column-1 <column-2 ... >)

KEEPLISTCOLUMNS=("column-1" <"column-2" ... >)

specifies the name of the column (or columns) to be kept by the DS2 program.

Alias KEEPLISTCOLS=

Applies to Hadoop
Spark

Interaction The parameters KEEPLISTFILE and KEEPLISTCOLUMNS are mutually exclusive.

KEEPLISTFILE="file-path"

specifies the name of the file that contains a list of columns to be kept by the DS2 program.

Applies to Hadoop

Spark

Interaction The parameters KEEPLISTFILE and KEEPLISTCOLUMNS are mutually exclusive.

CAUTION **This option is deprecated.** In the future, it will not be supported. Modify your code to remove this option.

MODELDATABASE="model-database"

specifies the database where the model is stored. If this option is not specified, then the database that is specified in the SCHEMA= option of the caslib is used. This option is supported for storing a model in a Hive table or in a Spark table. If you specify the MODELDIR= option, then the MODELDATABASE= option is ignored.

Applies to Databricks

Hadoop

Requirement When you publish a model, if MODELDATABASE= is specified, the database must exist already.

Note This option is supported as of 2023.09.

MODELDIR="model-directory"

specifies the root folder where the model directory is stored.

For Databricks as of 2023.10 and Synapse as of 2023.12, you are no longer required to publish a model to a file system. You are not required to specify the MODELDIR= option if you are not running models that have been published to ADLS or Amazon S3.

For Databricks, when you specify MODELDIR=, the model directory includes the prefix **dbfs:** and references a storage object that has been mounted to Databricks File System (DBFS). You can publish a model to Amazon S3 or Microsoft Azure Data Lake Storage Gen2 (ADLS) and run that model in Databricks. Here is an example:

```
modeldir="dbfs:/mnt/mycontainer/mymodels"
```

For Hadoop, in releases prior to 2023.09, models were always published to HDFS. Now SAS uses WebHDFS to access models that are stored in HDFS. If you want to access models in HDFS, you must specify the webHdfsUrl= option in the Hadoop caslib. However, this method is not recommended. In a future release, publishing to HDFS will not be supported.

For Hadoop, as of 2023.09 you can publish a model to a Hive table, and this method is recommended. In order to publish, delete, or run a model that is stored in a Hive table, omit the MODELDIR= option from PROC SCOREACCEL. When you omit the MODELDIR= option, the SERVER= parameter in the Hadoop caslib must specify the Hive server.

Applies to Hadoop

Spark

MODELSHEMA="model-schema"

specifies the schema for the stored model. This option is supported for storing a model in an SQL Server table. If you specify the MODELDIR= option, then the MODELSHEMA= option is ignored.

Applies to Synapse

Requirement For Synapse, when deleting, publishing, or running a model that is stored in an SQL Server table, the MODELSHEMA= option must be specified.

Note This option is supported as of 2023.12.

OUTDATASET="output-dataset"

specifies the name of the output Spark dataset to be created by the SAS Embedded Process.

Applies to Hadoop

Spark

Requirements When running a model in Hadoop, either OUTHDMD=, OUTDATASET=, or OUTTABLE= must be specified.

When running a model in Databricks or Synapse, either OUTDATASET= or OUTTABLE= must be specified.

See [OUTHDMD= option](#)

[OUTTABLE= option](#)

OUTHDMD="file-path"

specifies the name of the output Hadoop metadata file that is created by the SAS Embedded Process.

Applies to Hadoop

Requirement When running a model in Hadoop, either OUTHDMD=, OUTDATASET=, or OUTTABLE= must be specified.

See [OUTDATASET= option](#)

[OUTTABLE= option](#)

CAUTION **This option is deprecated.** In the future, it will not be supported. Modify your code to remove this option.

OUTKEY="column-1 <column-2 ...>"

OUTKEY=(column-1 <column-2 ...>

OUTKEY=("column-1" <"column-2" ...>

specifies the name of one or more columns used for the primary index of the output table that is created by the SAS Embedded Process.

Applies to SingleStore (SAS SpeedyStore)

Teradata

OUTPUTFOLDER="directory-path"

specifies the name of the directory where the output files are stored.

Applies to Hadoop

CAUTION This option is deprecated. In the future, it will not be supported. Modify your code to remove this option.

OUTPUTFORMATCLASS="class-name"

specifies the name of the output format class in dot notation that is used to write the output records.

Applies to Hadoop

CAUTION This option is deprecated. In the future, it will not be supported. Modify your code to remove this option.

OUTRECORDFORMAT=BINARY | DELIMITED

specifies the format of the output record that is produced by the SAS Embedded Process for Hadoop.

BINARY

specifies that the output record is binary.

DELIMITED

specifies that the output record is delimited.

Default DELIMITED

Applies to Hadoop

CAUTION This option is deprecated. In the future, it will not be supported. Modify your code to remove this option.

OUTSCHEMA="output-schema"

specifies the Hive output data source schema name when running a model. This argument is appropriate for Databricks, Hadoop, or Synapse.

Applies to Hadoop

Spark

OUTTABLE="output-table" | "caslib.output-table" | "schema.output-table"

specifies the name of the output table. The output table can be specified as a one-part or a two-part name:

- When running a model in CAS, the first part of a two-part name is interpreted as the name of the caslib that contains the output table.
- When running in SingleStore, you must use a one-part name. Use the DATABASE= argument to specify the database name.
- When running in Synapse against an SQL Server table, use a two-part name. Specify the schema name followed by a dot and then the table name.
- When running a model in Teradata, the output table must be in the same database as the input table. The first part of a two-part name is interpreted as the name of the database schema that contains the output table.

Alias OUTPUTTABLE=

Applies to CAS
 Hadoop
 SingleStore (SAS SpeedyStore)
 Spark
 Teradata

Requirements When running a model in Hadoop, either OUTHDMD=, OUTDATASET=, or OUTTABLE= must be specified.
 When running a model in Databricks or Synapse, either OUTDATASET= or OUTTABLE= must be specified.
 When running a model in SingleStore or Teradata, OUTTABLE= must be specified.

See [OUTDATASET= option](#)
 [OUTHDMD= option](#)

OUTTABLEOPTIONS="options-string"

provides user-specified options that are appended to the Hive CREATE TABLE statement.

Applies to Hadoop
 Spark

CAUTION **This option is deprecated.** In the future, it will not be supported. Modify your code to remove this option.

PASSWORD="password"

is the password for the user ID on the Hadoop or Teradata server.

Aliases PASS=
 PASSWD=
 PWD=

Applies to Hadoop

Teradata

CAUTION **Connection options in the procedure are deprecated.** Specify this option in the caslib assignment instead.

PLATFORM=MAPRED | SPARK

specifies the platform where the SAS Embedded Process for Hadoop is to be executed.

MAPRED

specifies to run the model in MapReduce.

SPARK

specifies to run the model in Spark.

Default SPARK

Applies to Hadoop

CAUTION **This option is deprecated.** The value SPARK is now used automatically. Modify your code to remove this option.

PROPERTIES="name1=value" <PROPERTIES="name2=value"> | PROPERTIES=("name=value"<, "name=value", ...>)

specifies a Hadoop configuration property as a name-value pair. For multiple properties, specify a single argument containing a comma-separated list of name-value pairs enclosed in parentheses. Any Hadoop configuration property can be assigned using this option. The PROPERTIES parameter can also be specified multiple times, once for each property.

Applies to Hadoop

Spark

Requirement If your output is to a Hive table and the Hive database folder is under an HDFS encrypted zone, you must set the SAS Embedded Process temporary folder to a location that is under the same encrypted zone. To do this, set the sas.ep.tmpdir configuration property. Here is an example:

```
properties="sas.ep.tmpdir=yourSASEPTemporaryFolder"
```

SCHEMA="schema-name"

specifies the name of the database or schema.

Applies to Hadoop

Spark

SingleStore (SAS SpeedyStore)

Teradata

Notes For Teradata, this option provides the connection option that the Teradata database uses to qualify the Teradata tables.

For Hadoop, SingleStore, or Spark, the DATABASE= option is an alias for the SCHEMA= option.

SENDPROGRAM=TRUE | FALSE

specifies whether the model program source code should be sent back to the client and displayed for the user.

TRUE

sends the source code back to the client.

Alias YES

FALSE

does not send the source code back to the client.

Alias NO

Default FALSE

Applies to CAS

SERVER="server"

specifies the name of the Teradata server or Hive server.

Applies to Hadoop

Spark

Teradata

CAUTION Connection options in the procedure are deprecated. Specify this option in the caslib assignment instead.

TRACE

runs the SAS Embedded Process for Hadoop with traces on.

Applies to Hadoop

Spark

USERNAME="id"

is an authorized user ID on the Hadoop or Teradata server.

Aliases USER=

USERID=

UID=

Applies to Hadoop

Teradata

CAUTION **Connection options in the procedure are deprecated.** Specify this option in the caslib assignment instead.

VERBOSE

specifies that the SAS Embedded Process provide additional logging information.

Applies to Hadoop

Spark

CAUTION **This option is deprecated.** In the future, it will not be supported. Modify your code to remove this option.

Examples: SCOREACCEL Procedure

Example 1: Publish, Run, and Delete a DATA Step Model in CAS

Features:

- CAS LIBNAME statement
- PROC SCOREACCEL statement
- PUBLISHMODEL statement
- RUNMODEL statement
- DELETEMODEL statement

Details

In this PROC SCOREACCEL example, a DATA step model is published and run in CAS. The model is then deleted.

PROC SCOREACCEL translates DATA step code into DS2 code on the SAS client.

Program

```
cas mysess1;  
libname mycaslib cas casref=mysess1;
```

```

proc delete data=mycaslib.model01_score_data; run;
libname eminput "/mydir/scoring/model01";
data mycaslib.model01_score_data;
    set eminput.traindata;
    id=_n_; run;
quit;

proc scoreaccel sessref=mysess1;
    publishmodel
        modelname="model01"
        modeltype=datastep
        modeltable="modeltable1"
        programfile="/mydir/scoring/model01/score.sas"
        xmlfile="/mydir/scoring/model01/score.xml"
        outdir="/user1/score/work/"
        modelnotes="Simple model01 test model"
        promotetable=no
        persisttable=no
    ;
quit;

proc delete data=mycaslib.model01out_data run;
proc scoreaccel sessref=mysess1;
    runModel
        modelname="model01"
        modeltable="modeltable1"
        intable="model01_score_data"
        outtable="model01_out_data"
    ;
quit;

proc scoreaccel sessref=mysess1;
    deletemodel
        modelname="model01"
        modeltable="modeltable1"
    ;
quit;

```

Program Description

Assign a CAS LIBNAME. The CAS LIBNAME is passed to the DATA step in PROC DELETE.

```

cas mysess1;
libname mycaslib cas casref=mysess1;

```

Create an input scoring table in CAS. The DELETE procedure removes an existing input scoring table.

```

proc delete data=mycaslib.model01_score_data; run;
libname eminput "/mydir/scoring/model01";
data mycaslib.model01_score_data;
    set eminput.traindata;
    id=_n_; run;
quit;

```

Publish a model in CAS. The DATA step model is converted to DS2 in this step and is published in CAS.

```
proc scoreaccel sessref=mysess1;
  publishmodel
    modelname="model01"
    modeltype=datastep
    modeltable="modeltable1"
    programfile="/mydir/scoring/model01/score.sas"
    xmlfile="/mydir/scoring/model01/score.xml"
    outdir="/user1/score/work/"
    modelnotes="Simple model01 test model"
    promotetable=no
    persisttable=no
  ;
quit;
```

Run the model in CAS. The DELETE procedure removes an existing CAS table before running the model.

```
proc delete data=mycaslib.model01out_data run;
proc scoreaccel sessref=mysess1;
  runModel
    modelname="model01"
    modeltable="modeltable1"
    intable="model01_score_data"
    outtable="model01_out_data"
  ;
quit;
```

Delete the model from the model table in CAS.

```
proc scoreaccel sessref=mysess1;
  deletemodel
    modelname="model01"
    modeltable="modeltable1"
  ;
quit;
```

Example 2: Publish, Run, and Delete a Model in Teradata

Features:	CAS procedure
	PROC SCOREACCEL statement
	PUBLISHMODEL statement
	RUNMODEL statement
	DELETEMODEL statement

Details

This PROC SCOREACCEL example publishes and runs a DS2 model in Teradata. PROC SCOREACCEL invokes the model publishing and scoring action set in CAS to delete a model, publish a model, or run a model.

Program

```
libname mytdlib teradata server=mytdserver user=myuser password=XXXXX
    database=model;

proc delete data=mytdlib.model01_score_data;
run;
libname eminput "/mydir/scoring/model01";
data mytdlib.model01_score_data;
    set eminput.traindata;
    id=_n_;
run;
quit;

proc cas;
    session mysess1;
    action addCaslib /
        caslib="tdlib1"
        datasource={
            srcType="teradata",
            server="mytdserver",
            username="myuser",
            password="XXXXX",
            database="mymodel"
        };
run;

proc scoreaccel sessref=mysess1;
    publishmodel
        caslib="tdlib1"
        exttype=teradata
        modelname="model01"
        modeltable="modeltable1"
        programfile="/mydir/scoring/model01/score.ds2"
        modelnotes="Simple model01 test model"
    ;
run;
quit;

proc scoreaccel sessref=mysess1;
    runmodel
        caslib="tdlib1"
        exttype=teradata
        modelname="model01"
        modeltable="modeltable1"
```

```

        intable="model01_score_data"
        outtable="model01_out_data"
        outkey="id"
    ;
run;
quit;

proc scoreaccel sessref=mysess1;
    deletemodel
        caslib="tdlib1"
        exttype=teradata
        modelname="model01"
        modeltable="modeltable1"
    ;
run;
quit;

```

Program Description

Create the Teradata LIBNAME reference. The Teradata libref is needed for PROC DELETE and the DATA step program.

```

libname mytdlib teradata server=mytdserver user=myuser password=XXXXX
    database=model;

```

Create an input scoring table. The DELETE procedure removes an existing input scoring table if it exists.

```

proc delete data=mytdlib.model01_score_data;
run;
libname eminput "/mydir/scoring/model01";
data mytdlib.model01_score_data;
    set eminput.traindata;
    id=_n_;
run;
quit;

```

Define a Teradata caslib. The addCaslib action adds a CAS library to the current mysess1 session.

```

proc cas;
    session mysess1;
    action addCaslib /
        caslib="tdlib1"
        datasource={
            srcType="teradata",
            server="mytdserver",
            username="myuser",
            password="XXXXX",
            database="mymodel"
        };
run;

```

Publish a DS2 model to Teradata. The PROGRAMFILE statement contains the DS2 model.

```

proc scoreaccel sessref=mysess1;

```

```

publishmodel
  caslib="tdlib1"
  exttype=teradata
  modelname="model01"
  modeltable="modeltable1"
  programfile="/mydir/scoring/model01/score.ds2"
  modelnotes="Simple model01 test model"
;
run;
quit;

```

Run the DS2 model in Teradata. The input table and output table must be in the same Teradata database.

```

proc scoreaccel sessref=mysess1;
  runmodel
    caslib="tdlib1"
    exttype=teradata
    modelname="model01"
    modeltable="modeltable1"
    intable="model01_score_data"
    outtable="model01_out_data"
    outkey="id"
  ;
run;
quit;

```

If necessary, delete the model from the model table in Teradata.

```

proc scoreaccel sessref=mysess1;
  deletemodel
    caslib="tdlib1"
    exttype=teradata
    modelname="model01"
    modeltable="modeltable1"
  ;
run;
quit;

```

Example 3: Publish a Model to a Hive Table and Run in Cloudera

Features:	CASLIB statement
	PROC CAS
	PROC SCOREACCEL statement
	PUBLISHMODEL statement
	RUNMODEL statement
	DELETEMODEL statement

Details

In this example, a DS2 model is published to a Hive table. Then the model is run by using a Spark Livy interface. This example code is appropriate for Cloudera or Amazon EMR.

Note: In 2023.10, SAS changed the redistributed JDBC drivers used by SAS/ACCESS Interface to Hadoop and SAS/ACCESS Interface to Spark. Changes might be needed in your connection options. For more information, see [“Updates Might Be Needed Due to Driver Change” in SAS In-Database Products: User's Guide](#).

Program

```
cas mysess1;
caslib myhadoop datasource=(
    srcType="hadoop",
    server="Hive-server",
    port="port-number",
    userName="user-name",
    password="password",
    schema="database-name",
    hadoopConfigDir="Hadoop_config_path_directory",
    jobManagementURL="http://server.company.com:8998"
);

proc scoreaccel sessref=mysess1;
    publishmodel
        exttype=hadoop
        caslib="myhadoop"
        modelname="simple01"
        modeldatabase="mydatabase"
        programfile="/score/simple/simple.ds2";
run;
quit;

proc cas;
    sparkEmbeddedProcess.startSparkEP caslib="myhadoop"
        executorInstances=2 executorCores=2
        executorMemory=10 driverMemory=4;
run;
quit;

proc scoreaccel sessref=mysess1;
    runmodel
        exttype=hadoop
        caslib="myhadoop"
        modelname="simple01"
        modeldatabase="mydatabase"
        intable="mytable"
```

```

        outtable="myoutput"
    ;
run;
quit;

proc scoreaccel sessref=mysess1;
    deletemodel
        exttype=hadoop
        caslib="myhadoop"
        modelname="simple01"
        modeldatabase="mydatabase";
run;
quit;

proc cas;
    sparkEmbeddedProcess.stopSparkEP caslib="myhadoop";
run;
quit;

```

Program Description

Start a CAS session and assign a caslib to Hadoop. The `hadoopConfigDir=` option specifies the folder where the `spark-defaults.config` file is stored. The `server=` option identifies the Hive server. You must set the `jobManagementURL=` option for scoring, in order to dispatch the job by using the Apache Spark Livy interface. If Kerberos is enabled, omit the `userName=` and `password=` options. Note that this caslib does not demonstrate the options that are needed for parallel data transfer.

```

cas mysess1;
caslib myhadoop datasource=(
    srcType="hadoop",
    server="Hive-server",
    port="port-number",
    userName="user-name",
    password="password",
    schema="database-name",
    hadoopConfigDir="Hadoop_config_path_directory",
    jobManagementURL="http://server.company.com:8998"
);

```

Publish the model. The following `PUBLISHMODEL` statement does not specify the `MODELDIR=` option. Therefore, the `webHdfsUrl=` option is not required in the caslib assignment above. Without the `MODELDIR=` option, the model is published to a Hive table. This method is recommended. The optional `MODELDATABASE=` option specifies an existing database where the model is stored.

```

proc scoreaccel sessref=mysess1;
    publishmodel
        exttype=hadoop
        caslib="myhadoop"
        modelname="simple01"
        modeldatabase="mydatabase"
        programfile="/score/simple/simple.ds2";
run;
quit;

```

Start the SAS Embedded Process for Spark continuous session. Use the CAS procedure to submit the startSparkEP action. In the caslib= parameter, specify the Hadoop caslib.

```
proc cas;
  sparkEmbeddedProcess.startSparkEP caslib="myhadoop"
    executorInstances=2 executorCores=2
    executorMemory=10 driverMemory=4;
run;
quit;
```

Run the model.

```
proc scoreaccel sessref=mysess1;
  runmodel
    exttype=hadoop
    caslib="myhadoop"
    modelname="simple01"
    modeldatabase="mydatabase"
    intable="mytable"
    outtable="myoutput "
  ;
run;
quit;
```

Delete the model if needed.

```
proc scoreaccel sessref=mysess1;
  deletemodel
    exttype=hadoop
    caslib="myhadoop"
    modelname="simple01"
    modeldatabase="mydatabase";
run;
quit;
```

To stop the SAS Embedded Process for Spark continuous session, submit the stopSparkEP action. The session is also terminated when the CAS session is terminated.

```
proc cas;
  sparkEmbeddedProcess.stopSparkEP caslib="myhadoop";
run;
quit;
```

Example 4: Publish a Model to a Spark Table and Run in Databricks on AWS

Features:

- CASLIB statement
- CAS procedure
- PROC SCOREACCEL statement
- PUBLISHMODEL statement
- RUNMODEL statement

DELETEMODEL statement

Details

In this example, PROC SCOREACCEL publishes a model to a Spark table. Then PROC SCOREACCEL runs the model in Databricks on AWS by using the SAS Embedded Process for Spark. For details about the caslib, see [“Spark Data Connector” in SAS Cloud Analytic Services: User's Guide](#).

Note: Starting with 2026.06 for Databricks, PROC SCOREACCEL is deprecated. The Model Publishing and Scoring action set and the SAS Embedded Process for Spark action set are also deprecated. You are encouraged to use PROC ACCELERATOR instead. To use PROC ACCELERATOR, you must deploy the SAS Embedded Process from the accelserver RPM. See [“Deployment Details for Databricks and Synapse” on page 98](#).

Note: In 2023.10, SAS changed the redistributed JDBC drivers used by SAS/ACCESS Interface to Hadoop and SAS/ACCESS Interface to Spark. Changes might be needed in your connection options. For more information, see [“Updates Might Be Needed Due to Driver Change” in SAS In-Database Products: User's Guide](#).

Start a CAS session. Assign a caslib to Spark. Consult your Databricks workspace to find the values for the clusterId=, httpPath=, jobManagementUrl=, and server= options.

```
cas mysess;
caslib myspark datasource=(
    srcType="spark",
    platform="databricks",
    schema="default"
    clusterId="cluster-id",
    userName="token",
    authToken="authentication-token",
    password="authentication-token",
    jobManagementUrl="https://nnnnn.cloud.databricks.com/",
    server="https://nnnnn.cloud.databricks.com/",
    httpPath="my-http-path"
);
```

Publish a model to a Spark table. Do not specify the MODELDIR= option. The optional MODELDATABASE= option specifies an existing database where the model is stored.

```
proc scoreaccel sessref=mysess;
  publishmodel
    exttype=databricks
    caslib="myspark"
    modelName="mymodel"
    storefiles="/myfiles/mystore.ast"
    programfile="/myfiles/myprogram.sas"
```

```

        modeldatabase="mydatabase";
    run;
quit;

```

Start a Databricks interactive session before you run the model. If you do not use an interactive session, then the SAS Embedded Process for Spark executes in a Databricks notebook, which causes the Embedded Process to start and stop for each execution request.

```

proc cas;
    sparkEmbeddedProcess.startSparkEP
        caslib="myspark";
run;
quit;

```

Run the model. The INTABLE= option specifies the input table that contains the data to run the model against.

```

proc scoreaccel sessref=mysess;
    runmodel
        exttype=spark
        caslib="myspark"
        modelname="mymodel"
        modeldatabase="mydatabase"
        intable="mytable"
        outtable="mytable_out";
run;
quit;

```

Delete the model if needed.

```

proc scoreaccel sessref=mysess;
    deletemodel
        exttype=databricks
        caslib="myspark"
        modelname="mymodel"
        modeldatabase="mydatabase";
run;
quit;

```

To stop the SAS Embedded Process for Spark continuous session, submit the stopSparkEP action. The session is also terminated when the CAS session is terminated.

```

proc cas;
    sparkEmbeddedProcess.stopSparkEP caslib="myspark";
run;
quit;

```

Example 5: Publish a Model to a Spark Table and Run in Azure Databricks

Features: CASLIB statement
CAS procedure

PROC SCOREACCEL statement
 PUBLISHMODEL statement
 RUNMODEL statement
 DELETEMODEL

Details

In this example, PROC SCOREACCEL publishes a model to a Spark table. Then PROC SCOREACCEL runs the model in Databricks on Azure by using the SAS Embedded Process for Spark. For details about the caslib, see [“Spark Data Connector” in SAS Cloud Analytic Services: User's Guide](#).

Note: Starting with 2026.06 for Databricks, PROC SCOREACCEL is deprecated. The Model Publishing and Scoring action set and the SAS Embedded Process for Spark action set are also deprecated. You are encouraged to use PROC ACCELERATOR instead. To use PROC ACCELERATOR, you must deploy the SAS Embedded Process from the accelserver RPM. See [“Deployment Details for Databricks and Synapse” on page 98](#).

Note: In 2023.10, SAS changed the redistributed JDBC drivers used by SAS/ACCESS Interface to Hadoop and SAS/ACCESS Interface to Spark. Changes might be needed in your connection options. For more information, see [“Updates Might Be Needed Due to Driver Change” in SAS In-Database Products: User's Guide](#).

Start a CAS session. Assign a caslib to Spark. Consult your Azure Databricks workspace to find the values for the clusterId=, httpPath=, jobManagementUrl=, and server= options.

```
cas mysess;
caslib myspark datasource=(
  srcType="spark",
  platform="databricks",
  server="myserver.azuredatabricks.net",
  userName="token",
  password="authentication-token",
  clusterId="cluster-id",
  jobManagementUrl="https://nnnnn.cloud.databricks.net/",
  httpPath="my-http-path",
  schema="my-schema"
);
```

Publish a model to a Spark table. Do not specify the MODELDIR= option. The optional MODELDATABASE= option specifies an existing database where the model is stored.

```
proc scoreaccel sessref=mysess;
  publishmodel
    exttype=databricks
    caslib="myspark"
```

```

        modelname="mymodel"
        storefiles="/myfiles/mystore.ast"
        programfile="/myfiles/myprogram.sas"
        modeldatabase="mydatabase";
    run;
quit;

```

Start a Databricks interactive session before you run the model. If you do not use an interactive session, then the SAS Embedded Process for Spark executes in a Databricks notebook, which causes the Embedded Process to start and stop for each execution request.

```

proc cas;
    sparkEmbeddedProcess.startSparkEP
        caslib="myspark";
run;
quit;

```

Run the model. The INTABLE= option specifies the input table that contains the data to run the model against.

```

proc scoreaccel sessref=mysess;
    runmodel
        exttype=spark
        caslib="myspark"
        modelname="mymodel"
        modeldatabase="mydatabase"
        intable="mytable"
        outtable="mytable_out";
run;
quit;

```

Delete the model if needed.

```

proc scoreaccel sessref=mysess;
    deletemodel
        exttype=databricks
        caslib="myspark"
        modelname="mymodel"
        modeldatabase="mydatabase";
run;
quit;

```

To stop the SAS Embedded Process for Spark continuous session, submit the stopSparkEP action. The session is also terminated when the CAS session is terminated.

```

proc cas;
    sparkEmbeddedProcess.stopSparkEP caslib="myspark";
run;
quit;

```

Example 6: Publish a Model to Amazon S3 and Run in Databricks on AWS

Features:

- CASLIB statement
- CAS procedure
- PROC SCOREACCEL statement
- PUBLISHMODEL statement
- RUNMODEL statement
- DELETEMODEL statement

Details

In this example, PROC SCOREACCEL publishes a model to Amazon S3. Then PROC SCOREACCEL runs the model in Databricks on Amazon Web Services (AWS) by using the SAS Embedded Process for Spark. For details about Amazon S3, see [“Amazon S3 Data Source” in SAS Cloud Analytic Services: User’s Guide](#).

Note: Starting with 2026.06 for Databricks, PROC SCOREACCEL is deprecated. The Model Publishing and Scoring action set and the SAS Embedded Process for Spark action set are also deprecated. You are encouraged to use PROC ACCELERATOR instead. To use PROC ACCELERATOR, you must deploy the SAS Embedded Process from the accelserver RPM. See [“Deployment Details for Databricks and Synapse” on page 98](#).

Start a CAS session. Assign a caslib to Amazon S3 for publishing.

```
cas casauto;  
caslib mys3 datasource=(  
    srctype="s3",  
    bucket="bucket-name",  
    accesskeyid="access-key-id",  
    secretaccesskey="secret-access-key",  
    region="region-value"  
);
```

Publish a model to Amazon S3.

```
proc scoreaccel sessref=casauto;  
  publishmodel  
    exttype=filesystem  
    caslib="mys3"  
    modelname="mymodel"  
    storefiles="/myfiles/mystore.ast"  
    programfile="/myfiles/myprogram.sas"
```



```

        modeldir="/scoring/models";
    run;
quit;

```

To run a model in Databricks on AWS, a Spark caslib is required.

```

caslib myspark datasource=(
    srctype="spark",
    platform="databricks",
    schema="default",
    clusterid="cluster-id",
    username="token",
    authtoken="authentication-token",
    password="authentication-token",
    jobmanagementurl="https://nnnnn.cloud.databricks.com/",
    hadoopjarpath="path-to-hadoop-jar-files"
);

```

Start a Databricks interactive session. If you do not use an interactive session, then the SAS Embedded Process for Spark executes in a Databricks notebook, which causes the Embedded Process to start and stop for each execution request.

```

proc cas;
    sparkEmbeddedProcess.startSparkEP
        caslib="myspark";
run;
quit;

```

Run the model in Databricks on AWS. The INTABLE= option specifies the input table that contains the data to run the model against.

```

proc scoreaccel sessref=casauto;
    runmodel
        exttype=spark
        caslib="myspark"
        modelname="mymodel"
        modeldir="dbfs:/mnt/s3-bucket-mount-point/scoring/models"
        intable="mytable"
        outtable="mytable_out";
run;
quit;

```

Delete the model if needed.

```

proc scoreaccel sessref=casauto;
    deletemodel
        exttype=filesystem
        caslib="mys3"
        modelname="mymodel"
        modeldir="/scoring/models";
run;
quit;

```

To stop the SAS Embedded Process for Spark continuous session, submit the stopSparkEP action. The session is also terminated when the CAS session is terminated.

```

proc cas;

```

```
sparkEmbeddedProcess.stopSparkEP caslib="myspark";
run;
quit;
```

Example 7: Publish a Model to ADLS and Run in Azure Databricks

Features:

- CASLIB statement
- CAS procedure
- PROC SCOREACCEL statement
- PUBLISHMODEL statement
- RUNMODEL statement
- DELETEMODEL statement

Details

In this example, PROC SCOREACCEL publishes a model to Microsoft Azure Data Lake Storage Gen2 (ADLS). Then PROC SCOREACCEL runs the model in Azure Databricks by using the SAS Embedded Process for Spark.

Note: Starting with 2026.06 for Databricks, PROC SCOREACCEL is deprecated. The Model Publishing and Scoring action set and the SAS Embedded Process for Spark action set are also deprecated. You are encouraged to use PROC ACCELERATOR instead. To use PROC ACCELERATOR, you must deploy the SAS Embedded Process from the accelserver RPM. See [“Deployment Details for Databricks and Synapse” on page 98](#).

Prepare to publish the model to Microsoft Azure Data Lake Storage Gen2 (ADLS). Start a CAS session. You must include the `azuretenantid=` session option.

```
cas casauto sessopts=(azuretenantid="tenant-ID");
```

Assign a caslib to ADLS for publishing.

```
caslib myadls datasource=(
    srctype="adls"
    accountname="account-name",
    applicationid="application-id",
    filesystem="mystorage",
    dnssuffix="dfs.core.windows.net"
);
```

Publish the model to ADLS by specifying EXTTYPE=FILESYSTEM in the PUBLISHMODEL statement. In the CASLIB= argument, specify the ADLS caslib.

For the PASSWORD= argument, use the password from RBAC, which is also known as the secret.

```
proc scoreaccel sessref=casauto;
  publishmodel
    exttype=filesystem
    caslib="myadls"
    password="RBAC-client-secret"
    modelname="mymodel"
    modeldir="/scoring/models"
    storefiles="/myfiles/mystore.ast"
    programfile="/myfiles/myprogram.sas";
run;
quit;
```

To run a model in Azure Databricks, a Spark caslib is required.

```
caslib myspark datasource=(
  srctype='spark',
  platform=databricks,
  schema="default",
  clusterid="cluster-id",
  username="token",
  authtoken="authentication-token",
  password="authentication-token",
  jobmanagementurl="https://nnnnn.cloud.databricks.com/",
  hadoopjarpath="path-to-hadoop-jar-files"
);
```

Start a Databricks interactive session. If you do not use an interactive session, then the SAS Embedded Process for Spark executes in a Databricks notebook, which causes the Embedded Process to start and stop for each execution request. Most customers choose to use an interactive session for better performance.

```
proc cas;
  sparkEmbeddedProcess.startSparkEP
  caslib="myspark";
run;
quit;
```

Run the model in Azure Databricks. The INTABLE= option specifies the input table that contains the data to run the model against.

```
proc scoreaccel sessref=casauto;
  runmodel
    exttype=spark
    caslib="myspark"
    modelname="mymodel"
    modeldir="dbfs:/mnt/blob-storage-mount-point/scoring/models"
    intable="mytable"
    outtable="mytable_out";
run;
quit;
```

Delete the model if needed.

```
proc scoreaccel sessref=casauto;
  deletemodel
    exttype=filesystem
```

```

caslib="myadls"
password="RBAC-client-secret"
modelname="mymodel"
modeldir="/scoring/models";

run;
quit;

```

To stop the SAS Embedded Process for Spark continuous session, submit the stopSparkEP action. The session is also terminated when the CAS session is terminated.

```

proc cas;
  sparkEmbeddedProcess.stopSparkEP caslib="myspark";
run;
quit;

```

Example 8: Publish a Model to an SQL Server Table and Run in Azure Synapse

Features:

- CASLIB statement
- CAS procedure
- PROC SCOREACCEL statement
- PUBLISHMODEL statement
- RUNMODEL statement
- DELETEMODEL

Details

Note: Starting with 2026.04 for Azure Synapse, PROC SCOREACCEL is deprecated. The Model Publishing and Scoring action set and the SAS Embedded Process for Spark action set are also deprecated. The change is a result of a Microsoft announcement to end support for Azure Synapse Runtime for Apache Spark 3.4. You can no longer deploy the sepcorespark RPM in Synapse. You are encouraged to use PROC ACCELERATOR instead. To use PROC ACCELERATOR, you must deploy the SAS Embedded Process from the accelserver RPM. See [“Deployment Details for Databricks and Synapse” on page 98](#).

In this example, PROC SCOREACCEL publishes a model to an SQL Server table. Then PROC SCOREACCEL runs the model in Synapse by using the SAS Embedded Process for Spark.

As a requirement before you can publish a model to an SQL Server table, you must download the Microsoft JDBC Driver for SQL Server. For instructions, see [“Configuration for In-Database Model Scoring” in SAS In-Database Products: Deployment and Administration Guide](#).

Start a CAS session. Assign a JDBC caslib in order to publish the model. For Synapse, you must identify both a schema and a database in the caslib in the following ways: Specify the schema value in the SCHEMA= option of the caslib. For the database value, the JDBC caslib does not support the DATABASE= option, so you must use the URI= option to specify a connection string for the Microsoft JDBC Driver for SQL Server. (The connection string includes the database value, but not the schema value.) As noted above, a prerequisite is to download the Microsoft JDBC Driver for SQL Server driver. For instructions, see [“Configuration for In-Database Model Scoring” in SAS In-Database Products: Deployment and Administration Guide](#). Then, in the Synapse workspace, in the JDBC connection strings, copy the connection string that is provided for JDBC (SQL authentication). Specify that connection string in the URI= option of the caslib. In addition, in the connection string, you must add the property useBulkCopyForBatchInsert=true.

```
cas mysess;
caslib myjdbc datasource=(
    srcType="jdbc",
    userName="user-name",
    password="password",
    uri="JDBC-connection-options;
        useBulkCopyForBatchInsert=true;BatchSize=400",
    schema="SQL-server-schema-name"
);
```

Publish a model to an SQL Server table. To store the model in an SQL Server table, do not specify the MODELDIR= option. For Synapse, when you delete, publish, or run a model that is stored in an SQL Server table, the MODELSchema= option is required. Specify EXTTYPE=SYNAPSE and reference your JDBC caslib.

```
proc scoreaccel sessref=mysess;
    publishmodel
        exttype=synapse
        caslib="myjdbc"
        modelname="mymodel"
        modelschema="myschema"
        storefiles="/myfiles/mystore.ast"
        programfile="/myfiles/myprogram.sas"
    ;
run;
quit;
```

Assign a Spark caslib in order to start an interactive session and run the model. In the caslib, specify platform="synapse". For the jobManagementUrl= parameter, specify the Livy REST URL that is specific to the Spark pool that you want to connect to. For more information about the jobManagementUrl= value, see [“Forming the Livy REST URL for Synapse” in SAS In-Database Products: User’s Guide](#).

```
cas mysess;
caslib myspark datasource=(
    srcType="spark",
    platform="synapse",
    userName="application-id",
    password="secret",
    server="myserver.xxxx.net",
    database="my-database-name",
    jobManagementUrl="Livy-rest-url"
```

```
);
```

Start an interactive session before you run the model. If you do not use an interactive session, then the SAS Embedded Process for Spark starts automatically, using the default resource settings that are configured for the cluster.

```
proc cas;
  sparkEmbeddedProcess.startSparkEP
    caslib="myspark";
run;
quit;
```

Run the model. For the INTABLE= and OUTTABLE= values, use a two-part name. Specify the SQL server schema name followed by a dot and then the table name.

```
proc scoreaccel sessref=mysess;
  runmodel
    exttype=spark
    caslib="myspark"
    modelname="mymodel"
    modelschema="myschema"
    intable="myschema.mytable"
    outtable="myschema.mytable_out";
run;
quit;
```

Delete the model if needed.

```
proc scoreaccel sessref=mysess;
  deletemodel
    exttype=synapse
    caslib="myjdbc"
    modelname="mymodel"
    modelschema="myschema";
run;
quit;
```

To stop the SAS Embedded Process for Spark continuous session, submit the stopSparkEP action. The session is also terminated when the CAS session is terminated.

```
proc cas;
  sparkEmbeddedProcess.stopSparkEP caslib="myspark";
run;
quit;
```

Example 9: Publish a Model to ADLS and Run in Azure Synapse

Features:

- CASLIB statement
- CAS procedure
- PROC SCOREACCEL statement
- PUBLISHMODEL statement

RUNMODEL statement
DELETEMODEL statement

Details

Note: Starting with 2026.04 for Azure Synapse, PROC SCOREACCEL is deprecated. The Model Publishing and Scoring action set and the SAS Embedded Process for Spark action set are also deprecated. The change is a result of a Microsoft announcement to end support for Azure Synapse Runtime for Apache Spark 3.4. You can no longer deploy the sepcorespark RPM in Synapse. You are encouraged to use PROC ACCELERATOR instead. To use PROC ACCELERATOR, you must deploy the SAS Embedded Process from the accelserver RPM. See [“Deployment Details for Databricks and Synapse” on page 98](#).

In this example, PROC SCOREACCEL publishes a model to Microsoft Azure Data Lake Storage Gen2 (ADLS). Then PROC SCOREACCEL runs the model in Azure Synapse by using the SAS Embedded Process for Spark. The model runs against an SQL Server table.

For Synapse, PROC SCOREACCEL uses Azure Active Directory (Azure AD) application registration and role-based access control (RBAC). For essential information, such as how to specify the Synapse Spark pool Livy REST URL, and the requirements for SQL Server, see [“In-Database Model Scoring” in SAS In-Database Products: User’s Guide](#).

Program

```
cas mysess sessopts=(azuretenantid="tenant-id");

caslib myadls datasource=(
    srctype="adls"
    accountname="myaccount",
    applicationid="application-id",
    filesystem="mystorage",
    dnssuffix="dfs.core.windows.net"
);

proc scoreaccel sessref=mysess;
  publishmodel
    exttype=filesystem
    caslib="myadls"
    password="RBAC-client-secret"
    modelname="mymodel"
    modeldir="/scoring/models"
    programfile="/myfiles/myprogram.ds2";
run;
quit;

caslib myspark datasource=(
```

```

        srctype="spark",
        platform="synapse",
        username="application-id",
        password="RBAC-client-secret",
        schema="SQL-server-database-name",
        hadoopjarpath="path-to-hadoop-jar-files",
        jobmanagementurl="Livy-rest-url"
    );

proc cas;
    sparkEmbeddedProcess.startSparkEP caslib="myspark"
        executorInstances=2 executorCores=2
        executorMemory=10 driverMemory=4;
run;
quit;

proc scoreaccel sessref=mysess;
    runmodel
        exttype=spark
        caslib="myspark"
        modelname="mymodel"
        modeldir="/scoring/models"
        intable="SQL-server-database-schema-name.in-table-name"
        outtable="SQL-server-database-schema-name.out-table-name";
run;
quit;

proc scoreaccel sessref=mysess;
    deletemodel
        exttype=filesystem
        caslib="myadls"
        password="RBAC-client-secret"
        modelname="mymodel"
        modeldir="/scoring/models";
run;
quit;

proc cas;
    sparkEmbeddedProcess.stopSparkEP caslib="myspark";
run;
quit;

```

Program Description

Prepare to publish the model to Microsoft Azure Data Lake Storage Gen2 (ADLS).

Start a CAS session. You must include the `azuretenantid=` session option.

```
cas mysess sessopts=(azuretenantid="tenant-id");
```

Assign a caslib to ADLS.

```
caslib myadls datasource=(
    srctype="adls"
    accountname="myaccount",
    applicationid="application-id",
    filesystem="mystorage",
    dnssuffix="dfs.core.windows.net"
);
```



```
);
```

Publish the model to ADLS by specifying EXTTYPE=FILESYSTEM in the PUBLISHMODEL statement. In the CASLIB= argument, specify the ADLS caslib. For the PASSWORD= argument, use the password from RBAC, which is also known as the secret.

```
proc scoreaccel sessref=mysess;
  publishmodel
    exttype=filesystem
    caslib="myadls"
    password="RBAC-client-secret"
    modelname="mymodel"
    modeldir="/scoring/models"
    programfile="/myfiles/myprogram.ds2";
run;
quit;
```

To run a model in Synapse, a Spark caslib is required. The username= is the application ID. For the password= parameter, use the password from RBAC, which is also known as the secret. For SQL Server tables, the schema name is the database name. For the jobmanagementurl= parameter, specify the Synapse Spark pool Livy REST URL. For more information about the jobManagementUrl= value, see [“Forming the Livy REST URL for Synapse” in SAS In-Database Products: User’s Guide](#). The Hadoop JAR path is required. The Hadoop configuration path is not required.

```
caslib myspark datasource=(
  srctype="spark",
  platform="synapse",
  username="application-id",
  password="RBAC-client-secret",
  schema="SQL-server-database-name",
  hadoopjarpath="path-to-hadoop-jar-files",
  jobmanagementurl="Livy-rest-url"
);
```

Start the SAS Embedded Process for Spark continuous session. Use the CAS procedure to submit the startSparkEP action. In the caslib= parameter, specify the Spark caslib.

```
proc cas;
  sparkEmbeddedProcess.startSparkEP caslib="myspark"
    executorInstances=2 executorCores=2
    executorMemory=10 driverMemory=4;
run;
quit;
```

Run the model in Synapse by specifying EXTTYPE=SPARK in the RUNMODEL statement. In the CASLIB= argument, specify the Spark caslib.

```
proc scoreaccel sessref=mysess;
  runmodel
    exttype=spark
    caslib="myspark"
    modelname="mymodel"
    modeldir="/scoring/models"
    intable="SQL-server-database-schema-name.in-table-name"
    outtable="SQL-server-database-schema-name.out-table-name";
```

```
run;
quit;
```

Delete the model if needed.

```
proc scoreaccel sessref=mysess;
  deletemodel
    exttype=filesystem
    caslib="myadls"
    password="RBAC-client-secret"
    modelname="mymodel"
    modeldir="/scoring/models";
run;
quit;
```

To stop the SAS Embedded Process for Spark continuous session, submit the stopSparkEP action. The session is also terminated when the CAS session is terminated.

```
proc cas;
  sparkEmbeddedProcess.stopSparkEP caslib="myspark";
run;
quit;
```

Example 10: Publish, Run, and Delete a Model in SingleStore

Features:	CASLIB statement
	PROC SCOREACCEL statement
	PUBLISHMODEL statement
	RUNMODEL statement
	DELETEMODEL statement

Details

In this example, PROC SCOREACCEL publishes and runs a model in SingleStore. In-database model scoring is supported only if you license and deploy the product SAS SpeedyStore. For more information, see [“SAS SpeedyStore Data Connector” in SAS Cloud Analytic Services: User’s Guide](#). In-database model scoring is not supported by the SingleStore standard data connector.

For information about deployment, see [SAS SpeedyStore: Administration and Configuration Guide](#).

Program

```

cas casauto;

caslib mycaslib datasource=(
    srcType="singlestore"
    database="my_database"
    host="svc-sas-singlestore-cluster-dml"
    port="3306"
    password="my_password"
    userName="my_username"
);

proc scoreaccel;
    publishmodel
        exttype=singlestore
        caslib="mycaslib"
        modelName="mymodel"
        programfile="/myfiles/myprogram.ds2"
;
run;
quit;

proc scoreaccel;
    runmodel
        exttype=singlestore
        caslib="mycaslib"
        modelName="mymodel"
        intable="in-table-name"
        outtable="out-table-name";
;
run;
quit;

proc scoreaccel;
    deletemodel
        exttype=singlestore
        caslib="mycaslib"
        modelName="mymodel"
;
run;
quit;

```

Program Description

Prepare to publish the model to SingleStore. Start a CAS session if one is not already started.

```
cas casauto;
```

Assign a caslib to SingleStore. If single sign-on is enabled, do not include the password= and userName= options or the authenticationDomain= option.

```
caslib mycaslib datasource=(
    srcType="singlestore"
    database="my_database"
```

```

host="svc-sas-singlestore-cluster-dml"
port="3306"
password="my_password"
userName="my_username"
);

```

Publish the model to SingleStore by specifying EXTTYPE=SINGLESTORE in the PUBLISHMODEL statement. In the CASLIB= argument, specify the SingleStore caslib.

```

proc scoreaccel;
  publishmodel
    exttype=singlestore
    caslib="mycaslib"
    modelname="mymodel"
    programfile="/myfiles/myprogram.ds2"
  ;
run;
quit;

```

Run the model by specifying EXTTYPE=SINGLESTORE in the PUBLISHMODEL statement. In the CASLIB= argument, specify the SingleStore caslib.

```

proc scoreaccel;
  runmodel
    exttype=singlestore
    caslib="mycaslib"
    modelname="mymodel"
    intable="in-table-name"
    outtable="out-table-name";
  ;
run;
quit;

```

Delete the model if needed.

```

proc scoreaccel;
  deletemodel
    exttype=singlestore
    caslib="mycaslib"
    modelname="mymodel"
  ;
run;
quit;

```

SORT Procedure

Overview: SORT Procedure	2039
What Does the SORT Procedure Do?	2040
Sorting SAS Data Sets	2040
Concepts: SORT Procedure	2042
Threaded Sorting	2042
Sorting Orders for Numeric Variables	2043
Sorting Orders for Character Variables	2043
Stored Sort Information	2045
Presorted Input Data Sets	2046
Linguistic Sorting of Data Sets and ICU	2046
Force SAS to Sort Data Sets Using System Option SORTPGM	2048
Syntax: SORT Procedure	2049
PROC SORT Statement	2050
ATTRIB Statement	2068
BY Statement	2069
FORMAT Statement	2070
KEY Statement	2071
LABEL Statement	2072
WHERE Statement	2074
Usage: SORT Procedure	2076
In-Database Processing: PROC SORT	2076
SAS Cloud Analytic Services Processing for PROC SORT	2078
Integrity Constraints: SORT Procedure	2079
Results: SORT Procedure	2079
Procedure Output	2079
Output Data Set	2079
Examples: SORT Procedure	2080
Example 1: Sorting by the Values of Multiple Variables	2080
Example 2: Sorting in Descending Order	2083
Example 3: Maintaining the Relative Order of Observations in Each BY Group	2085
Example 4: Retaining the First Observation of Each BY Group	2088
Example 5: Linguistic Sorting Using ALTERNATE_HANDLING=	2090
Example 6: Linguistic Sorting Using ALTERNATE_HANDLING= and STRENGTH=	2093
Example 7: Eliminate All Duplicate Observations Using NODUPKEY	2096

Overview: SORT Procedure

What Does the SORT Procedure Do?

The SORT procedure orders SAS data set observations by the values of one or more character or numeric variables. The SORT procedure either replaces the original data set or creates a new data set. PROC SORT produces only an output data set. For more information, see [“Procedure Output” on page 2079](#).

Note: If extended attributes are defined on the input data set, PROC SORT propagates the extended attributes to the output data set. For information about extended attributes, see [“Extended Attributes” on page 371](#).

Operating Environment Information: The sorting capabilities that are described in this chapter are available for all operating environments. In addition, if you use the HOST value of the SAS system option SORTPGM=, you might be able to use other sorting options that are available for your operating environment. For more information about other sorting capabilities, see the SAS documentation for your operating environment.

Sorting SAS Data Sets

In the following example, the original data set was in alphabetical order by last name. PROC SORT replaces the original data set with a data set that is sorted by employee identification number. The following log shows the results from running this PROC SORT step. [Output 55.286 on page 2041](#) shows the results of the PROC PRINT step. The statements that produce the output follow:

```
proc sort data=employee;
    by idnumber;
run;

proc print data=employee;
run;
```

Example Code 55.1 SAS Log Generated by PROC SORT

NOTE: There were six observations read from the data set WORK.EMPLOYEE.
 NOTE: The data set WORK.EMPLOYEE has six observations and three variables.
 NOTE: PROCEDURE SORT used:
 real time 0.01 seconds
 cpu time 0.01 seconds

Output 55.1 Observations Sorted by the Values of One Variable

The SAS System			1
Obs	Name	IDnumber	
1	Belloit	1988	
2	Wesley	2092	
3	Lemeux	4210	
4	Arnsbarger	5466	
5	Pierce	5779	
6	Capshaw	7338	

The following output shows the results of a more complicated sort by three variables. The businesses in this example are sorted by town, then by debt from highest amount to lowest amount, then by account number. For an explanation of the program that produces this output, see [“Example 2: Sorting in Descending Order”](#) on page 2083.

Output 55.2 Observations Sorted by the Values of Three Variables

Customers with Past-Due Accounts Listed by Town, Amount, Account Number					1
Obs	Company	Town	Debt	Account Number	
1	Paul's Pizza	Apex	83.00	1019	
2	Peter's Auto Parts	Apex	65.79	7288	
3	Watson Tabor Travel	Apex	37.95	3131	
4	Tina's Pet Shop	Apex	37.95	5108	
5	Apex Catering	Apex	37.95	9923	
6	Deluxe Hardware	Garner	467.12	8941	
7	Boyd & Sons Accounting	Garner	312.49	4762	
8	World Wide Electronics	Garner	119.95	1122	
9	Elway Piano and Organ	Garner	65.79	5217	
10	Ice Cream Delight	Holly Springs	299.98	2310	
11	Tim's Burger Stand	Holly Springs	119.95	6335	
12	Strickland Industries	Morrisville	657.22	1675	
13	Pauline's Antiques	Morrisville	302.05	9112	
14	Bob's Beds	Morrisville	119.95	4998	

Concepts: SORT Procedure

Threaded Sorting

The THREADS system option enables threaded sorting. Threaded sorting achieves a degree of parallelism in the sorting operations. This parallelism is intended to reduce the real time to completion for a given operation and therefore limit the cost of additional CPU resources. For more information, see [“THREADS System Option” in SAS System Options: Reference](#).

The multi-threaded SAS sort can also be invoked when you specify the THREADS option in the PROC SORT statement. The multi-threaded sort stores all temporary data in a single utility file within one of the locations that are specified by the UTILLOC= system option. The size of this utility file is proportional to the amount of data that is read from the input data set. A second utility file of the same size can be created in another of these locations when the amount of data that is read from the input data set is large or the amount of memory that is available to the SORT procedure is small. For more information, refer to [“UTILLOC= System Option” in SAS System Options: Reference](#).

Note: The TAGSORT option on page 2066 does not support threaded sorting.

The multi-threaded SAS sort can be invoked when the THREADS system option is specified and the value of the CPUCOUNT= system option is greater than 1. The value of the SAS system option CPUCOUNT= affects the performance of the threaded sort. CPUCOUNT= suggests how many system CPUs are available for use by the threaded procedures.

Calculated statistics can vary slightly, depending on the order in which observations are processed. Such variations are due to numerical errors that are introduced by floating-point arithmetic, the results of which should be considered approximate and not exact. The order of observation processing can be affected by nondeterministic effects of multithreaded or parallel processing. The order of processing can also be affected by inconsistent or nondeterministic ordering of observations that are produced by a data source, such as a DBMS that delivers query results through an ACCESS engine. For more information, see [“Threading in Base SAS” in SAS Programmer’s Guide: Essentials](#)

For more information, see the [“THREADS System Option” in SAS System Options: Reference](#) and the [“CPUCOUNT= System Option” in SAS System Options: Reference](#).

Sorting Orders for Numeric Variables

For numeric variables, the following is the smallest-to-largest comparison sequence:

- 1 SAS missing values (shown as a period or special missing value)
- 2 negative numeric values
- 3 zero
- 4 positive numeric values

Sorting Orders for Character Variables

Default Collating Sequence

The order in which alphanumeric characters are sorted is known as the collating sequence. This sort order is determined by the session encoding.

By default PROC SORT uses the ASCII collating sequence when it compares character values in the SAS Viya platform.

For more information about the various collating sequences and when they are used, see [“Collating Sequence” in SAS National Language Support \(NLS\): Reference Guide](#).

Note: ASCII and EBCDIC represent the family names of the session encodings. The sort order can be determined by referring to the encoding.

ASCII Order

The Linux operating environment uses the ASCII collating sequence.

From the smallest to the largest character that you can display, the English-language ASCII sequence is consistent with the order shown in the following table.

Table 55.1 ASCII Sort Order Example

blank ! " # \$ % & ' () * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @
--

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [ \ ] ° _
```

```
a b c d e f g h i j k l m n o p q r s t u v w x y z { } ~
```

The main features of the ASCII sequence are that digits are sorted before uppercase letters, and uppercase letters are sorted before lowercase letters. The blank is the smallest character that you can display.

EBCDIC Order

The sorting order of the English-language EBCDIC sequence is consistent with the following sort order example.

Table 55.2 EBCDIC Sort Order Example

```
blank . < ( + | & ! $ * ) ; ¬ - / , % _ > ? : # @ ' = "
```

```
a b c d e f g h i j k l m n o p q r ~ s t u v w x y z
```

```
{ A B C D E F G H I } J K L M N O P Q R S T
```

```
U V W X Y Z
```

```
0 1 2 3 4 5 6 7 8 9
```

The main features of the EBCDIC sequence are that lowercase letters are sorted before uppercase letters, and uppercase letters are sorted before digits. Note also that some special characters interrupt the alphabetic sequences. The blank is the smallest character that you can display.

Specifying Sorting Orders for Character Variables

The options ASCII, EBCDIC, DANISH, NATIONAL, SWEDISH, and REVERSE specify collating sequences that are stored in the HOST catalog.

If you want to provide your own collating sequences or change a collating sequence provided for you, then use the TRANTAB procedure to create or modify translation tables. When you create your own translation tables, they are stored in your PROFILE catalog, and they override any translation tables that have the same name in the HOST catalog. For complete details, see [“TRANTAB Procedure” in SAS National Language Support \(NLS\): Reference Guide](#).

Note: System managers can modify the HOST catalog by copying newly created tables from the PROFILE catalog to the HOST catalog. Then all users can access the new or modified translation table.

Linguistic Collation sorts data according to rules of language. For detailed information about Linguistic Collation, see [“Collating Sequence” in SAS National Language Support \(NLS\): Reference Guide](#).

Stored Sort Information

PROC SORT records the BY variables, collating sequence, and character set that it uses to sort the data set. This information is stored with the data set to help avoid unnecessary sorts.

Before PROC SORT sorts a data set, it checks the stored sort information. If you try to sort a data set how it is currently sorted, then PROC SORT does not perform the sort and writes a message to the log to that effect. To override this behavior, use the FORCE option. If you try to sort a data set the same way it is currently sorted and you specify an OUT= data set, then PROC SORT simply makes a copy of the DATA= data set.

To override the sort information that PROC SORT stores, use the _NULL_ value with the SORTEDBY= data set option. Refer to the [“SORTEDBY= Data Set Option” in SAS Data Set Options: Reference](#).

If you want to change the sort information for an existing data set, then use the SORTEDBY= data set option in the MODIFY statement in the DATASETS procedure. For more information, see [“MODIFY Statement” on page 446](#).

To access the sort information that is stored with a data set, use the CONTENTS statement in PROC DATASETS. For more information, see [“CONTENTS Statement” on page 405](#).

The number of variables by which you can sort a data set with PROC SORT is limited only by available memory. The number of columns by which you can order the rows of a result set using PROC SQL, is also limited only by available memory. The sort indicator, whether stored in the metadata of a Base data set or represented in memory, is limited to 127 variables. For this reason, up to 127 variables can be stored in the sort indicator or listed on the SORTEDBY= data set option. If you are sorting by more than 127 variables, then only the first 127 are recorded in the sort indicator. If you sort the data set again by the entire list of BY variables, the data set is not recognized as being sorted, because the additional variables (beyond 127) are not found within the sort indicator. For a detailed explanation, see [“Sorted Data Sets” in SAS Programmer's Guide: Essentials](#).

Presorted Input Data Sets

Specifying the “[PRESORTED](#)” on [page 2065](#) option prevents SAS from sorting an already sorted data set. Before sorting, SAS checks the sequence of observations within the input data set to determine whether the observations are in order. Use the PRESORTED option when you know or strongly suspect that a data set is already in order according to the key variables specified in the BY statement. The sequence of observations within the data set is checked by reading the data set and comparing the BY variables of each observation read to the BY variables of the preceding observation. This process continues until either the entire data set has been read or an out-of-sequence observation is detected.

If the entire data set has been read and no out-of-sequence observations have been found, then one of two actions is taken. If no output data set has been specified, the sort order metadata of the input data set is updated to indicate that the sequence has been verified. This verification notes that the data set is validly sorted according to the specified BY variables. Otherwise, the data set is considered sorted and either the input data set metadata is updated or, if OUT= has been specified, the data is copied to an output data set.

If observations within the data set are not in sequence, then the data set is sorted.

If the “[NODUPKEY](#)” on [page 2061](#) option has been specified, then the sequence checking determines whether observations with duplicate keys are present in the data set. If observations with duplicate keys are found, then the data set is considered unsorted and a sort is performed. Otherwise, the data set is considered sorted and actions are taken where either the metadata of the input data set is updated or, if OUT= has been specified, data is copied to the output data set. The actions taken are described in more detail in the previous paragraphs.

If the metadata of the input data set indicates that the data is already sorted according to the key variables listed in the BY statement and the input data set has been validated, then neither sequence checking nor sorting is performed.

See “[Sorted Data Sets](#)” in *SAS Programmer's Guide: Essentials* and interactions with the “[SORTVALIDATE System Option](#)” in *SAS System Options: Reference*.

Linguistic Sorting of Data Sets and ICU

Linguistic collation sorts characters in a culturally sensitive manner according to rules that are associated with a language and locale. The rules and default collating sequence are based on the language specified in the current locale setting. The implementation is provided by the International Components for Unicode (ICU) library. It produces results that are largely compatible with the Unicode Collation Algorithms (UCA).

SAS provides ICU collation when the linguistic option (SORTSEQ=LINGUISTIC) is specified on the Base SAS procedure, PROC SORT. You can specify linguistic

collation using the SORTSEQ= option in the SQL procedure and by specifying the SORTSEQ=LINGUISTIC system option.

Note: Only PROC SORT and PROC SQL are affected when the SORTSEQ=LINGUISTIC system option is specified.

When the SORTSEQ=LINGUISTIC option is specified, SAS relies on the ICU libraries as the reference implementation of the Unicode Collation Algorithm (UCA) and as a de facto standard.

In the SAS Viya platform, the ICU library version incorporated by SAS and used by PROC SORT is ICU 63. For in-depth information, see [Download ICU 63](#).

Note: ICU libraries have been upgraded to 70.1. With the upgrade, you might see incompatibilities in the linguistic collation for the Chinese locales such as zh_CN and zh_TW.

Existing data sets that were sorted by the linguistic collator for the Chinese locale must be re-sorted.

A change in the version of the ICU that is used by PROC SORT for linguistic collation, can affect the interpretation of data sets sorted by another version of SAS. If a data set is linguistically sorted by one or more character variables in one version of SAS, the data set is recognized as being sorted when accessed in another version of SAS if the two SAS versions use different versions of the ICU. Because collation rules can change between ICU versions, variations in the rules can cause the order of observations produced by PROC SORT to be different. If the ordering differences are ignored, unexpected results can be seen during processing.

When sorting linguistically, the ICU version used by SAS is recorded in the sort indicator that is stored in the data set header. The ICU version is examined when determining if a data set is considered sorted. A difference between the ICU version in use and the ICU version recorded in the sort indicator of a data set causes SAS to ignore the indicated sort order and assume that the data set is unsorted.

Note: The PROC CONTENTS output shows the ICU version in use. See [“Example 5: Linguistic Sorting Using ALTERNATE_HANDLING=”](#) on page 2090.

If a sort indicator on a permanent data set is ignored, to facilitate processing, it can be desirable to reassert the order and reestablish the sort indicator on the data set. This can be done using PROC SORT with the PRESORTED option. Most often, because the order of observations within the data set has not been disturbed and is likely correct, the SORT procedure probably only needs to sequentially read the data set to reestablish the indicator instead of performing a complete sort. If the order of observations is not correct, then the SORT procedure reorders the observations as necessary.

For both the COPY and MIGRATE procedures, if the ICU version recorded on an input data set is different from the version in use by SAS, then the sort indicator on the input data set is ignored, the output data set is not marked as sorted, and a message is written to the SAS log. However, both procedures write observations to an output data set in the same order as they are read from the input. This order is

preserved if a physical order is supported by the engine used for the OUT= destination library. For these reasons, when migrating to a new release of SAS, consider re-establishing the sort order of permanent data sets using PROC SORT with the PRESORTED option.

Additional information about how linguistic collation is used by SAS can be found in the following documents, as well as in the PROC SORT SORTSEQ=LINGUISTIC system option.

- See [“SORTSEQ=sort-table | LINGUISTIC” in SAS SQL Procedure User's Guide](#).
- See PROC SORT option [“LINGUISTIC<\(collating-options\)>” on page 2054](#).
- See [“Specifying Linguistic Collation” in SAS National Language Support \(NLS\): Reference Guide](#).
- See [Chapter 12, “CONTENTS Procedure,” on page 289](#).
- See [Chapter 13, “COPY Procedure,” on page 309](#).
- See [Chapter 36, “MIGRATE Procedure,” on page 1343](#).
- See the [Appendix 3, “UNICODE License,” on page 2549](#).

The following are SAS papers that provide detailed information about Linguistic Collation.

- [Creating Order out of Character Chaos: Collation Capabilities of the SAS System](#)
- [Linguistic Collation: Everyone Can Get What They Expect](#)
- [Processing Multilingual Data with the SAS 9.2 Unicode Server](#)
- [New Language Features in SAS 9.2 for the Global Enterprise](#)

Here is a list of third-party documentation that should be read for in-depth information about Linguistic Collation.

- See the [Unicode Collation Algorithm \(UCA\) Specification](#).
- See the Collation section of the [ICU User Guide](#)
- For information about the collation rules, see [ICU Documentation: Collation](#).

Force SAS to Sort Data Sets Using System Option SORTPGM

You can force data sets to be sorted by SAS by setting system option SORTPGM=SAS. This system option must be set prior to using PROC SORT. Setting system option SORTPGM=SAS ensures that your data set is sorted in the order in which SAS expects.

By default, SORTPGM defaults to BEST, what SAS determines is probably the best performance choice. When SORTPGM is set to BEST, sorting might be performed by the following:

- a system or third-party host sorting utility program, if one is installed and available.
- a DBMS if the input data resides in a database and a SAS/ACCESS engine is used to read it.

To work properly, the host sort or DBMS and SAS must be configured for compatible operation. The host sort or the DBMS can be configured to order data differently from SAS. When the observations returned to SAS are not ordered as SAS expects, the SORTPGM= system option can be set to SAS to instruct SAS to sort the data in the order needed by SAS.

See “SORTPGM System Option” in *SAS System Options: Reference* and “In-Database Processing: PROC SORT” on page 2076.

Syntax: SORT Procedure

Requirement: BY statement

Supports: Threaded processing

Notes: Some SAS statements and options are not supported when running in CAS and might result in an error when PROC SORT is deciding whether to run in CAS. Some SAS statements and options are not supported when running in CAS but will not prevent operation in CAS. A note is printed for such options when PROC SORT decides to run in CAS. Most PROC SORT specific options do not have any effect and do not prevent running in CAS.

Tips: You can use the ATTRIB, FORMAT, LABEL, and WHERE statements with the PROC SORT procedure.

For in-database processing to occur, your data must reside within a supported version of a DBMS that has been properly configured for SAS In-Database processing. For more information, see “In-Database Processing: PROC SORT” on page 2076.

PROC SORT <collating-sequence-option> <other options>;

ATTRIB variable-list(s) attribute-list(s);

BY <DESCENDING> variable-1 <<DESCENDING> variable-2 ...>;

FORMAT variable-1 <...variable-n> <format> <DEFAULT=default-format>;

KEY variable(s) </ option>;

LABEL variable-1=label-1<...variable-n=label-n>;

WHERE where-expression-1
< logical-operator where-expression-n>;

Statement	Task	Example
PROC SORT	Order SAS data set observations by the values of one or more character or numeric variables	Ex. 1, Ex. 3, Ex. 4
BY	Specify the sorting variables	Ex. 1, Ex. 2, Ex. 4
KEY	Specify sorting keys and variables	

PROC SORT Statement

Orders SAS data set observations by the values of one or more character or numeric variables.

Examples:

[“Example 1: Sorting by the Values of Multiple Variables” on page 2080](#)

[“Example 3: Maintaining the Relative Order of Observations in Each BY Group” on page 2085](#)

[“Example 4: Retaining the First Observation of Each BY Group” on page 2088](#)

[“Example 7: Eliminate All Duplicate Observations Using NODUPKEY” on page 2096](#)

Syntax

PROC SORT *<collating-sequence-option>* *<other options>*;

Summary of Optional Arguments

DATA=SAS-data-set

specifies the input data set.

DATECOPY

sorts a SAS data set without changing the created and modified dates.

FORCE

forces redundant sorting.

OVERWRITE

deletes the input data set before the replacement output data set is populated.

PRESORTED

specifies whether the data set is likely already sorted.

SORTSIZE=memory-specification

specifies the available memory.

TAGSORT

reduces temporary disk usage.

Create output data sets

DUPOUT= *SAS-data-set*

specifies the output data set to which duplicate observations are written.

OUT= *SAS-data-set*

specifies the output data set.

UNIQUEOUT= *SAS-data-set*

specifies the output data set for eliminated observations.

Eliminate duplicate observations

NODUPKEY

deletes observations with duplicate BY values.

Eliminate unique observations

NOUNIQUEKEY

eliminates observations from the output data set that have a unique sort key.

Override SAS system option THREADS

NOTHEADS

prevents threaded sorting.

THREADS

enables or prevents the activation of threaded sorting.

Specify the collating sequence

ASCII

specifies ASCII.

DANISH

specifies Danish.

EBCDIC

specifies EBCDIC.

FINNISH

specifies Finnish.

NATIONAL

specifies a customized sequence.

NORWEGIAN

specifies Norwegian.

REVERSE

reverses the collation order for character variables.

SORTSEQ=*collating-sequence*

specifies the collating sequence.

SWEDISH

specifies Swedish.

Specify the output order

EQUALS

specifies the relative order within BY groups.

NOEQUALS

does not maintain relative order within BY groups.

Collating-Sequence-Options

You can specify only one *collating-sequence-option* and multiple *other options* in a PROC SORT step. The order of the two types of options does not matter and both types are not necessary in the same PROC SORT step.

ASCII

sorts character variables using the ASCII collating sequence. You need this option only when you want to achieve an ASCII ordering on a system where EBCDIC is the native collating sequence.

See [“ASCII Order” on page 2043](#)

DANISH

sorts characters according to the Danish and Norwegian convention.

The Danish and Norwegian collating sequence is shown in [Figure 55.58 on page 2053](#).

EBCDIC

sorts character variables using the EBCDIC collating sequence. You need this option only when you want to achieve an EBCDIC ordering on a system where ASCII is the native collating sequence.

See [“EBCDIC Order” on page 2044](#)

FINNISH

sorts characters according to the Finnish and Swedish convention.

The Finnish and Swedish collating sequence is shown in [Figure 55.58 on page 2053](#).

NATIONAL

sorts character variables using an alternate collating sequence, as defined by your installation, to reflect a country's National Use Differences. To use this option, your site must define a customized national sort sequence. Check with the SAS Installation Representative at your site to determine whether a customized national sort sequence is available.

NORWEGIAN

sorts characters according to the Danish and Norwegian convention.

The Norwegian collating sequence is shown in [Figure 55.58 on page 2053](#).

REVERSE

sorts character variables using a collating sequence that is reversed from the normal collating sequence.

For information about the normal collating sequence for your operating environment, see [“EBCDIC Order” on page 2044](#) and [“ASCII Order” on page 2043](#).

Restriction Only one collating-sequence-option can be specified.

Interaction Using REVERSE with the DESCENDING option in the BY statement restores the sequence to the normal order.

See The “[DESCENDING](#)” on page 2070 option in the BY statement. The difference is that the DESCENDING option can be used with both character and numeric variables.

SWEDISH

sorts characters according to the Finnish and Swedish convention.

The Finnish and Swedish collating sequence is shown in [Figure 55.58 on page 2053](#).

SORTSEQ=collating-sequence

The *collating-sequence* can be one of the following:

- [collating-sequence-option on page 2053](#)
- [translation_table on page 2053](#)
- [encoding-value on page 2054](#)
- [LINGUISTIC on page 2054](#)

For detailed information, refer to “[Collating Sequence](#)” in *SAS National Language Support (NLS): Reference Guide*.

Figure 55.1 National Collating Sequences of Alphanumeric Characters

Danish:	0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZÅäåabcdefghijklmnopqrstuvwxyzæøå
Finnish:	0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZÄÅäåabcdefghijklmnopqrstuvwxyzääö
Italian:	0123456789AÀBÇDÈÉÊFGHIÌJJKLMNOÏPQRSTUÙVWXYZaàbçdèéèfghiìjklmnoòpqrstuùvwxyz
Norwegian:	0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZÅäåabcdefghijklmnopqrstuvwxyzæøå
Spanish:	0123456789AÁaáBbCcDdEéEeFfGgHhIíIiJjKkLlMmNnÑñOóOoPpQqRrSsTtUúUúÜü\00fcVvWwXxYyZz
Swedish:	0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZÄÅäåabcdefghijklmnopqrstuvwxyzääö

Here are descriptions of the types of collating sequences:

collating-sequence-option

translation_table

specifies one of the PROC SORT statement *collating-sequence-options* (ASCII, DANISH, EBCDIC, FINNISH, NORWEGIAN, POLISH, REVERSE, SWEDISH) or a translation table, which can be one that SAS provides or any user-defined translation table. Translation tables provided by SAS are: ASCII, DANISH, EBCDIC, FINNISH, ITALIAN, NORWEGIAN, POLISH, REVERSE, SPANISH, and SWEDISH.

Restriction You can specify only one *collating-sequence-option* or one translation table for the SORTSEQ= option.

Interaction In-database processing will not occur when the SORTSEQ= option is specified.

Tip The SORTSEQ= collating sequence options are specified without parenthesis and there are no arguments associated with them.

See For a more detailed description of each *collating-sequence-option*, see “[Collating-Sequence-Options](#)” on page 2052.

To see the Sorting Order of Character variables, see [“EBCDIC Order” on page 2044](#), [“ASCII Order” on page 2043](#), and [Figure 55.58 on page 2053](#) for all others.

Example `proc sort data=mydata SORTSEQ=ASCII;`

Example For an example of using PROC TRANTAB and PROC SORT with SORTSEQ=, see [“Using Different Translation Tables for Sorting” in SAS National Language Support \(NLS\): Reference Guide](#).

encoding-value

specifies an encoding value. The result is the same as a binary collation of the character data represented in the specified encoding. See the supported [encoding value](#) in the [SAS National Language Support \(NLS\): Reference Guide](#).

Restriction PROC SORT is the only procedure or part of the SAS system that recognizes an encoding specified for the SORTSEQ= option.

Tip When the encoding value contains a character other than an alphanumeric character or underscore, the value needs to be enclosed in quotation marks.

See The list of the [encodings](#) that can be specified in the [SAS National Language Support \(NLS\): Reference Guide](#).

LINGUISTIC<(collating-options)>

specifies linguistic collation, which sorts characters in a culturally sensitive manner according to rules that are associated with a language and locale. The rules and default *collating-sequence* options are based on the language that is specified in the current locale setting. The implementation is provided by the International Components for Unicode (ICU) library. It produces results that are largely compatible with the Unicode Collation Algorithms (UCA). For more information, see [“Linguistic Sorting of Data Sets and ICU” on page 2046](#).

Note: Only PROC SORT and PROC SQL are affected when the linguistic collation system option is specified.

The following are options that can be used when specifying SORTSEQ=LINGUISTIC. These options modify the linguistic collating sequence:

ALTERNATE_HANDLING=SHIFTED

controls the handling of variable characters like spaces, punctuation, and symbols. When this option is not specified (using the default value Non-Ignorable), differences among these variable characters are of the same importance as differences among letters. If the ALTERNATE_HANDLING option is specified, these variable characters are of minor importance.

Default NON_IGNORABLE

- Tip** The SHIFTED value is often used in combination with STRENGTH= set to Quaternary. In such a case, spaces, punctuation, and symbols are considered when comparing strings, but only if all other aspects of the strings (base letters, accents, and case) are identical.
- See** [“Example 5: Linguistic Sorting Using ALTERNATE_HANDLING=” on page 2090](#) and [“Example 6: Linguistic Sorting Using ALTERNATE_HANDLING= and STRENGTH=” on page 2093](#).

CASE_FIRST=

specifies the order of uppercase and lowercase letters. This argument is valid for only TERTIARY, QUATERNARY, or IDENTICAL levels. The following table provides the values and information for the CASE_FIRST argument:

Table 55.3 Arguments for CASE_FIRST=

Value	Description
UPPER	Sorts uppercase letters first, then the lowercase letters.
LOWER	Sorts lowercase letters first, then the uppercase letters.

COLLATION=

specifies character ordering. The following table lists the available COLLATION= values.

Note: If you do not select a collation value, then the user's locale-default collation is selected.

Table 55.4 Values for COLLATION=

Value	Description
BIG5HAN	Specifies Pinyin ordering for Latin and specifies bug5 charset ordering for Chinese, Japanese, and Korean characters.
DIRECT	Specifies a Hindi variant.
GB21312HAN	Specifies Pinyin ordering for Latin and specifies gb2312han charset ordering for Chinese, Japanese, and Korean characters.
PHONEBOOK	Specifies a telephone-book style for ordering of characters. Select PHONEBOOK only with the German language.

Value	Description
PINYIN	Specifies an ordering for Chinese, Japanese, and Korean characters based on character-by-character transliteration into Pinyin. This ordering is typically used with simplified Chinese.
POSIX	Portable Operating System Interface. This option specifies a “C” locale ordering of characters.
STROKE	Specifies a nonalphabetic writing style ordering of characters. Select STROKE with Chinese, Japanese, Korean, or Vietnamese languages. This ordering is typically used with Traditional Chinese.
TRADITIONAL	Specifies a traditional style for ordering of characters. For example, select TRADITIONAL with the Spanish Language.

LOCALE= *locale_name*

specifies the locale name in the form of a POSIX name (for example, ja_JP). For a list of locale and POSIX values supported by PROC SORT, see “[Locale-to-Encoding Mapping](#)” in *SAS National Language Support (NLS): Reference Guide*.

Restriction The following Locales are not supported by PROC SORT:

- Afrikaans_SouthAfrica, af_ZA
- Cornish_UnitedKingdom, kw_GB
- ManxGaelic_UnitedKingdom, gv_GB

NUMERIC_COLLATION=

orders integer values within the text by the numeric value instead of characters used to represent the numbers.

Table 55.5 Values for NUMERIC_COLLATION

Value	Description
ON	Order numbers by the numeric value. For example, “8 Main St.” would sort before “45 Main St.”.
OFF	Order numbers by the character value. For example, “45 Main St.” would sort before “8 Main St.”.

Default OFF

STRENGTH=

The value of strength is related to the collation level. There are five collation-level values. The following table provides information about the five levels. The default value for strength is related to the locale.

Table 55.6 Values for *STRENGTH=*

Value	Type of Collation	Description
PRIMARY or 1	PRIMARY specifies differences between base characters (for example, "a" < "b").	It is the strongest difference. For example, dictionaries are divided into different sections by base character.
SECONDARY or 2	Accents in the characters are considered secondary differences (for example, "as" < "às" < "at").	A secondary difference is ignored when there is a primary difference anywhere in the strings. Other differences between letters can also be considered secondary differences, depending on the language.
TERTIARY or 3	Upper and lowercase differences in characters are distinguished at the tertiary level (for example, "ao" < "Ao" < "aò"). For an example, see "Example 5: Linguistic Sorting Using ALTERNATE_HANDLING=" on page 2090.	A tertiary difference is ignored when there is a primary or secondary difference anywhere in the strings. Another example is the difference between large and small Kana.

Value	Type of Collation	Description
QUATERNARY or 4	When punctuation is ignored at level 1-3, an additional level can be used to distinguish words with and without punctuation (for example, "a-b" < "ab" < "aB"). For an example, see "Example 6: Linguistic Sorting Using ALTERNATE_HANDLING=" and STRENGTH=" on page 2093.	The quaternary level should be used if ignoring punctuation is required or when processing Japanese text. This difference is ignored when there is a primary, secondary, or tertiary difference.
IDENTICAL or 5	When all other levels are equal, the identical level is used as a tiebreaker. The Unicode code point values of the Normalization Form D (NFD) form of each string are compared at this level, just in case there is no difference at levels 1-4.	This level should be used sparingly, because code-point value differences between two strings rarely occur. For example, only Hebrew cantillation marks are distinguished at this level.

Alias LEVEL=

Alias UCA

Restriction The SORTSEQ=LINGUISTIC option is available only on the PROC SORT SORTSEQ= option and is not available for the system option SORTSEQ.

Interaction The ICU version can change in a new SAS release. The order of observations produced when sorting a data set linguistically, using one release of SAS, can be different from the order produced by another release if the two releases use different versions of the ICU. When migrating to a new release of SAS, consider re-establishing the sort order of permanent data sets using PROC SORT with the PRESORTED option. For more details, see ["Linguistic Sorting of Data Sets and ICU"](#) on page 2046.

- Tips** The CONTENTS procedure or CONTENTS statement output shows the ICU version number of a data set that is linguistically sorted.
- The *collating-options* must be enclosed in parentheses. More than one collating option can be specified.
- When BY processing is performed on data sets that are sorted with linguistic collation, the NOBYSORTED system option might need to be specified in order for the data set to be treated properly. BY processing is performed differently than collating sequence processing.
- See** For ICU License Agreement, see [Appendix 3, “UNICODE License,” on page 2549](#).
- For more information, see [“Specifying Linguistic Collation” in SAS National Language Support \(NLS\): Reference Guide](#). For more information, see [“Linguistic Sorting of Data Sets and ICU” on page 2046](#).

CAUTION

If you use a host sort utility to sort your data, then specifying a translation-table-based collating sequence with the SORTSEQ= option might corrupt the character BY variables. For more information, see the PROC SORT documentation for your operating environment.

- Interaction** In-database processing does not occur when the SORTSEQ= option is specified.
- Tip** The SORTSEQ= *collating-sequence* options are specified without parenthesis and no arguments are associated with them. Here is an example of how to specify a collating sequence: `proc sort data=mydata SORTSEQ=ASCII;`

Other Options

Options can include one *collating-sequence-option* and multiple other options. The order of the two types of options does not matter and both types are not necessary in the same PROC SORT step.

DATA=SAS-data-set

identifies the input SAS data set.

- Restrictions** For in-database processing to occur, the data set must refer to a table residing on the DBMS.

SAS data set options DROP=, KEEP=, RENAME= are not supported in CAS. When these options are specified on the data set, CAS operations will not occur.

Notes	PROC SORT supports extended attributes by copying the attributes from the input data set to the output data set. When options NODUPKEY or NOUNIKEY are specified and the DATA= option refers to a CAS table, and the OUT=, DUPOUT=, or UNIOUT= options refer to a CAS table, PROC SORT invokes the CAS deduplicate action on a CAS server. For more information, see the deduplicate Action.
See	“Input Data Sets” on page 24 SAS Data Set Options: Reference

DATECOPY

copies the SAS internal date and time at which the SAS data set was created and the date and time at which it was last modified before the sort to the resulting sorted data set. Note that the operating-environment date and time are not preserved.

Restriction	DATECOPY can be used only when the resulting data set uses the V8 or V9 engine.
Tip	You can alter the file creation date and time with the DTC= option in the MODIFY statement in PROC DATASETS. For more information, see “MODIFY Statement” on page 446 .

DUPOUT= SAS-data-set

specifies the output data set to which duplicate observations are written.

Interactions	In-database processing does not occur when the DUPOUT= option is specified. The DUPOUT= and UNIQUEOUT= options are not compatible and cannot be specified simultaneously.
Note	When options NODUPKEY or NOUNIKEY are specified and the DATA= option refers to a CAS table, and the OUT=, DUPOUT=, or UNIOUT= options refer to a CAS table, PROC SORT invokes the CAS deduplicate action on a CAS server. For more information, see the deduplicate Action .
Tips	The DUPOUT= option can be used only with the NODUPKEY option. It cannot be combined with the NOUNIQUEKEY option. If the DUPOUT= data set name that is specified is the same as the INPUT data set name, SAS does not sort or overwrite the INPUT data set. Instead, SAS generates an error message. The FORCE option must be specified in order to overwrite the INPUT data set with the DUPOUT= data set of the same name.
See	SAS Data Set Options: Reference

EQUALS

specifies the order of the observations in the output data set. For observations with identical BY-variable values, EQUALS maintains the relative order of the observations within the input data set in the output data set. NOEQUALS does not necessarily preserve this order in the output data set.

Default EQUALS

Interactions When you use NODUPKEY to remove observations in the output data set, the choice of EQUALS or NOEQUALS can affect which observations are removed.

The EQUALS | NOEQUALS procedure option overrides the default sort stability behavior that is established with the SORTEQUALS | NOSORTEQUALS system option.

The EQUALS option is supported by the threaded sort. However, I/O performance might be reduced when using the EQUALS option with the threaded sort because partitioned data sets are processed as if they consist of a single partition.

The NOEQUALS option is supported by the threaded sort. The order of observations within BY groups that are returned by the threaded sort might not be consistent between runs.

Tip Using NOEQUALS can save CPU time and memory.

See [“NOEQUALS” on page 2063](#).

FORCE

sorts and replaces an indexed data set when the OUT= option is not specified. Without the FORCE option, PROC SORT does not sort and replace an indexed data set because sorting destroys user-created indexes for the data set. When you specify FORCE, PROC SORT sorts and replaces the data set and destroys all user-created indexes for the data set. Indexes that were created or required by integrity constraints are preserved.

Restriction If you use PROC SORT with the FORCE option on data sets that were created with the older compatibility engines or with a sequential engine such as a tape format engine, you might also need to specify the OUT= option.

Tip PROC SORT checks for the sort indicator before it sorts a data set so that data is not sorted again unnecessarily. By default, PROC SORT does not sort a data set if the sort information matches the requested sort. You can use FORCE to override this behavior. You might need to use FORCE if SAS cannot verify the sort specification in the data set option SORTEDBY=. For more information about [SORTEDBY=](#), see the chapter on SAS data set options in [SAS Data Set Options: Reference](#).

NODUPKEY

checks for and eliminates observations with duplicate BY values. If you specify this option, PROC SORT compares all BY values for each observation to the

ones for the previous observation that is written to the output data set. If an exact match is found, the observation is not written to the output data set.

Sorting causes observations that have equal BY variable values to be grouped together for output. Normally, PROC SORT writes all observations for each BY group that it assembles to the output data set. The NODUPKEY option instructs the SORT procedure to write only the first observation of each BY group to the output data set and to discard any additional observations that are contained within that BY group.

Several factors determine whether an observation occurs first within a BY group that PROC SORT has assembled for output. These factors include the order in which observations are read from the input data set, the sort program being used, and whether stability is provided by the sorting algorithm.

When the SORT procedure's input is a V9 engine data set and the sorting is done by SAS, the order of observations within an output BY group is predictable. The order of the observations within the group is the same as the order in which they were written to the data set when it was created. Because the V9 engine maintains observations in the order that they were written to the data set, they are read by PROC SORT in the same order. While processing, PROC SORT maintains the order of the observations because it uses a stable sorting algorithm. The stable sorting algorithm is used because the EQUALS option is set by default. Therefore, the observation that is selected by PROC SORT to be written to the output data set for a given BY group is the first observation in the data set having the BY variable values that define the group.

If the SORT procedure reads its input from an engine that does not provide a predictable observation order or an alternative sorting program (when a host sort performs the sort), the observations that are eliminated and the one that is written to the output data set might not be well defined. For example, determining which observations are kept and which observations are discarded might be unpredictable if data is being read into SAS from a DBMS that presents query results in a nondeterministic order due to parallel processing.

To ensure that each observation in the output data set is unique, you can use the NODUPKEY option with PROC SORT and sort by `_ALL_` variables in the input data set. An observation in a data set is unique when no other observation in the data set has the same combination of variable values. See example program [“Example 7: Eliminate All Duplicate Observations Using NODUPKEY” on page 2096](#).

Note: If you drop one or more BY variables from the output data set when using NODUPKEY, you void (eliminate) the guarantee that each observation has a unique set of BY variable values. Similarly, when you use NODUPKEY and sort by `_ALL_` variables to produce unique observations, if you drop one or more variables from the output data set, observations in the output data set are no longer guaranteed to be unique.

Another way to ensure that observations in an output data set are unique is to use PROC SQL with the DISTINCT keyword. Here is a simple example using PROC SQL:

```
PROC SQL;
```

```
CREATE TABLE DL (keep division league) AS SELECT DISTINCT *
FROM SASHELP.BASEBALL;
QUIT;
```

Interactions The V9 engine provides a consistent ordering where the first observation (the first observation that was written and stored) is generally the first one that is read by PROC SORT. The sorted data set contains only the first observation of each BY group that PROC SORT reads.

When you are removing observations with duplicate BY values with NODUPKEY, the choice of EQUALS or NOEQUALS can have an effect on which observations are removed. Use the EQUALS option (the default value) with the NODUPKEY option for consistent results in your output data sets.

In-database sorting occurs when the NODUPKEY option is specified and the system option SQLGENERATION= is assigned a DBMS and the system option SORTPGM=BEST.

The options NODUPKEY and NOUNIQUEKEY are not compatible. If these options are specified together, an error is printed to the SAS log.

Note When options NODUPKEY or NOUNIQUEKEY are specified and the DATA= option refers to a CAS table, and the OUT=, DUPOUT=, or UNIOUT= options refer to a CAS table, PROC SORT invokes the CAS deduplicate action on a CAS server. For more information, see the [deduplicate Action](#).

Tip The DUPOUT= option can be used with the NODUPKEY option. However, it cannot be used with the NOUNIQUEKEY option.

Examples [“Example 4: Retaining the First Observation of Each BY Group” on page 2088](#)

[“Example 7: Eliminate All Duplicate Observations Using NODUPKEY” on page 2096](#)

NOEQUALS

See [“EQUALS” on page 2061](#).

NOTHEADS

See [“Threading in Base SAS” in SAS Programmer’s Guide: Essentials](#).

NOUNIQUEKEY

checks for and eliminates observations from the output data set that have a unique sort key. A sort key is unique when the observation containing the key is the only observation within a BY group.

Note: Unlike NODUPKEY, which writes one observation of a BY group to the output data set and discards all other observations from the BY group, the NOUNIQUEKEY maintains BY group integrity. Either all observations of a BY group are written to the output data set when the BY group consists of two or

more observations, or all observations of the BY group are discarded when the BY group consists of a single observation.

Aliases	NOUNIKEY NOUNIKEYS NOUNIQUEKEYS
Interaction	Options NODUPKEY and NOUNIQUEKEY are not compatible. If NODUPKEY and NOUNIQUEKEY are specified together, an error is printed to the SAS log.
Note	When options NODUPKEY or NOUNIKEY are specified and the DATA= option refers to a CAS table, and the OUT=, DUPOUT=, or UNIOUT= options refer to a CAS table, PROC SORT invokes the CAS deduplicate action on a CAS server. For more information, see the deduplicate Action .
Tip	The UNIQUEOUT= option can be used with the NOUNIQUEKEY option. It cannot be combined with the NODUPKEY option.
See	UNIQUEOUT= to direct the observations that have been eliminated to an output data set.

OUT= SAS-data-set

names the output data set. If *SAS-data-set* does not exist, then PROC SORT creates it.

CAUTION

Use care when you use PROC SORT without OUT=. Without the OUT= option, PROC SORT replaces the original data set with the sorted observations when the procedure executes without errors.

Default	Without OUT=, PROC SORT replaces the original data set.
Note	When options NODUPKEY or NOUNIKEY are specified and the DATA= option refers to a CAS table, and the OUT=, DUPOUT=, or UNIOUT= options refer to a CAS table, PROC SORT invokes the CAS deduplicate action on a CAS server. For more information, see the deduplicate Action .
Tips	With in-database sorts, the output data set cannot refer to the input table on the DBMS. You can use data set options with OUT=.
See	SAS Data Set Options: Reference
Example	“Example 1: Sorting by the Values of Multiple Variables” on page 2080

OVERWRITE

enables the input data set to be deleted before the replacement output data set of the same name is populated with observations.

CAUTION

Use the OVERWRITE option only with a data set that is backed up or with a data set that you can reconstruct. Because the input data set is deleted, data is lost if a failure occurs while the output data set is being written.

Restrictions If the OVERWRITE and OUT= options are specified and the OUT= data set name is not the same as the INPUT data set name, SAS does not overwrite the INPUT data set.

The OVERWRITE option has no effect if you also specify the TAGSORT option. You cannot overwrite the input data set because TAGSORT must reread the input data set while populating the output data set.

The OVERWRITE option is supported by the SAS sort and SAS threaded sort only. The option has no effect if you are using a host sort.

Tip Using the OVERWRITE option can reduce disk space requirements.

PRESORTED

before sorting, checks within the input data set to determine whether the sequence of observations is in order. Use the PRESORTED option when you know or strongly suspect that a data set is already in order according to the key variables that are specified in the BY statement. By specifying this option, you avoid the cost of sorting the data set.

Interaction Sequence checking is not performed when the [“FORCE” on page 2061](#) option is specified.

Tips You can use the DATA step to import data, from external text files, in a sequence compatible with SAS processing and according to the sort order specified by the combination of SORT options and key variables listed in the BY statement. You can then specify the PRESORTED option if you know or highly suspect that the data is sorted accordingly.

Using the PRESORTED option with ACCESS engines and DBMS data is not recommended. These external databases are not guaranteed to return observations in sorted order unless an ORDER BY clause is specified in a query. Generally, physical ordering is not a concept that external databases use. Therefore, these databases are not guaranteed to return observations in the same order when executing a query multiple times. Physical order can be important for producing consistent, repeatable results when processing data. Without a repeatable data retrieval order, PROC SORT does not guarantee the return of observations in the same order from one PROC SORT execution to another, even when the [“EQUALS” on page 2061](#) option is used to request sort stability. Without a repeatable

retrieval order, the detection and elimination of adjacent duplicate records by PROC SORT can also vary from one PROC SORT execution to another.

See System option [“SORTVALIDATE System Option” in SAS System Options: Reference](#).

SORTSIZE=memory-specification

specifies the maximum amount of memory that is available to PROC SORT. Valid values for *memory-specification* are as follows:

MAX

specifies that all available memory can be used.

n

specifies the amount of memory in bytes, where *n* is a real number.

nK

specifies the amount of memory in kilobytes, where *n* is a real number.

nM

specifies the amount of memory in megabytes, where *n* is a real number.

nG

specifies the amount of memory in gigabytes, where *n* is a real number.

Specifying the SORTSIZE= option in the PROC SORT statement temporarily overrides the SAS system option. For more information, see [“SORTSIZE= System Option” in SAS System Options: Reference](#).

Operating Environment Information: Some system sort utilities might treat this option differently. Refer to the SAS documentation for your operating environment.

Alias SIZE=

Default the value of the SAS system option SORTSIZE=

Tips Setting the SORTSIZE= option in the PROC SORT statement to MAX or 0, or not setting the SORTSIZE= option, limits the PROC SORT to the available physical memory based on the settings of the SAS system options REALMEMSIZE and MEMSIZE.

For information about the SAS system options REALMEMSIZE and MEMSIZE, see the SAS documentation for your operating environment.

TAGSORT

stores only the BY variables and the observation numbers in temporary files. The BY variables and the observation numbers are called *tags*. At the completion of the sorting process, PROC SORT uses the tags to retrieve records from the input data set in sorted order.

Note: The utility file created is much smaller than it would be if the TAGSORT option were not specified.

- Restriction** The TAGSORT option is not compatible with the OVERWRITE option.
- Interaction** The TAGSORT option is not supported by the threaded sort.
- Tip** When the total length of BY variables is small compared with the record length, TAGSORT reduces temporary disk usage considerably. However, processing time might be much higher.

THREADS

enables or prevents the activation of threaded sorting.

- Default** The value of the THREADS | NOTTHREADS SAS system option. Note that the default can be overridden using the SORT procedure THREADS | NOTTHREADS option.

- Restrictions** Your site administrator can create a restricted options table. A restricted options table specifies SAS system option values that are established at start-up and cannot be overridden. If the THREADS | NOTTHREADS system option is listed in the restricted options table, any attempt to set these system options is ignored and a warning message is written to the SAS log.

If a failure occurs when adding the THREADS | NOTTHREADS procedure option using the SPD Engine, PROC SORT stops processing and writes a message to the SAS log.

- Interactions** The PROC SORT THREADS | NOTTHREADS options override the SAS THREADS | NOTTHREADS options unless the system option is restricted. (See Restriction.) For more information, see [“THREADS System Option” in SAS System Options: Reference](#).

The THREADS system option is honored if PROC SORT determines that threaded processing is deemed to be beneficial. If the value of the SAS system option CPUCOUNT=1, then threaded processing is not beneficial. However, you can specify the PROC SORT THREADS option to force threaded processing when the system option is set to NOTTHREADS or when the system option is THREADS and the procedure option is NOTTHREADS. This option combination prevents threaded processing and overrides the actions taken that are based on the system options. Note that when threaded sorting is in effect and NOEQUALS is specified, observations within BY groups might be returned in an unpredictable order.

If threaded SAS sort is being used, the UTILLOC= system option affects the placement of utility files. Thread-enabled SAS applications are able to create temporary files that can be accessed in parallel by separate threads. For more information, see [“UTILLOC= System Option” in SAS System Options: Reference](#).

The page size of the utility file used by PROC SORT is influenced by the new STRIPESIZE= system option. For more information, see [“STRIPESIZE= System Option” in SAS System Options: Reference](#).

The TAGSORT option is not supported by the threaded sort. Specifying the TAGSORT option prevents threaded processing.

See [“NOTHEADS” on page 2063](#).

[“Threaded Sorting” on page 2042](#) and [“Threading in Base SAS” in SAS Programmer’s Guide: Essentials](#).

UNIQUEOUT=SAS-data-set

specifies the output data set for observations eliminated by the NOUNIQUEKEY option.

Alias UNIOUT=

Interaction The DUPOUT= and = options are not compatible and cannot be specified simultaneously.

Note When options NODUPKEY or NOUNIKEY are specified and the DATA= option refers to a CAS table, and the OUT=, DUPOUT=, or UNIOUT= options refer to a CAS table, PROC SORT invokes the CAS deduplicate action on a CAS server. For more information, see the [deduplicate Action](#).

Tip The UNIQUEOUT= option can be used with the NOUNIQUEKEY option. It cannot be combined with the NODUPKEY option.

See [“NOUNIQUEKEY” on page 2063](#)

[SAS Data Set Options: Reference](#)

ATTRIB Statement

Associates a format, informat, label, and length with one or more variables.

Syntax

ATTRIB *variable-list(s) attribute-list(s);*

Arguments

variable-list(s)

names the variables that you want to associate with the attributes.

TIP List the variables in any form that SAS allows.

attribute-list(s)

specifies one or more attributes to assign to *variable-list*. Specify one or more of these attributes in the ATTRIB statement:

FORMAT=*format*

associates a format with variables in *variable-list*.

Tip The format can be either a standard SAS format or a format that is defined with the FORMAT procedure.

INFORMAT=*informat*

associates an informat with variables in *variable-list*.

Tip The informat can be either a standard SAS informat or an informat that is defined with the FORMAT procedure.

LABEL=*'label'*

associates a label with variables in *variable-list*.

See Also

[“ATTRIB Statement” in SAS DATA Step Statements: Reference.](#)

BY Statement

Specifies the sorting variables.

Examples: [“Example 1: Sorting by the Values of Multiple Variables” on page 2080](#)
[“Example 2: Sorting in Descending Order” on page 2083](#)
[“Example 4: Retaining the First Observation of Each BY Group” on page 2088](#)

Syntax

BY <DESCENDING> *variable-1* <<DESCENDING> *variable-2* ...>;

Required Argument

variable

specifies the variable by which PROC SORT sorts the observations. PROC SORT first arranges the data set by the values in ascending order, by default, of the first BY variable. PROC SORT then arranges any observations that have the

same value of the first BY variable by the values of the second BY variable in ascending order. This sorting continues for every specified BY variable.

Optional Argument

DESCENDING

reverses the sort order for the variable that immediately follows in the statement so that observations are sorted from the largest value to the smallest value. The DESCENDING keyword modifies the variable that follows it.

Tips In a PROC SORT BY statement, the DESCENDING keyword modifies the variable that follows it.

The THREADS SAS system option is the default as long as the PROC SORT THREADS | NOTTHREADS option is unspecified.

Example [“Example 2: Sorting in Descending Order” on page 2083](#)

FORMAT Statement

Associates formats with variables.

Syntax

FORMAT *variable-1* <...*variable-n*> *format variable-1* <...*variable-n*> *format*;

Arguments

variable

names one or more variables for SAS to associate with a format. You must specify at least one *variable*.

Tip To disassociate a format from a variable, use the variable in a FORMAT statement without specifying a format in a DATA step or in PROC DATASETS. In a DATA step, place this FORMAT statement after the SET statement. See [“Removing a Format” in SAS DATA Step Statements: Reference](#). You can also use PROC DATASETS.

format

specifies the format that is listed for writing the values of the variables.

Tip Formats that are associated with variables by using a FORMAT statement behave like formats that are used with a colon modifier in a subsequent PUT statement. For more information about using a colon modifier, see [“PUT Statement: List” in SAS DATA Step Statements: Reference](#).

See [SAS Formats and Informats: Reference](#)

KEY Statement

Specifies sorting keys and variables. The KEY statement is an alternative to the BY statement. The KEY statement syntax allows for the future possibility of specifying different collation options for each KEY variable. Currently, the only options allowed are ASCENDING and DESCENDING.

Restriction: The BY statement cannot be used with the KEY statement.

Tip: Multiple KEY statements can be specified.

Syntax

KEY *variable(s)* *</ option>* ;

Required Argument

variable(s)

specifies the variable by which PROC SORT orders the observations. Multiple variables can be specified. Each of these variables must be separated by a space. A range of variables can also be specified. For example, the following code shows how to specify multiple variables and a range of variables:

```
data sortKeys;
    input x1 x2 x3 x4 ;
cards;
    7 8 9 8
    0 0 0 0
    1 2 3 4 ;
run;
proc sort data=sortKeys out=sortedOutput;
    key x1 x2-x4;
run;
```

Multiple KEY statements can also be specified. The first sort key encountered from among all sort keys is considered the primary sort key. Sorting continues for every specified KEY statement and its variables. For example, the following code shows how to specify multiple KEY statements:

```
proc sort data=sortKeys out=sortedOutput;
    key x2;
    key x3;
run;
```

The following code example uses the BY statement to accomplish the same type of sort as the previous example:

```
proc sort data=sortKeys out=sortedOutput;
    by x2 x3;
```

```
run;
```

Optional Arguments

ASCENDING

sorts in ascending order the variable or variables that it follows. Observations are sorted from the smallest value to the largest value. The **ASCENDING** keyword modifies all the variables that precede it in the **KEY** statement.

Alias **ASC**

Default **ASCENDING** is the default sort order.

Tip In a **PROC SORT KEY** statement, the **ASCENDING** option modifies all the variables that it follows. The option must follow the **/**. In the following example, the **x1** variable in the input data set is sorted in ascending order.

```
proc sort data=sortVar out=sortedOutput;
  key x1 / ascending;
run;
```

DESCENDING

reverses the sort order for the variable that it follows in the statement so that observations are sorted from the largest value to the smallest value. The **DESCENDING** keyword modifies all the variables that it precedes in the **KEY** statement.

Alias **DESC**

Default **ASCENDING (ASC)** is the default sort order.

Tip In a **PROC SORT KEY** statement, the **DESCENDING** option modifies the variables that follows it. The option must follow the **/**. In the following example, the **x1** and **x2** variables in the input data set is sorted in descending order:

```
proc sort data=sortVar out=sortedOutput;
  key x1 x2 / descending;
run;
```

The following example uses the **BY** statement to accomplish the same type of sort as the previous example:

```
proc sort data=sortVar out=sortedOutput;
  by descending x1 descending x2 ;
run;
```

LABEL Statement

Assigns descriptive labels to variables.

Syntax

LABEL *variable-1* = '*text-string*' <...*variable-n* = '*text-string*'>;

LABEL *variable-1* = '' <...*variable-n*=''>; | *variable-1* = <...*variable-n*=>;

Arguments

variable

specifies the variable that you want to label. You can specify labels for multiple variables in a single LABEL statement:

```
data test;
  L = 5;
  W = 10;
  label L = 'Length' W = 'Width';
run;
```

text-string

specifies a label of up to 256 bytes.

```
data test;
  L = 5;
  label L = 'Length';
run;
```

Restrictions If the label includes a semicolon (;) or an equal sign (=), you must enclose the label in either single or double quotation marks.

```
label N = 'Number = ';
```

If the label includes single quotation marks ('), you must enclose the label in double quotation marks.

```
label Name = "Owner's Last Name";
```

There is partial support for variable name substitution with the JAVAIMG device. When the label text for the variable exceeds the space allowed, the variable name is used instead by procedures that are labeling an axis or a legend title. With the exception of the SAS/GRAPH GPLOT procedure, the variable name is not used when the JAVAIMG device is specified.

Notes Quotation marks are not required unless the label includes a semicolon, quotation marks, or an equal sign.

The LABEL statement expects an equal sign (=) after the first variable. If any other character is found, an error occurs. The LABEL statement considers everything after the equal sign, following the first variable, as part of the label until the statement comes to another variable followed by an equal sign, regardless of quotation marks. In this example, the label for x is **this is x y 4 = this is y** because the next variable that is followed by an equal sign is z.

```
data test;
```

```

x = 1;
y = 1;
z = 1;
label x = 'this is x' y 4 = 'this is y' z = 'This is z';
run;

```

In this LABEL statement, the x variable has the same label:

```
label x = this is x y 4 = this is y z = This is z;
```

See [“Character Constants and Character Variables” in SAS Programmer’s Guide: Essentials.](#)

“

removes a label from a variable. Enclose a single blank space in quotation marks to remove an existing label. You can remove labels from multiple variables in a single LABEL statement:

```

data test;
  L=5; W=10;
  label L = ' ' W = ' ';
run;

```

Alternatively, you can remove a label by specifying nothing after the equal sign:

```

data test;
  L=5; W=10;
  label L = W = ;
run;

```

WHERE Statement

Subsets the input data set by specifying certain conditions that each observation must meet before it is available for processing.

Syntax

WHERE *where-expression-1*
 < *logical-operator where-expression-n*>;

Arguments

where-expression

is an arithmetic or logical expression that generally consists of a sequence of operands and operators.

Tips The operands and operators described in the next several sections are also valid for the WHERE= data set option.

You can specify multiple where-expressions.

logical-operator

can be AND, AND NOT, OR, or OR NOT.

Example

Procedures That Support the WHERE Statement

You can use the WHERE statement with any of the following Base SAS procedures that read a SAS data set:

COMPARE	REPORT
CORR	SORT
DATASETS (APPEND statement)	SQL
FREQ	STANDARD
MEANS or SUMMARY	TABULATE
PRINT	TRANSPOSE
RANK	UNIVARIATE

Details

- The COMPARE procedure and the APPEND statement in PROC DATASETS accept more than one input data set. For more information, see the documentation for the specific procedure.
- To subset the output data set, use the WHERE= data set option:

```
proc report data=debate nowd  
    out=onlyfr (where= (year='1')) ;  
run;
```

For more information about WHERE=, see [SAS DATA Step Statements: Reference](#).

Temporarily Dissociating a Format from a Variable

In this example, PROC PRINT prints only those observations that meet the condition of the WHERE expression.

```
options nodate pageno=1 linesize=64
      pagesize=40;

proc print data=debate noobs;
  where gpa>3.5;
  title 'Team Members with a GPA';
  title2 'Greater than 3.5';
run;
```

Output

Team Members with a GPA Greater than 3.5			
Name	Gender	Year	GPA
Capiccio	m	Freshman	3.598
Tucker	m	Freshman	3.901
Bagwell	f	Sophomore	3.722
Gold	f	Junior	3.609
Syme	f	Junior	3.883
Baglione	f	Senior	4.000
Carr	m	Senior	3.750
Hall	m	Senior	3.574

Usage: SORT Procedure

In-Database Processing: PROC SORT

In-database processing has several advantages over processing within SAS. These advantages include increased security, reduced network traffic, and the potential for faster processing. Increased security is possible because sensitive data does not

have to be extracted from the DBMS. Faster processing is possible because data is manipulated locally, on the DBMS, using high-speed secondary storage devices instead of being transported across a relatively slow network connection, because the DBMS might have more processing resources at its disposal, and because the DBMS might be capable of optimizing a query for execution in a highly parallel and scalable fashion.

When the DATA= input data set is stored as a table or view in a database management system (DBMS), the PROC SORT procedure can use in-database processing to sort the data. In-database processing can provide the advantages of faster processing and reduced data transfer between the database and SAS software.

In-database processing for PROC SORT supports the following database management systems:

- Amazon Redshift
- DB2
- Google BigQuery

Note: In-database processing for PROC SORT requires a BY variable that is not of type FLOAT64.

- Greenplum
- Hadoop
- Impala
- Microsoft SQL Server
- MySQL
- Netezza
- Oracle
- PostgreSQL
- SAP HANA
- Snowflake
- Spark
- Teradata
- Vertica

PROC SORT performs in-database processing using SQL explicit pass-through. The pass-through facility uses SAS/ACCESS to connect to a DBMS and to send statements directly to the DBMS for execution. This facility lets you use the SQL syntax of your DBMS. For details, see "Pass-Through Facility for Relational Databases" in [SAS/ACCESS for Relational Databases: Reference](#).

In-database processing is used by PROC SORT when a combination of procedure and system options are properly set. When system option SORTPGM=BEST, system option SQLGENERATION= is set to cause in-database processing, and when the PROC SORT NODUPKEY option is specified, PROC SORT generates a

DBMS SQL query that sorts the data. The sorted results can either remain as a new table within the DBMS or can be returned to SAS. To view the SQL queries generated, set the `SASTRACE=` option.

The SAS system option `SORTPGM=` can also be used without setting the `SQLGENERATION` option to instruct PROC SORT to use either the DBMS, SAS, or the HOST to perform the sort. If `SORTPGM=BEST` is specified, then either the DBMS, SAS, or HOST performs the sort. The observation ordering that is produced by PROC SORT depends on whether the DBMS or SAS performs the sorting.

If the DBMS performs the sort, then the configuration and characteristics of the DBMS sorting program affects the resulting data order. The DBMS configuration settings and characteristics that can affect data order include character collation, ordering of NULL values, and sort stability. Most database management systems do not guarantee sort stability, and the sort might be performed by the DBMS regardless of the state of the `SORTEQUALS/NOSORTEQUALS` system option and `EQUALS/NOEQUALS` procedure option.

If you set the SAS system option `SORTPGM=` to SAS, then unordered data is delivered from the DBMS to SAS and SAS performs the sorting. However, consistency in the delivery order of observations from a DBMS is not guaranteed. Therefore, even though SAS can perform a stable sort on the DBMS data, SAS cannot guarantee that the ordering of observations within output BY groups is the same from one PROC SORT execution to the next. To achieve consistency in the ordering of observations within BY groups, first populate a SAS data set with the DBMS data, and then use the `EQUALS` or `SORTEQUALS` option to perform a stable sort.

In-database processing is affected by the following circumstances:

- When PROC SORT options, `SORTSEQ=`, or `DUPOUT=`, are specified, no in-database processing occurs.
- For in-database processing, the `OUT=` procedure option must be specified and the output data set cannot refer to the input table on the DBMS.
- LIBNAME options and data set options can also affect whether in-database processing occurs and what type of query is generated. See "In-Database Procedures" in [SAS/ACCESS for Relational Databases: Reference](#) for a complete list of these options. The user can also set `OPTIONS MSGLEVEL=I` in SAS to see which options prevent or affect in-database processing.

SAS Cloud Analytic Services Processing for PROC SORT

PROC SORT supports the ability to run the duplicate detection and manipulation options (`NODUPKEY`, `NOUNIQUE`, `DUPOUT=`, `UNIOUT=`) in CAS. This functionality is provided to facilitate the migration of code for users of the SAS Viya platform. The de-duplicate action is used to eliminate duplicate entries in data sets in CAS.

When the input data set references the CAS view or an in-memory table, the SORT procedure uses the CAS de-duplicate action to perform the equivalent functionality

of the NODUPKEY or NOUNIKEY options of PROC SORT. When options NODUPKEY or NOUNIKEY are specified in the PROC SORT statement and input to PROC SORT is read by CAS, and all output from PROC SORT (including the optional DUPOUT= and = UNIOUT data sets) is written to a CAS table, PROC SORT invokes the CAS de-duplicate action on the CAS server. Under these conditions, the action is invoked and the work is performed in CAS with no data moved into or out of the CAS server.

See the [deduplicate Action](#) for information about the CAS aAction.

For conceptual information about procedures that run in CAS, see [Chapter 4, "CAS Processing of Base Procedures,"](#) on page 75.

Integrity Constraints: SORT Procedure

Sorting the input data set and replacing it with the sorted data set preserves both referential and general integrity constraints, as well as any indexes that they might require. A sort that creates a new data set does not preserve any integrity constraints or indexes. For more information about implicit replacement, explicit replacement, and no replacement with and without the OUT= option, see ["Output Data Set" on page 2079](#). For more information about integrity constraints, see ["Integrity Constraints" in SAS V9 LIBNAME Engine: Reference](#).

Results: SORT Procedure

Procedure Output

PROC SORT produces only an output data set. To see the output data set, you can use PROC PRINT, PROC REPORT, or another of the many available methods of printing in SAS.

Output Data Set

Without the OUT= option, PROC SORT replaces the original data set with the sorted observations when the procedure executes without errors. When you specify the OUT= option using a new data set name, PROC SORT creates a new data set that contains the sorted observations.

Table 55.7 Data Set Replacement Options

Task	Options
implicit replacement of input data set	proc sort data=names;
explicit replacement of input data set	proc sort data=names out=names;
no replacement of input data set	proc sort data=names out=namesbyid;

With all three replacement options (implicit replacement, explicit replacement, and no replacement) there must be at least enough space in the output library for a copy of the original data set.

You can also sort compressed data sets. If you specify a compressed data set as the input data set and omit the OUT= option, then the input data set is sorted and remains compressed. If you specify an OUT= data set, then the resulting data set is compressed only if you choose a compression method with the COMPRESS= data set option. For more information, see [“COMPRESS= Data Set Option” in SAS Data Set Options: Reference](#).

Also note that PROC SORT manipulates the uncompressed observation in memory and, if there is insufficient memory to complete the sort, stores the uncompressed data in a utility file. For these reasons, sorting compressed data sets might be intensive and require more storage than anticipated. Consider using the TAGSORT option when sorting compressed data sets.

Note: If the SAS system option NOREPLACE is in effect, then you cannot replace an original permanent data set with a sorted version. You must either use the OUT= option or specify the SAS system option REPLACE in an OPTIONS statement. The SAS system option NOREPLACE does not affect temporary SAS data sets.

Examples: SORT Procedure

Example 1: Sorting by the Values of Multiple Variables

- Features:
- PROC SORT statement option

OUT=

BY statement

PROC PRINT

Details

This example does the following:

- sorts the observations by the values of two variables
- creates an output data set for the sorted observations
- prints the results

Program

```
data account;
    input Company $ 1-22 Debt 25-30 AccountNumber 33-36
           Town $ 39-51;
    datalines;
Paul's Pizza            83.00  1019  Apex
World Wide Electronics  119.95  1122  Garner
Strickland Industries   657.22  1675  Morrisville
Ice Cream Delight        299.98  2310  Holly Springs
Watson Tabor Travel      37.95   3131  Apex
Boyd & Sons Accounting  312.49  4762  Garner
Bob's Beds              119.95  4998  Morrisville
Tina's Pet Shop          37.95   5108  Apex
Elway Piano and Organ    65.79   5217  Garner
Tim's Burger Stand       119.95  6335  Holly Springs
Peter's Auto Parts        65.79   7288  Apex
Deluxe Hardware          467.12  8941  Garner
Pauline's Antiques       302.05  9112  Morrisville
Apex Catering            37.95   9923  Apex
;

proc sort data=account out=bytown;

    by town company;
run;

proc print data=bytown;

    var company town debt accountnumber;

    title  'Customers with Past-Due Accounts';
    title2 'Listed Alphabetically within Town';
run;
```

Program Description

Create the input data set ACCOUNT. ACCOUNT contains the name of each business that owes money, the amount of money that it owes on its account, the account number, and the town where the business is located.

```
data account;
  input Company $ 1-22 Debt 25-30 AccountNumber 33-36
        Town $ 39-51;
  datalines;
Paul's Pizza          83.00  1019  Apex
World Wide Electronics 119.95  1122  Garner
Strickland Industries 657.22  1675  Morrisville
Ice Cream Delight     299.98  2310  Holly Springs
Watson Tabor Travel   37.95  3131  Apex
Boyd & Sons Accounting 312.49  4762  Garner
Bob's Beds           119.95  4998  Morrisville
Tina's Pet Shop       37.95  5108  Apex
Elway Piano and Organ 65.79  5217  Garner
Tim's Burger Stand    119.95  6335  Holly Springs
Peter's Auto Parts    65.79  7288  Apex
Deluxe Hardware       467.12  8941  Garner
Pauline's Antiques    302.05  9112  Morrisville
Apex Catering         37.95  9923  Apex
;
```

Create the output data set BYTOWN. OUT= creates a new data set for the sorted observations.

```
proc sort data=account out=bytown;
```

Sort by two variables. The BY statement specifies that the observations should be first ordered alphabetically by town and then by company.

```
  by town company;
run;
```

Print the output data set BYTOWN. PROC PRINT prints the data set BYTOWN.

```
proc print data=bytown;
```

Specify the variables to be printed. The VAR statement specifies the variables to be printed and their column order in the output.

```
  var company town debt accountnumber;
```

Specify the titles.

```
  title 'Customers with Past-Due Accounts';
  title2 'Listed Alphabetically within Town';
run;
```


Output: HTML

Output 55.3 *Sorting by the Values of Multiple Variables*

Customers with Past-Due Accounts Listed Alphabetically within Town				
Obs	Company	Town	Debt	AccountNumber
1	Apex Catering	Apex	37.95	9923
2	Paul's Pizza	Apex	83.00	1019
3	Peter's Auto Parts	Apex	65.79	7288
4	Tina's Pet Shop	Apex	37.95	5108
5	Watson Tabor Travel	Apex	37.95	3131
6	Boyd & Sons Accounting	Garner	312.49	4762
7	Deluxe Hardware	Garner	467.12	8941
8	Elway Piano and Organ	Garner	65.79	5217
9	World Wide Electronics	Garner	119.95	1122
10	Ice Cream Delight	Holly Springs	299.98	2310
11	Tim's Burger Stand	Holly Springs	119.95	6335
12	Bob's Beds	Morrisville	119.95	4998
13	Pauline's Antiques	Morrisville	302.05	9112
14	Strickland Industries	Morrisville	657.22	1675

Example 2: Sorting in Descending Order

Features: This example BY statement option
DESCENDING
PROC PRINT

Data set: Account

Details

This example does the following:

- sorts the observations by the values of three variables
- sorts one of the variables in descending order

- prints the results

Program

```
proc sort data=account out=sorted;
    by town descending debt accountnumber;
run;

proc print data=sorted;
    var company town debt accountnumber;
    title  'Customers with Past-Due Accounts';
    title2 'Listed by Town, Amount, Account Number';
run;
```

Program Description

Create the output data set SORTED. OUT= creates a new data set for the sorted observations.

```
proc sort data=account out=sorted;
```

Sort by three variables with one in descending order. The BY statement specifies that observations should be first ordered alphabetically by town, then by descending value of amount owed, then by ascending value of the account number.

```
    by town descending debt accountnumber;
run;
```

Print the output data set SORTED. PROC PRINT prints the data set SORTED.

```
proc print data=sorted;
```

Specify the variables to be printed. The VAR statement specifies the variables to be printed and their column order in the output.

```
    var company town debt accountnumber;
```

Specify the titles.

```
    title  'Customers with Past-Due Accounts';
    title2 'Listed by Town, Amount, Account Number';
run;
```

Output: HTML

Note that sorting last by AccountNumber puts the businesses in Apex with a debt of \$37.95 in order of account number.

Output 55.4 Sorting in Descending Order

Customers with Past-Due Accounts Listed by Town, Amount, Account Number				
Obs	Company	Town	Debt	AccountNumber
1	Paul's Pizza	Apex	83.00	1019
2	Peter's Auto Parts	Apex	65.79	7288
3	Watson Tabor Travel	Apex	37.95	3131
4	Tina's Pet Shop	Apex	37.95	5108
5	Apex Catering	Apex	37.95	9923
6	Deluxe Hardware	Garner	467.12	8941
7	Boyd & Sons Accounting	Garner	312.49	4762
8	World Wide Electronics	Garner	119.95	1122
9	Elway Piano and Organ	Garner	65.79	5217
10	Ice Cream Delight	Holly Springs	299.98	2310
11	Tim's Burger Stand	Holly Springs	119.95	6335
12	Strickland Industries	Morrisville	657.22	1675
13	Pauline's Antiques	Morrisville	302.05	9112
14	Bob's Beds	Morrisville	119.95	4998

Example 3: Maintaining the Relative Order of Observations in Each BY Group

Features: PROC SORT statement option
EQUALS | NOEQUALS
PROC PRINT

Details

This example does the following:

- sorts the observations by the value of the first variable
- maintains the relative order with the EQUALS option
- does not maintain the relative order with the NOEQUALS option

Program

```
data insurance;
  input YearsWorked 1 InsuranceID 3-5;
  datalines;
5 421
5 336
1 209
1 564
3 711
3 343
4 212
4 616
;

proc sort data=insurance out=byyears1 equals;
  by yearsworked;
run;

proc print data=byyears1;
  var yearsworked insuranceid;
  title 'Sort with EQUALS';
run;

proc sort data=insurance out=byyears2 noequals;
  by yearsworked;
run;

proc print data=byyears2;
  var yearsworked insuranceid;
  title 'Sort with NOEQUALS';
run;
```

Program Description

Create the input data set INSURANCE. INSURANCE contains the number of years worked by all insured employees and their insurance IDs.

```
data insurance;
  input YearsWorked 1 InsuranceID 3-5;
  datalines;
5 421
5 336
1 209
1 564
3 711
3 343
4 212
4 616
;
```

Create the output data set BYYEARS1 with the EQUALS option. OUT= creates a new data set for the sorted observations. The EQUALS option maintains the order of the observations relative to each other.

```
proc sort data=insurance out=byyears1 equals;
```

Sort by the first variable. The BY statement specifies that the observations should be ordered numerically by the number of years worked.

```
  by yearsworked;
run;
```

Print the output data set BYYEARS1. PROC PRINT prints the data set BYYEARS1.

```
proc print data=byyears1;
```

Specify the variables to be printed. The VAR statement specifies the variables to be printed and their column order in the output.

```
  var yearsworked insuranceid;
```

Specify the title.

```
  title 'Sort with EQUALS';
run;
```

Create the output data set BYYEARS2. OUT= creates a new data set for the sorted observations. The NOEQUALS option does not maintain the order of the observations relative to each other.

```
proc sort data=insurance out=byyears2 noequal;
```

Sort by the first variable. The BY statement specifies that the observations should be ordered numerically by the number of years worked.

```
  by yearsworked;
run;
```

Print the output data set BYYEARS2. PROC PRINT prints the data set BYYEARS2.

```
proc print data=byyears2;
```

Specify the variables to be printed. The VAR statement specifies the variables to be printed and their column order in the output.

```
  var yearsworked insuranceid;
```

Specify the title.

```
  title 'Sort with NOEQUALS';
run;
```

Output: HTML

Note that sorting with the EQUALS option versus sorting with the NOEQUALS option causes a different sort order for the observations where YearsWorked=3.

Output 55.5 *Sorting with the EQUALS Option*

Sort with EQUALS		
Obs	YearsWorked	InsuranceID
1	1	209
2	1	564
3	3	711
4	3	343
5	4	212
6	4	616
7	5	421
8	5	336

Output 55.6 *Sorting with the NOEQUALS Option*

Sort with NOEQUALS		
Obs	YearsWorked	InsuranceID
1	1	209
2	1	564
3	3	343
4	3	711
5	4	212
6	4	616
7	5	421
8	5	336

Example 4: Retaining the First Observation of Each BY Group

Features:

- PROC SORT statement option
NODUPKEY
- BY statement
- PROC PRINT

Data set: [Account](#)

Note: The EQUALS option must be in effect to ensure that the first observation for each BY group is the one that is retained by the NODUPKEY option. The EQUALS option is the default. If the NOEQUALS option has been specified, then one observation for each BY group is retained by the NODUPKEY option, but not necessarily the first observation.

Details

For this example, the V9 engine is used. The V9 engine provides a consistent ordering where the first observation (the first observation that was written and stored) is generally the first one that is read by PROC SORT. The sorted data set contains only the first observation of each BY group.

Sorting causes observations that have equal BY variable values to be grouped together for output. Normally, PROC SORT writes all observations for each BY group that it assembles to the output data set. The NODUPKEY option instructs the SORT procedure to write only the first observation of each BY group to the output data set and discard any additional observations contained within that BY group. The resulting report in this example contains one observation for each town where the businesses are located.

Program

```
proc sort data=account out=towns nodupkey;
    by town;
run;

proc print data=towns;
    var town company debt accountnumber;
    title 'Towns of Customers with Past-Due Accounts';
run;
```

Program Description

Create the output data set TOWNS but include only the first observation of each BY group. NODUPKEY writes only the first observation of each BY group to the new data set TOWNS.

```
proc sort data=account out=towns nodupkey;
```

Sort by one variable. The BY statement specifies that observations should be ordered by town.

```
    by town;
```

```
run;
```

Print the output data set TOWNS. PROC PRINT prints the data set TOWNS.

```
proc print data=towns;
```

Specify the variables to be printed. The VAR statement specifies the variables to be printed and their column order in the output.

```
var town company debt accountnumber;
```

Specify the title.

```
title 'Towns of Customers with Past-Due Accounts';
run;
```

Output: HTML

The output data set contains only four observations, one for each town in the input data set.

Output 55.7 *Retaining the First Observation of Each BY Group*

Towns of Customers with Past-Due Accounts				
Obs	Town	Company	Debt	AccountNumber
1	Apex	Paul's Pizza	83.00	1019
2	Garner	World Wide Electronics	119.95	1122
3	Holly Springs	Ice Cream Delight	299.98	2310
4	Morrisville	Strickland Industries	657.22	1675

Example 5: Linguistic Sorting Using ALTERNATE_HANDLING=

Features:

- PROC SORT statement option
 - sortseq=linguistic
 - ALTERNATE_HANDLING=SHIFTED
 - STRENGTH=3
- BY statement
- VAR statement
- PROC PRINT
- PROC CONTENTS

Note:

For more information about strengthening the linguistic sort of strings, see [“Example 6: Linguistic Sorting Using ALTERNATE_HANDLING= and STRENGTH=”](#) on page 2093.

Details

In this example, PROC SORT creates an output data set that contains only the first observation of each BY group. You have specified ALTERNATE_HANDLING=SHIFTED because you want “a-b” to sort close to “ab” and “aB”. That is, you do not want “a-b” to appear somewhere far away from “ab” and “aB” by virtue of its hyphen.

Note: In this example, the default STRENGTH for this locale is 3.

Notice how “a-b” and “ab” are treated equivalently in the following example. To order them beyond the first three levels of comparison (alphabetic, diacritic, and case), you can use the fourth level of comparison and specify STRENGTH=4. [“Example 6: Linguistic Sorting Using ALTERNATE_HANDLING= and STRENGTH=”](#) on page 2093 shows how to distinguish the strings further.

PROC CONTENTS shows a Sort Information section in the output. The ICU version is also shown in the sort information.

Program

```
data a;
  length x $ 10;
  x='a-b'; output;
  x='ab'; output;
  x='a-b'; output;
  x='aB'; output;
run;

proc sort data=a sortseq=linguistic( ALTERNATE_HANDLING=SHIFTED );
  by x;
run;

title1 "Linguistic Collation with ALTERNATE_HANDLING=SHIFTED";
proc print data=a;
run;

title1 "Linguistic Collation with ALTERNATE_HANDLING=SHIFTED and BY
Processing";
proc print data=a;
  var x;
  by x;
run;

proc contents data=a;
run;
```

Program Description

Create the data set.

```
data a;
  length x $ 10;
  x='a-b'; output;
  x='ab'; output;
  x='a-b'; output;
  x='aB'; output;
run;
```

Sort the data set using linguistic sorting. Use linguistic sorting and the ALTERNATE_HANDLING=SHIFTED option to sort the data set. Note that the default STRENGTH for this locale is 3. Also use the BY statement to order observations by x.

```
proc sort data=a sortseq=linguistic( ALTERNATE_HANDLING=SHIFTED );
  by x;
run;
```

Print data set A. The TITLE1 statement tells the PRINT procedure the title to use for the output. PROC PRINT then prints data set A.

```
title1 "Linguistic Collation with ALTERNATE_HANDLING=SHIFTED";
proc print data=a;
run;
```

Print data set A using By processing. The TITLE1 statement tells the PRINT procedure the title to use for the output. PROC PRINT then prints data set A using By processing.

```
title1 "Linguistic Collation with ALTERNATE_HANDLING=SHIFTED and BY
Processing";
proc print data=a;
  var x;
  by x;
run;
```

Print the Sort Information when linguistic sorting is being used. The PROC CONTENTS output contains a Sort Information section when PROC SORT is used with linguistic collation. This sort information also includes the ICU version being used.

```
proc contents data=a;
run;
```

Output: HTML

The first PROC PRINT shows that the order of "a-b" and "ab" is not well defined. The second PROC PRINT uses BY processing to show that these values are considered equivalent. ["Example 6: Linguistic Sorting Using ALTERNATE_HANDLING= and STRENGTH=" on page 2093](#) shows how to distinguish the strings more.

Output 55.8 Linguistic Sorting Using the ALTERNATE_HANDLING Option**Linguistic Collation with ALTERNATE_HANDLING=SHIFTED**

Obs	x
1	a-b
2	ab
3	a-b
4	aB

Linguistic Collation with ALTERNATE_HANDLING=SHIFTED and BY Processing

x=a-b

Obs	x
1	a-b
2	ab
3	a-b

x=aB

Obs	x
4	aB

PROC CONTENTS prints out sort information when linguistic sorting is used.
Information about the ICU version that is being used is also provided.

Sort Information	
Sortedby	x
Validated	YES
Character Set	ASCII
Collating Sequence	LINGUISTIC
Locale	en_US
Strength	3
Alternate Handling	SHIFTED
ICU Version Number	63

Example 6: Linguistic Sorting Using ALTERNATE_HANDLING= and STRENGTH=

Features:

- PROC SORT statement option
 - sortseq=linguistic
 - ALTERNATE_HANDLING=SHIFTED
 - STRENGTH=4
- BY statement
- VAR statement
- PROC PRINT

Details

In this example, PROC SORT creates an output data set that contains only the first observation of each BY group. In this example, `ALTERNATE_HANDLING=SHIFTED` is specified because you want "a-b" to sort close to "ab" and "aB" regardless of the hyphen.

Notice how "a-b" and "ab" are treated equivalently in the following example. However, if you want to further distinguish between them and have them appear in two separate BY groups, you must order the strings further. To order them beyond the first three levels of comparison (alphabetic, diacritic, and case), use the fourth level of comparison, `STRENGTH=4`.

Program

```
data a;
  length x $ 10;
  x='a-b'; output;
  x='ab'; output;
  x='a-b'; output;
  x='aB'; output;
run;

proc sort data=a sortseq=linguistic( ALTERNATE_HANDLING=SHIFTED
STRENGTH=4);
  by x;
run;

title1 "Linguistic Collation with STRENGTH=4";
proc print data=a;
run;

Title1 "Linguistic Collation with STRENGTH=4 and BY Processing";
proc print data=a;
  var x;
  by x;
run;
```

Program Description

Create the data set.

```
data a;
  length x $ 10;
  x='a-b'; output;
  x='ab'; output;
  x='a-b'; output;
  x='aB'; output;
```

```
run;
```

Sort the data set using linguistic sorting. Use linguistic sorting and the ALTERNATE_HANDLING=SHIFTED option to sort the data set. Note that the default STRENGTH for this locale is 4. The BY statement specifies that observations should be ordered by x.

```
proc sort data=a sortseq=linguistic( ALTERNATE_HANDLING=SHIFTED
STRENGTH=4);
    by x;
run;
```

Print the output data set A. The TITLE1 statement tells the PRINT procedure the title to use for the output. PROC PRINT then prints data set A.

```
title1 "Linguistic Collation with STRENGTH=4";
proc print data=a;
run;
```

Print the output data set A using By processing. The TITLE statement tells the PRINT procedure what title to use for this output. PROC PRINT then prints data set A using By processing.

```
Title1 "Linguistic Collation with STRENGTH=4 and BY Processing";
proc print data=a;
    var x;
    by x;
run;
```

Output: HTML

The first PROC PRINT shows that the order of "a-b" and "ab" is not well defined. Differentiate between the two by setting STRENGTH=4. The second PROC PRINT uses BY processing to show the order of precedence and how they are differentiated.

Output 55.9 *Linguistic Sorting Using the ALTERNATE_HANDLING and STRENGTH Options*

Linguistic Collation with STRENGTH=4

Obs	x
1	a-b
2	a-b
3	ab
4	aB

Linguistic Collation with STRENGTH=4 and BY Processing**x=a-b**

Obs	x
1	a-b
2	a-b

x=ab

Obs	x
3	ab

x=aB

Obs	x
4	aB

Example 7: Eliminate All Duplicate Observations Using NODUPKEY

Features:

- PROC SORT statement option
NODUPKEY
- OUT=
- BY statement
- PROC PRINT
- KEEP= data set option

Details

In this example, PROC SORT with NODUPKEY creates an output data set that has no duplicate observations. Each of these observations is unique. There is only one observation in the output data set for a given set of variable values. The BY _ALL_ variable sorts by the kept variables (KEEP= option), DIVISION and LEAGUE.

Program

```
proc sort data=sashelp.baseball(keep=division league)out=DL NODUPKEY;
    by _ALL_;
run;
```

```
proc print data=DL;  
title 'Baseball Leagues and Divisions';  
run;
```

Program Description

Processing only the division and league variables from the input data set (KEEP=), create the DL output data set and remove all duplicate entries.

```
proc sort data=sashelp.baseball(keep=division league)out=DL NODUPKEY;
```

Sort by all variables.

```
by _ALL_;  
run;
```

Print the output data set DL.

```
proc print data=DL;  
title 'Baseball Leagues and Divisions';  
run;
```

Output: HTML

The output data set contains only four observations, one for each unique division and league in baseball. No duplicate entries for Division and League are kept.

Output 55.10 Remove All Duplicate Observations from a Data Set

Baseball Leagues and Divisions

Obs	League	Division
1	American	East
2	American	West
3	National	East
4	National	West

SQL Procedure

<i>A Brief Overview</i>	2099
<i>Example: Using the SQL Procedure</i>	2100

A Brief Overview

The SQL procedure is the Base SAS implementation of Structured Query Language. PROC SQL is part of Base SAS software, and you can use it with any SAS data set (table). Often, PROC SQL can be an alternative to other SAS procedures or the DATA step. You can use SAS language elements such as global statements, data set options, functions, informats, and formats with PROC SQL just as you can with other SAS procedures. PROC SQL enables you to perform the following tasks:

- generate reports
- generate summary statistics
- retrieve data from tables or views
- combine data from tables or views
- create tables, views, and indexes
- update the data values in PROC SQL tables
- update and retrieve data from database management system (DBMS) tables
- modify a PROC SQL table by adding, modifying, or dropping columns

For more information, see [SAS SQL Procedure User's Guide](#).

Example: Using the SQL Procedure

The following example selects all rows where Type="Sports" and the MSRP is greater than 100,000 from the Cars data set.

```
proc sql;
  select * from sashelp.cars
    where Type='Sports' and MSRP>100000;
quit;
```

Output 56.1 SQL Procedure Result

Make	Model	Type	Origin	DriveTrain	MSRP	Invoice	Engine Size (L)	Cylinders	Horsepower	MPG (City)	MPG (Highway)	Weight (LBS)	Wheelbase (IN)	Length (IN)
Mercedes-Benz	SL55 AMG 2dr	Sports	Europe	Rear	\$121,770	\$113,388	5.5	8	493	14	21	4235	101	179
Mercedes-Benz	SL600 convertible 2dr	Sports	Europe	Rear	\$126,670	\$117,854	5.5	12	493	13	19	4429	101	179
Porsche	911 GT2 2dr	Sports	Europe	Rear	\$192,465	\$173,560	3.6	6	477	17	24	3131	93	175

SQOOP Procedure

Overview: SQOOP Procedure	2101
What Does the SQOOP Procedure Do?	2101
Syntax: SQOOP Procedure	2102
PROC SQOOP Statement	2102
Usage: SQOOP Procedure	2105
General Usage	2105
Using Workflows	2105
Requirements for Using the SQOOP Procedure	2105
Example: Importing from Teradata to HDFS Using an SQL Query	2107

Overview: SQOOP Procedure

What Does the SQOOP Procedure Do?

You can use Apache Sqoop (pronounced *scoop*) to transfer data between Hadoop and relational database management systems (RDBMSs). You can use the SQOOP procedure to access Apache Sqoop from a SAS session to transfer data between a database and HDFS. It lets you submit Sqoop commands from within your SAS application to your Hadoop cluster.

Sqoop commands are passed to the cluster using the Apache Oozie Workflow Scheduler for Hadoop. PROC SQOOP defines an Oozie workflow for your Sqoop task, which is then submitted to an Oozie server using a RESTful API.

PROC SQOOP works similarly to the Apache Sqoop command-line interface (CLI). Using the same syntax, a user who has licensed SAS/ACCESS Interface to Hadoop can transfer data between a database and HDFS. The user can submit Sqoop CLI

commands in the COMMAND statement for the SQOOP procedure. The procedure provides feedback as to whether the job completed successfully and where to get more details from your Hadoop cluster if the Sqoop task failed.

Some Hadoop distributions support different versions of the Sqoop command. Refer to the documentation for your distribution for specific Sqoop command syntax.

For more information about Apache Sqoop, see the online documentation at <http://sqoop.apache.org> and <http://sqoop.apache.org/docs/1.4.5/index.html>.

For Sqoop considerations and usage, refer to the *Apache Sqoop Cookbook*.

You can find more information about Oozie at <http://oozie.apache.org>.

Syntax: SQOOP Procedure

- Requirements:** To use this procedure requires a separate license to SAS/ACCESS Interface to Hadoop.
- Specify the --username or --password options separately with the DBUSER= and DBPWD= options in the PROC SQOOP statement rather than in the Sqoop command.
- Supports:** Kerberos only on Linux
- Notes:** By default, the SQOOP procedure uses the hive-site.xml file to locate the Zookeeper service discovery. If the Zookeeper service discovery is not available, in the SQOOP procedure you can set a server name and port with the HIVE_SERVER= option or a JDBC URI with the HIVE_URI= option.
- Support for HIVE_SERVER= and HIVE_URI= options was added in [SAS Viya 3.5](#)
- See:** [SERVER= LIBNAME option](#), [URI= LIBNAME option](#)
- Example:** [Importing from Teradata to HDFS Using an SQL Query](#)
- PROC SQOOP** *required-arguments* <*additional-sqoop-options*>;

Statement	Task	Example
PROC SQOOP	Allow access to Apache Sqoop for data transfer	Ex. 1

PROC SQOOP Statement

Allows access to Apache Sqoop using options to allow data transfer between a database and HDFS.

Syntax

PROC SQOOP

```

COMMAND='command-to-sqoop'
DBPWD='database-password'
DBUSER='database-user-name'
<DELETEWF>
HADOOPUSER='hadoop-user-name'
HADOOPPWD='hadoop-password'
<HIVE_SERVER='server-name-and-port'>
<HIVE_URI='JDBC-URI'>
<JOBTRACKER='job-tracker-URL'>
<NAMENODE='name-node-URL'>
OOZIEURL='oozie-URL'
<PASSWORDFILE='password-file'>
<WFHDFS_PATH='Oozie-workflow-path'>;
run;

```

Required Arguments

COMMAND='command-to-sqoop'

specifies the Apache Sqoop command. Here is how you must specify the command:

```
SQOOP-command --option ... --option
```

Restriction Do not use the `sqoop` invocation command.

Note The escape character, a forward slash (\), in \$CONDITIONS is not required when you submit Sqoop commands using the SQOOP procedure.

Tip For more information about Apache Sqoop commands, see the online documentation at <http://sqoop.apache.org>.

DBPWD='database-password'

specifies the database password that is associated with the DBUSER option. Because this option is mutually exclusive with the PASSWORDFILE= option, you must specify only one or the other.

DBUSER='database-user-name'

specifies the database user name to use for import or export.

HADOOPPWD='hadoop-password'

specifies the Hadoop password that is associated with the HADOOPUSER= option.

Restriction Do not provide this option when connecting to a cluster that is enabled for Kerberos.

HADOOPUSER='hadoop-user-name'

specifies the Hadoop user name to use for import or export.

Restriction Do not provide this option when connecting to a cluster that is enabled for Kerberos.

OOZIEURL='oozie-URL'

specifies the URL to the Oozie server.

Optional Arguments

DELETEWF

specifies that, if an Oozie workflow file exists as specified by the location in WFHDFS_PATH, it should be deleted. The SQOOP procedure then creates a new workflow file at that location.

HIVE_SERVER='server-name-and-port'

specifies the Hive server name that runs the Hive service and the port number to use to connect to the specified Hive service.

Restriction You can specify only one option at a time: HIVE_SERVER= or HIVE_URI=.

Requirements If the server name contains spaces or nonalphanumeric characters, you must enclose it in quotation marks.

This option is required when connecting to a Kerberized HDP 3.1 cluster when the hive-site.xml file does not contain the Zookeeper service.

HIVE_URI=jdbc:hive2://

HiveServerHost:HiveServerPort</Schema:Property1=Value; <... PropertyN=ValueN;>>

specifies the JDBC URI.

Restriction You can specify only one option at a time: HIVE_SERVER= or HIVE_URI=.

Requirement This option is required when connecting to a Kerberized HDP 3.1 cluster when the hive-site.xml file does not contain the Zookeeper service.

Example jdbc:hive2://MyHiveServer:10000

JOBTRACKER='job-tracker-URL'

specifies the URL to the JobTracker or ResourceManager services.

Default SAS_HADOOP_CONFIG_PATH determines the value for this option if you do not specify one.

NAMENODE='name-node-URL'

specifies the URL to the NameNode services.

Default SAS_HADOOP_CONFIG_PATH determines the value for this option if you do not specify one.

PASSWORDFILE='password-file'

specifies the name of the file that is located in HDFS that contains the database password for import or export. This separate password file must exist before you can run this procedure, and care should be taken to keep this file secure.

WFHDFSPATH='Oozie-workflow-path-and-filename'

specifies the path and filename for where to upload the Oozie workflow, as shown below.

```
/user/myID/mydir/myfile.xml
```

Default A generated path is used if you do not provide a value.

Tip You can use hdfs://server syntax, although it is not required.

Usage: SQOOP Procedure

General Usage

Because the Oracle JDBC Connector requires it, you must specify the value to be used for the --table option in Sqoop in uppercase letters. For details about case sensitivity for tables, see the documentation for your specific DBMS.

Connection strings should include the character set option that is appropriate for the data to be imported. For details, refer to your connector documentation.

Using Workflows

Workflows are created only when they are required. Some SQOOP jobs can use Oozie proxy submission, which generates no workflow file. Proxy submission is selected if you are running Oozie 4.1 or later and are not using a Hive table as the destination. The SAS log contains a note if PROC SQOOP uses proxy submission.

Requirements for Using the SQOOP Procedure

Here is what you need before you can begin using the SQOOP procedure.

What You Need	PROC SQOOP Option	Where to Find It
database connector	COMMAND: Sqoop syntax requires a --connection option that is specific to your database.	Refer to the documentation for your Hadoop distribution.
database user ID	DBUSER	Contact your database administrator.
database password	DBPWD	
HDFS file that contains the database password	PASSWORDFILE	Contact your database administrator and your Hadoop cluster administrator.
Hadoop user ID	HADOOPUSER	Contact your Hadoop cluster administrator.
Hadoop password	HADOOPPWD	
Oozie URL	OOZIEURL	
name node	NAMENODE	
Job Tracker (MR1) or Resource Manager (MR2)	JOBTRACKER	
Oozie workflow output path	WFHDFSPATH	Contact your Hadoop cluster administrator. Generally, the Oozie workflow can be written to your user directory in HDFS.
Sqoop Command	COMMN	Refer to the Sqoop documentation for your Hadoop distribution.

Example: Importing from Teradata to HDFS Using an SQL Query

In this example, DELETEWF is included to replace an existing workflow with a new workflow for this task.

Note: For proper default NAMENODE and JOBTRACKER port values for your environment, check the configuration for your particular distribution or refer to your Hadoop documentation.

```
proc sqoop dbuser='mydbusr1' dbpwd='mydbpwd1'
  hadoopuser='sashdpusr1' hadooppwd='sashdppwd1'
  oozieurl='http://myoozie-04:55000/oozie'
  namenode='hdfs://myoozie-04.unx.srvr.com:8020'
  jobtracker='myoozie-04.unx.srvr.com:8032'
  wfhdfspath='hdfs://myoozie-04.unx.srvr.com:8020/user/mydbusr1/
myworkflow.xml'
  deletewf
  command='import
    --connection-manager
com.cloudera.connector.teradata.TeradataManager
    --connect jdbc:teradata://myconnecti/Database=sqoop
    --query "SELECT * FROM sales where ($CONDITIONS and I < 25)" -m 1
    --split-by i --delete-target-dir
    --target-dir /user/mydbusr1/sales2';
run;
```


STANDARD Procedure

Overview: STANDARD Procedure	2109
What Does the STANDARD Procedure Do?	2110
Standardizing Data	2110
Syntax: STANDARD Procedure	2112
PROC STANDARD Statement	2113
ATTRIB Statement	2116
BY Statement	2117
FORMAT Statement	2121
FREQ Statement	2121
LABEL Statement	2124
VAR Statement	2125
WEIGHT Statement	2126
WHERE Statement	2127
Usage: STANDARD Procedure	2129
Statistical Computations: STANDARD Procedure	2129
Results: STANDARD Procedure	2130
Missing Values	2130
Output Data Set	2130
Examples: STANDARD Procedure	2131
Example 1: Standardizing to a Given Mean and Standard Deviation	2131
Example 2: Standardizing BY Groups and Replacing Missing Values	2134

Overview: STANDARD Procedure

What Does the STANDARD Procedure Do?

The STANDARD procedure standardizes variables in a SAS data set to a given mean and standard deviation, and it creates a new SAS data set containing the standardized values.

Standardizing Data

The following output shows a simple standardization where the output data set contains standardized student exam scores. The statements that produce the output follow:

```
proc standard data=score mean=75 std=5
              out=stndtest;
run;

proc print data=stndtest;
run;
```

Output 58.1 Standardized Test Scores Using PROC STANDARD

The SAS System			1
Obs	Student	Test1	
1	Capalleti	80.5388	
2	Dubose	64.3918	
3	Engles	80.9143	
4	Grant	68.8980	
5	Krupski	75.2816	
6	Lundsford	79.7877	
7	McBane	73.4041	
8	Mullen	78.6612	
9	Nguyen	74.9061	
10	Patel	71.9020	
11	Si	73.4041	
12	Tanaka	77.9102	

The following output shows a more complex example that uses BY-group processing. PROC STANDARD computes Z scores separately for two BY groups by standardizing life-expectancy data to a mean of 0 and a standard deviation of 1.

The data are 1950 and 1993 life expectancies at birth for 16 countries. The birth rates for each country, classified as stable or rapid, form the two BY groups. The statements that produce the analysis also do the following:

- print statistics for each variable to standardize
- replace missing values with the given mean
- calculate standardized values using a given mean and standard deviation
- print the data set with the standardized values

For an explanation of the program that produces this output, see [“Example 2: Standardizing BY Groups and Replacing Missing Values”](#) on page 2134.

Output 58.2 Z Scores for Each BY Group Using PROC STANDARD

Life Expectancies by Birth Rate				2
----- PopulationRate=Stable -----				
The STANDARD Procedure				
Name Label	Mean	Standard Deviation	N	
Life50 1950 life expectancy	67.400000	1.854724	5	
Life93 1993 life expectancy	74.500000	4.888763	6	
----- PopulationRate=Rapid -----				
Name Label	Mean	Standard Deviation	N	
Life50 1950 life expectancy	42.000000	5.033223	8	
Life93 1993 life expectancy	59.100000	8.225300	10	

Standardized Life Expectancies at Birth by a Country's Birth Rate				3
Population Rate	Country	Life50	Life93	
Stable	France	-0.21567	0.51138	
Stable	Germany	0.32350	0.10228	
Stable	Japan	-1.83316	0.92048	
Stable	Russia	0.00000	-1.94323	
Stable	United Kingdom	0.86266	0.30683	
Stable	United States	0.86266	0.10228	
Rapid	Bangladesh	0.00000	-0.74161	
Rapid	Brazil	1.78812	0.96045	
Rapid	China	-0.19868	1.32518	
Rapid	Egypt	0.00000	0.10942	
Rapid	Ethiopia	-1.78812	-1.59265	
Rapid	India	-0.59604	-0.01216	
Rapid	Indonesia	-0.79472	-0.01216	
Rapid	Mozambique	0.00000	-1.47107	
Rapid	Philippines	1.19208	0.59572	
Rapid	Turkey	0.39736	0.83888	

Syntax: STANDARD Procedure

Tip:

You can use the ATTRIB, FORMAT, LABEL, and WHERE statements. For more information, see “Statements with the Same Function in Multiple Procedures” on page 23.

PROC STANDARD <options>;

ATTRIB variable-list(s) attribute-list(s);

BY <DESCENDING> variable-1

<... <DESCENDING> variable-n>

<NOTSORTED>;

FREQ variable;

FORMAT variable-1 <...variable-n> <format> <DEFAULT=default-format>;

LABEL variable-1=label-1<...variable-n=label-n>;

VAR variable(s);

WEIGHT variable;

WHERE where-expression-1

< logical-operator where-expression-n>;

Statement	Task	Example
PROC STANDARD	Standardize variables to a given mean and standard deviation	Ex. 1, Ex. 2

Statement	Task	Example
BY	Calculate separate standardized values for each BY group	Ex. 2
BY	Identify a variable whose values represent the frequency of each observation	
VAR	Select the variables to standardize and determine the order in which they appear in the printed output	Ex. 1
WEIGHT	Identify a variable whose values weight each observation in the statistical calculations	

PROC STANDARD Statement

Standardizes variables in a SAS data set to a given mean and standard deviation, and it creates a new SAS data set containing the standardized values.

Examples: [“Example 2: Standardizing BY Groups and Replacing Missing Values” on page 2134](#)
 [“Example 1: Standardizing to a Given Mean and Standard Deviation” on page 2131](#)

Syntax

PROC STANDARD *<options>*;

Summary of Optional Arguments

DATA=*SAS-data-set*

specifies the input data set.

EXCLNPWGT

excludes observations with nonpositive weights.

MEAN=*mean-value*

specifies the mean value.

OUT=*SAS-data-set*

specifies the output data set.

REPLACE

replace missing values with a variable mean or MEAN= value.

STD=*std-value*

specifies the standard deviation value.

VARDEF=*divisor*

specifies the divisor for variance calculations and standard deviation. The following are choices are possible values for *divisor* and the associated divisors.

Control printed output

NOPRINT

suppresses all printed output.

PRINT

prints statistics for each variable to standardize.

Preserve values

PRESERVEBYVALUES

preserves raw by values.

Without Arguments

If you do not specify **MEAN=**, **REPLACE**, or **STD=**, the output data set is an identical copy of the input data set.

Optional Arguments

DATA=SAS-data-set

specifies the input SAS data set.

Restriction You cannot use PROC STANDARD with an engine that supports concurrent access if another user is updating the data set at the same time.

See [“Input Data Sets” on page 24](#)

EXCLNPWGT

excludes observations with nonpositive weight values (zero or negative). The procedure does not use the observation to calculate the mean and standard deviation, but the observation is still standardized. By default, the procedure treats observations with negative weights like those with zero weights and counts them in the total number of observations.

Alias EXCLNPWGTS

MEAN=mean-value

standardizes variables to a mean of *mean-value*.

Alias M=

Default mean of the input values

Example [“Example 1: Standardizing to a Given Mean and Standard Deviation” on page 2131](#)

NOPRINT

suppresses the printing of the procedure output. NOPRINT is the default value.

OUT=SAS-*data-set*

specifies the output data set. If SAS-*data-set* does not exist, PROC STANDARD creates it. If you omit OUT=, the data set is named *Datan*, where *n* is the smallest integer that makes the name unique.

Default *Datan*

Example [“Example 1: Standardizing to a Given Mean and Standard Deviation” on page 2131](#)

PRESERVEBYVALUES

preserves raw by values. of all BY variables when those variables are propagated to the output data set.

PRINT

prints the original frequency, mean, and standard deviation for each variable to standardize.

Example [“Example 2: Standardizing BY Groups and Replacing Missing Values” on page 2134](#)

REPLACE

replaces missing values with the variable mean.

Interaction If you use MEAN=, PROC STANDARD replaces missing values with the given mean.

Example [“Example 2: Standardizing BY Groups and Replacing Missing Values” on page 2134](#)

STD=std-value

standardizes variables to a standard deviation of *std-value*.

Alias S=

Default standard deviation of the input values

Example [“Example 1: Standardizing to a Given Mean and Standard Deviation” on page 2131](#)

VARDEF=divisor

specifies the divisor for variance calculations and standard deviation. The following are choices are possible values for *divisor* and the associated divisors.

DF

degrees of freedom. The formula for divisor is $n - 1$.

N

number of observations. The formula for divisor is n .

WDF

sum of weights minus one. The formula for divisor is $(\sum_i w_i) - 1$.

WEIGHT

sum of weights. The formula for divisor is $\sum_i w_i$.

Alias WGT

The procedure computes the variance as $CSS/divisor$, where CSS is the corrected sums of squares and equals $\sum (x_i - \bar{x})^2$. When you weight the analysis variables, CSS equals $\sum w_i(x_i - \bar{x}_w)^2$ where $\bar{x}_w$ is the weighted mean.

Default DF

Tips When you use the WEIGHT statement and VARDEF=DF, the variance is an estimate of σ^2 , where the variance of the i th observation is $var(x_i) = \sigma^2/w_i$ and w_i is the weight for the i th observation. This yields an estimate of the variance of an observation with unit weight.

When you use the WEIGHT statement and VARDEF=WGT, the computed variance is asymptotically (for large n) an estimate of $\sigma^2/\bar{w}$, where $\bar{w}$ is the average weight. This yields an asymptotic estimate of the variance of an observation with average weight.

See [“Keywords and Formulas” on page 2426](#)

ATTRIB Statement

Associates a format, informat, label, and length with one or more variables.

Syntax

ATTRIB *variable-list(s) attribute-list(s);*

Arguments

variable-list(s)

names the variables that you want to associate with the attributes.

TIP List the variables in any form that SAS allows.

attribute-list(s)

specifies one or more attributes to assign to *variable-list*. Specify one or more of these attributes in the ATTRIB statement:

FORMAT=format

associates a format with variables in *variable-list*.

Tip The format can be either a standard SAS format or a format that is defined with the FORMAT procedure.

INFORMAT=informal

associates an informat with variables in *variable-list*.

Tip The informat can be either a standard SAS informat or an informat that is defined with the FORMAT procedure.

LABEL='label'

associates a label with variables in *variable-list*.

See Also

[“ATTRIB Statement” in SAS DATA Step Statements: Reference.](#)

BY Statement

Orders the output according to the BY groups.

Restriction: Do not specify a group variable in the BY statement. The BY variable cannot be the same as the group variable. A *group variable* is a variable in the input data set that is used to categorize chart variable values into groups. For more information, see [“Group Variables” on page 1760](#).

Syntax

```
BY <DESCENDING> variable-1
    <... <DESCENDING> variable-n>
    <NOTSORTED>;
```

Required Argument

variable

specifies the variable that the procedure uses to form BY groups. You can specify more than one variable. If you do not use the NOTSORTED option in the BY statement, then either the observations in the data set must be sorted by all the variables that you specify, or they must be indexed appropriately. Variables in a BY statement are called *BY variables*.

Optional Arguments

DESCENDING

specifies that the observations are sorted in descending order by the variable that immediately follows the word DESCENDING in the BY statement.

NOTSORTED

specifies that observations are not necessarily sorted in alphabetic or numeric order. The observations are grouped in another way (for example, chronological order).

The requirement for ordering or indexing observations according to the values of BY variables is suspended for BY-group processing when you use the NOTSORTED option. In fact, the procedure does not use an index if you specify NOTSORTED. The procedure defines a BY group as a set of contiguous observations that have the same values for all BY variables. If observations with the same values for the BY variables are not contiguous, then the procedure treats each contiguous set as a separate BY group.

You cannot use the NOTSORTED option in a PROC SORT step.

Note: You cannot use the GROUPFORMAT option, which is available in the BY statement in a DATA step, in a BY statement in any PROC step.

Details

BY-Group Processing

Procedures create output for each BY group. For example, the elementary statistics procedures and the scoring procedures perform separate analyses for each BY group. The reporting and ODS procedures produce a report for each BY group.

Note: All Base SAS procedures except PROC PRINT process BY groups independently. PROC PRINT can report the number of observations in each BY group as well as the number of observations in all BY groups. Similarly, PROC PRINT can sum numeric variables in each BY group and across all BY groups.

You can use only one BY statement in each PROC step. When you use a BY statement, the procedure expects an input data set that is sorted by the order of the BY variables or one that has an appropriate index. If your input data set does not meet these criteria, then an error occurs. Either sort it with the SORT procedure or create an appropriate index on the BY variables.

Depending on the order of your data, you might need to use the NOTSORTED or DESCENDING option in the BY statement in the PROC step.

- For more information about the BY statement, see [SAS DATA Step Statements: Reference](#).
- For more information about PROC SORT, see [Chapter 55, “SORT Procedure,” on page 2039](#).
- For more information about BY-group processing in CAS, see [“Grouping and Ordering” in SAS Cloud Analytic Services: DATA Step Programming](#).
- For more information about creating indexes, see [“INDEX CREATE Statement” on page 440](#).

Formatting BY-Variable Values

When a procedure is submitted with a BY statement, the following actions are taken with respect to processing of BY groups:

- 1 The procedure determines whether the data is sorted by the internal (unformatted) values of the BY variable(s).
- 2 The procedure determines whether a format has been applied to the BY variable(s). If the BY variable is numeric and has no user-applied format, then the BEST12. format is applied for the purpose of BY-group processing.
- 3 The procedure continues adding observations to the current BY group until both the internal and formatted values of the BY variable or variables change.

This process can have unexpected results if, for example, nonconsecutive internal BY values share the same formatted value. In this case, the formatted value is represented in different BY groups. Alternatively, if different consecutive internal BY values share the same formatted value, then these observations are grouped into the same BY group.

BY Variables That Have Different Lengths in Two Data Sets

When a procedure has a BY statement and two input data sets, one of the input data sets is called the primary data set and the other is called the secondary data set. The primary data set is usually, but not always, the DATA= data set. A BY statement always applies to the primary data set. The variables in the BY statement must appear in the primary data set.

Each procedure determines whether a BY statement applies to a secondary data set, and performs one of the following actions:

- The procedure might always apply the BY statement to the secondary data set. In this case, one or more variables in the BY statement must appear in the secondary data set.
- The procedure might never apply the BY statement to the secondary data set. In this case, the variables in the BY statement are not required in the secondary data set.
- The procedure might check whether the BY variables are in the secondary data set. If none of the BY variables are in the secondary data set, then BY processing does not occur for the secondary data set. If one or more of the BY variables are in the secondary data set, and they match the BY variables in the primary data set, then BY processing is done for the secondary data set. If some but not all BY variables are in the secondary data set, then the procedure might issue an error message and quit. Or, it might take some other action described in the documentation for that particular procedure.

If the BY statement is applied to the secondary data set, then each BY variable that exists on both the data sets must have the same type, character or numeric, in both data sets. The BY variables are required to have either the same formatted value or the same unformatted value. Formatted values match only if both the formatted lengths and the formatted values are the same. Unformatted values are not required to have the same length in order to match. The unformatted character

values match if the unformatted values are the same after stripping the trailing blanks. The unformatted doubles match if they have the same value.

A secondary data set does not need to have all of the BY variables that are in the primary data set. A procedure can define a subset of the BY variables for the secondary data set. For example, if the primary data set has the BY variables A,B,C,D, then the procedure can define the following BY variables on the secondary data set:

- A
- A,B
- A,B,C
- A,B,C,D

If both the primary and secondary data sets have the same number of BY variables, and all the BY variables have the same byte lengths and format lengths, then either the unformatted values or the formatted values in the BY buffer (for all of the BY variables) have to match. If they do not match, then each variable is compared. The formatted values of each variable are compared first. The formatted lengths have to match, and the formatted values have to match. If the formatted lengths and values do not match, then the unformatted values are compared even if the byte lengths are different.

If corresponding character variable lengths differ, then the longer character variable can contain only trailing blanks for the extra characters. If the lengths of the character variables are different, then the values match as long as they are the same after stripping the trailing blanks. For example, 'ABCD' in the primary data set matches 'ABCD ' in the secondary data set. If the secondary data set contained 'ABCDEF', then they would not match.

Base SAS Procedures That Support the BY Statement

COMPARE	SORT (required)
CORR	STANDARD
FREQ	SUMMARY
MEANS	TABULATE
PRINT	TRANSPOSE
RANK	UNIVARIATE
REPORT (nonwindowing environment only)	ODSTEXT
ODSLIST	ODSTABLE

Note: In the SORT procedure, the BY statement specifies how to sort the data. In the other procedures, the BY statement specifies how the data is currently sorted.

See Also

[“BY Statement” in SAS DATA Step Statements: Reference](#)

FORMAT Statement

Associates formats with variables.

Syntax

FORMAT *variable-1* <...*variable-n*> *format variable-1* <...*variable-n*> *format*;

Arguments

variable

names one or more variables for SAS to associate with a format. You must specify at least one *variable*.

Tip To disassociate a format from a variable, use the variable in a FORMAT statement without specifying a format in a DATA step or in PROC DATASETS. In a DATA step, place this FORMAT statement after the SET statement. See [“Removing a Format” in SAS DATA Step Statements: Reference](#). You can also use PROC DATASETS.

format

specifies the format that is listed for writing the values of the variables.

Tip Formats that are associated with variables by using a FORMAT statement behave like formats that are used with a colon modifier in a subsequent PUT statement. For more information about using a colon modifier, see [“PUT Statement: List” in SAS DATA Step Statements: Reference](#).

See [SAS Formats and Informats: Reference](#)

FREQ Statement

Treats observations as if they appear multiple times in the input data set. You can use a WEIGHT statement and a FREQ statement in the same step of any procedure that supports both statements.

Syntax

FREQ *variable*;

Required Argument

variable

specifies a numeric variable whose value represents the frequency of the observation. If you use the FREQ statement, then the procedure assumes that each observation represents

n observations, where n is the value of *variable*. If *variable* is not an integer, then SAS truncates it. If *variable* is less than 1 or is missing, then the procedure does not use that observation to calculate statistics. If a FREQ statement does not appear, then each observation has a default frequency of 1.

The sum of the frequency variable represents the total number of observations.

Examples

Example 1: Working with Frequencies

Procedures That Support the FREQ Statement

- CORR
- MEANS or SUMMARY
- REPORT
- STANDARD
- TABULATE
- UNIVARIATE

Use a Frequency Variable in PROC MEANS

The data in this example represents a ship's course and speed (in nautical miles per hour), recorded every hour. The frequency variable Hours represents the number of hours that the ship maintained the same course and speed. Each of the following PROC MEANS steps calculates average course and speed. The different results demonstrate the effect of using Hours as a frequency variable. The first PROC MEANS step does not use a frequency variable. Without a frequency variable, each observation has a frequency of 1, and the total number of observations is 8. The second PROC MEANS step uses Hours as a frequency variable. When you use Hours as a frequency variable, the frequency of each observation is the value of

Hours. The total number of observations is 141 (the sum of the values of the frequency variable).

```
options nodate pageno=1 linesize=64 pagesize=40;

data track;
  input Course Speed Hours @@;
  datalines;
30 4 8 50 7 20
75 10 30 30 8 10
80 9 22 20 8 25
83 11 6 20 6 20
;

proc means data=track maxdec=2 n mean;
  var course speed;
  title 'Average Course and Speed';
run;

proc means data=track maxdec=2 n mean;
  var course speed;
  freq hours;
  title 'Average Course and Speed';
run;
```

Example 2: Average Course and Speed With and Without a Frequency Variable

Output

Average Course and Speed

The MEANS Procedure

Variable	N	Mean
Course	8	48.50
Speed	8	7.88

Average Course and Speed

The MEANS Procedure

Variable	N	Mean
Course	141	49.28
Speed	141	8.06

LABEL Statement

Assigns descriptive labels to variables.

Syntax

LABEL *variable-1* = '*text-string*' <...*variable-n* = '*text-string*'>;

LABEL *variable-1* = '' < ...*variable-n*='' >; | *variable-1* = <...*variable-n*= >;

Arguments

variable

specifies the variable that you want to label. You can specify labels for multiple variables in a single LABEL statement:

```
data test;
  L = 5;
  W = 10;
  label L = 'Length' W = 'Width';
run;
```

text-string

specifies a label of up to 256 bytes.

```
data test;
  L = 5;
  label L = 'Length';
run;
```

Restrictions If the label includes a semicolon (;) or an equal sign (=), you must enclose the label in either single or double quotation marks.

```
label N = 'Number = ';
```

If the label includes single quotation marks ('), you must enclose the label in double quotation marks.

```
label Name = "Owner's Last Name";
```

There is partial support for variable name substitution with the JAVAIMG device. When the label text for the variable exceeds the space allowed, the variable name is used instead by procedures that are labeling an axis or a legend title. With the exception of the SAS/GRAPH GPLOT procedure, the variable name is not used when the JAVAIMG device is specified.

Notes Quotation marks are not required unless the label includes a semicolon, quotation marks, or an equal sign.

The LABEL statement expects an equal sign (=) after the first variable. If any other character is found, an error occurs. The LABEL statement considers everything after the equal sign, following the first variable, as part of the label until the statement comes to another variable followed by an equal sign, regardless of quotation marks. In this example, the label for x is **this is x y 4 = this is y** because the next variable that is followed by an equal sign is z.

```
data test;
  x = 1;
  y = 1;
  z = 1;
  label x = 'this is x' y 4 = 'this is y' z = 'This is z';
run;
```

In this LABEL statement, the x variable has the same label:

```
label x = this is x y 4 = this is y z = This is z;
```

See

[“Character Constants and Character Variables” in SAS Programmer’s Guide: Essentials.](#)

”

removes a label from a variable. Enclose a single blank space in quotation marks to remove an existing label. You can remove labels from multiple variables in a single LABEL statement:

```
data test;
  L=5; W=10;
  label L = ' ' W = ' ';
run;
```

Alternatively, you can remove a label by specifying nothing after the equal sign:

```
data test;
  L=5; W=10;
  label L = W = ;
run;
```

VAR Statement

Specifies the variables to standardize and their order in the printed output.

Aliases: VARIABLE
 VARIABLES

Default: If you omit the VAR statement, PROC STANDARD standardizes all numeric variables not listed in the other statements.

Example: [“Example 1: Standardizing to a Given Mean and Standard Deviation” on page 2131](#)

Syntax

VAR *variable(s)*;

Required Argument

variable(s)

identifies one or more variables to standardize.

WEIGHT Statement

Specifies weights for analysis variables in the statistical calculations.

Alias: WGT

See: For information about calculating weighted statistics and for an example that uses the WEIGHT statement, see [“Calculating Weighted Statistics” on page 2198](#).

Syntax

WEIGHT *variable*;

Required Argument

variable

specifies a numeric variable whose values weight the values of the analysis variables. The values of the variable do not have to be integers. The table below shows what the action will be based on the weight value.

Table 58.1 *WEIGHT Statement Value and PROC STANDARD Action*

Weight Value	PROC STANDARD Action
0	Counts the observation in the total number of observations
Less than 0	Converts the weight value to zero and counts the observation in the total number of observations
Missing	Excludes the observation from the calculation of mean and standard deviation

To exclude observations that contain negative and zero weights from the calculation of mean and standard deviation, use EXCLNPWGT. Note that most SAS/STAT procedures, such as PROC GLM, exclude negative and zero weights by default.

Tip When you use the WEIGHT statement, consider which value of the VARDEF= option is appropriate. For more information, see [“VARDEF=divisor” on page 2115](#) and the calculation of weighted statistics in [“Keywords and Formulas” on page 2426](#).

Details

Note: Prior to Version 7 of SAS, the procedure did not exclude the observations with missing weights from the count of observations.

WHERE Statement

Subsets the input data set by specifying certain conditions that each observation must meet before it is available for processing.

Syntax

WHERE *where-expression-1*
 < *logical-operator where-expression-n* >;

Arguments

where-expression

is an arithmetic or logical expression that generally consists of a sequence of operands and operators.

Tips The operands and operators described in the next several sections are also valid for the WHERE= data set option.

You can specify multiple where-expressions.

logical-operator

can be AND, AND NOT, OR, or OR NOT.

Example

Procedures That Support the WHERE Statement

You can use the WHERE statement with any of the following Base SAS procedures that read a SAS data set:

COMPARE	REPORT
CORR	SORT
DATASETS (APPEND statement)	SQL
FREQ	STANDARD
MEANS or SUMMARY	TABULATE
PRINT	TRANSPOSE
RANK	UNIVARIATE

Details

- The COMPARE procedure and the APPEND statement in PROC DATASETS accept more than one input data set. For more information, see the documentation for the specific procedure.
- To subset the output data set, use the WHERE= data set option:

```
proc report data=debate nowd
           out=onlyfr (where=(year='1')) ;
run;
```

For more information about WHERE=, see [SAS DATA Step Statements: Reference](#).

Temporarily Dissociating a Format from a Variable

In this example, PROC PRINT prints only those observations that meet the condition of the WHERE expression.

```
options nodate pageno=1 linesize=64
      pagesize=40;

proc print data=debate noobs;
  where gpa>3.5;
  title 'Team Members with a GPA';
  title2 'Greater than 3.5';
run;
```

Output

Team Members with a GPA Greater than 3.5			
Name	Gender	Year	GPA
Capiccio	m	Freshman	3.598
Tucker	m	Freshman	3.901
Bagwell	f	Sophomore	3.722
Gold	f	Junior	3.609
Syme	f	Junior	3.883
Baglione	f	Senior	4.000
Carr	m	Senior	3.750
Hall	m	Senior	3.574

Usage: STANDARD Procedure

Statistical Computations: STANDARD Procedure

Standardizing values removes the location and scale attributes from a set of data. The formula to compute standardized values is

$$x'_i = \frac{S^*(x_i - \bar{x})}{s_x} + M$$

where

x'_i

is a new standardized value.

 S

is the value of STD=.

 M

is the value of MEAN=.

 x_i

is an observation's value.

 $\bar{x}$

is a variable's mean.

 s_x

is a variable's standard deviation.

PROC STANDARD calculates the mean ($\bar{x}$) and standard deviation (s_x) from the input data set. The resulting standardized variable has a mean of **M** and a standard deviation of **S**.

If the data are normally distributed, standardizing is also studentizing since the resulting data have a Student's **t** distribution.

Results: STANDARD Procedure

Missing Values

By default, PROC STANDARD excludes missing values for the analysis variables from the standardization process, and the values remain missing in the output data set. When you specify the REPLACE option, the procedure replaces missing values with the variable's mean or the MEAN= value.

If the value of the WEIGHT variable or the FREQ variable is missing, then the procedure does not use the observation to calculate the mean and the standard deviation. However, the observation is standardized.

Output Data Set

PROC STANDARD always creates an output data set that stores the standardized values in the VAR statement variables, regardless of whether you specify the OUT= option. The output data set contains all the input data set variables, including those

not standardized. PROC STANDARD does not print the output data set. Use PROC PRINT, PROC REPORT, or another SAS reporting tool to print the output data set.

Examples: STANDARD Procedure

Example 1: Standardizing to a Given Mean and Standard Deviation

Features:

- PROC STANDARD statement options
 - MEAN=
 - OUT=
 - STD=
- VAR statement
- PRINT procedure

Details

This example does the following:

- standardizes two variables to a mean of 75 and a standard deviation of 5
 - specifies the output data set
 - combines standardized variables with original variables
 - prints the output data set
-

Program

```
options nodate pageno=1 linesize=80 pagesize=60;
data score;
  length Student $ 9;
  input Student $ StudentNumber Section $
        Test1 Test2 Final @@;
  format studentnumber z4.;
  datalines;
Capalleti 0545 1 94 91 87  Dubose      1252 2 51 65 91
Engles    1167 1 95 97 97  Grant       1230 2 63 75 80
Krupski   2527 2 80 69 71  Lundsford 4860 1 92 40 86
```

```

McBane      0674 1 75 78 72  Mullen      6445 2 89 82 93
Nguyen      0886 1 79 76 80  Patel       9164 2 71 77 83
Si          4915 1 75 71 73  Tanaka      8534 2 87 73 76
;

proc standard data=score mean=75 std=5 out=stdntest;

    var test1 test2;
run;

proc sql;
    create table combined as
    select old.student, old.studentnumber,
           old.section,
           old.test1, new.test1 as StdTest1,
           old.test2, new.test2 as StdTest2,
           old.final
    from score as old, stdntest as new
    where old.student=new.student;

proc print data=combined noobs round;
    title 'Standardized Test Scores for a College Course';
run;

```

Program Description

Set the SAS system options. The NODATE option specifies to omit the date and time at which the SAS job began. The PAGENO= option specifies the page number for the next page of output that SAS produces. The LINESIZE= option specifies the line size. The PAGESIZE= option specifies the number of lines for a page of SAS output.

```
options nodate pageno=1 linesize=80 pagesize=60;
```

Create the Score data set. This data set contains test scores for students who took two tests and a final exam. The FORMAT statement assigns the *Zw.d* format to StudentNumber. This format pads right-justified output with Os instead of blanks. The LENGTH statement specifies the number of bytes to use to store values of Student.

```

data score;
    length Student $ 9;
    input Student $ StudentNumber Section $
           Test1 Test2 Final @@;
    format studentnumber z4.;
    datalines;
Capalleti 0545 1 94 91 87  Dubose      1252 2 51 65 91
Engles    1167 1 95 97 97  Grant       1230 2 63 75 80
Krupski   2527 2 80 69 71  Lundsford 4860 1 92 40 86
McBane    0674 1 75 78 72  Mullen      6445 2 89 82 93
Nguyen    0886 1 79 76 80  Patel       9164 2 71 77 83
Si        4915 1 75 71 73  Tanaka      8534 2 87 73 76
;

```

Generate the standardized data and create the Stdntest output data set. PROC STANDARD uses a mean of 75 and a standard deviation of 5 to standardize the values. OUT= identifies Stdntest as the data set to contain the standardized values.

```
proc standard data=score mean=75 std=5 out=stdntest;
```

Specify the variables to standardize. The VAR statement specifies the variables to standardize and their order in the output.

```
var test1 test2;
run;
```

Create a data set that combines the original values with the standardized values. PROC SQL joins Score and Stdntest to create the Combined data set (table) that contains standardized and original test scores for each student. Using AS to rename the standardized variables NEW.TEST1 to StdTest1 and NEW.TEST2 to StdTest2 makes the variable names unique.

```
proc sql;
create table combined as
select old.student, old.studentnumber,
       old.section,
       old.test1, new.test1 as StdTest1,
       old.test2, new.test2 as StdTest2,
       old.final
from score as old, stdntest as new
where old.student=new.student;
```

Print the data set. PROC PRINT prints the Combined data set. ROUND rounds the standardized values to two decimal places. The TITLE statement specifies a title.

```
proc print data=combined noobs round;
title 'Standardized Test Scores for a College Course';
run;
```

Output: Listing

The following data set contains variables with both standardized and original values. StdTest1 and StdTest2 store the standardized test scores that PROC STANDARD computes.

Output 58.3 Standardized Test Scores for a College Course

Standardized Test Scores for a College Course							1
Student	Student Number	Section	Test1	Std Test1	Test2	Std Test2	Final
Capalleti	0545	1	94	80.54	91	80.86	87
Dubose	1252	2	51	64.39	65	71.63	91
Engles	1167	1	95	80.91	97	82.99	97
Grant	1230	2	63	68.90	75	75.18	80
Krupski	2527	2	80	75.28	69	73.05	71
Lundsford	4860	1	92	79.79	40	62.75	86
McBane	0674	1	75	73.40	78	76.24	72
Mullen	6445	2	89	78.66	82	77.66	93
Nguyen	0886	1	79	74.91	76	75.53	80
Patel	9164	2	71	71.90	77	75.89	83
Si	4915	1	75	73.40	71	73.76	73
Tanaka	8534	2	87	77.91	73	74.47	76

Example 2: Standardizing BY Groups and Replacing Missing Values

Features:

- PROC STANDARD statement options
- PRINT
- REPLACE
- BY statement
- FORMAT procedure
- PRINT procedure
- SORT procedure

Details

This example does the following:

- calculates Z scores separately for each BY group using a mean of 0 and standard deviation of 1
- replaces missing values with the given mean
- prints the mean and standard deviation for the variables to standardize
- prints the output data set

Program

```
options nodate pageno=1 linesize=80 pagesize=60;

proc format;
  value popfmt 1='Stable'
               2='Rapid';
run;

data lifexp;
  input PopulationRate Country $char14. Life50 Life93 @@;
  label life50='1950 life expectancy'
        life93='1993 life expectancy';
  datalines;
2 Bangladesh      .  53 2 Brazil      51 67
2 China           41 70 2 Egypt      42 60
2 Ethiopia        33 46 1 France     67 77
1 Germany         68 75 2 India       39 59
2 Indonesia       38 59 1 Japan       64 79
2 Mozambique      .  47 2 Philippines 48 64
1 Russia          .  65 2 Turkey      44 66
```

```

1 United Kingdom 69 76 1 United States 69 75
;
proc sort data=lifexp;
    by populationrate;
run;

proc standard data=lifexp mean=0 std=1 replace
    print out=zscore;

    by populationrate;

    format populationrate popfmt.;
    title1 'Life Expectancies by Birth Rate';
run;

proc print data=zscore noobs;
    title 'Standardized Life Expectancies at Birth';
    title2 'by a Country's Birth Rate';
run;

```

Program Description

Set the SAS system options. The NODATE option specifies to omit the date and time at which the SAS job began. The PAGENO= option specifies the page number for the next page of output that SAS produces. The LINESIZE= option specifies the line size. The PAGESIZE= option specifies the number of lines for a page of SAS output.

```
options nodate pageno=1 linesize=80 pagesize=60;
```

Assign a character string format to a numeric value. PROC FORMAT creates the format POPFMT to identify birth rates with a character value.

```

proc format;
    value popfmt 1='Stable'
                2='Rapid';
run;

```

Create the Lifexp data set. Each observation in this data set contains information about 1950 and 1993 life expectancies at birth for 16 nations. The birth rate for each nation is classified as stable (1) or rapid (2). The nations with missing data obtained independent status after 1950. Data are from Vital Signs 1994: The Trends That Are Shaping Our Future, Lester R. Brown, Hal Kane, and David Malin Roodman, eds. Copyright © 1994 by Worldwatch Institute. Reprinted by permission of W.W. Norton & Company, Inc.

```

data lifexp;
    input PopulationRate Country $char14. Life50 Life93 @@;
    label life50='1950 life expectancy'
          life93='1993 life expectancy';
    datalines;
2 Bangladesh      . 53 2 Brazil      51 67
2 China           41 70 2 Egypt      42 60
2 Ethiopia        33 46 1 France     67 77
1 Germany         68 75 2 India       39 59
2 Indonesia       38 59 1 Japan       64 79
2 Mozambique      . 47 2 Philippines 48 64

```

```

1 Russia . 65 2 Turkey 44 66
1 United Kingdom 69 76 1 United States 69 75
;

```

Sort the Lifeexp data set. PROC SORT sorts the observations by the birth rate.

```

proc sort data=lifexp;
  by populationrate;
run;

```

Generate the standardized data for all numeric variables and create the Z-score output data set. PROC STANDARD standardizes all numeric variables to a mean of 1 and a standard deviation of 0. REPLACE replaces missing values. PRINT prints statistics.

```

proc standard data=lifexp mean=0 std=1 replace
  print out=zscore;

```

Create the standardized values for each BY group. The BY statement standardizes the values separately by birth rate.

```

  by populationrate;

```

Assign a format to a variable and specify a title for the report. The FORMAT statement assigns a format to PopulationRate. The output data set contains formatted values. The TITLE statement specifies a title.

```

  format populationrate popfmt.;
  title1 'Life Expectancies by Birth Rate';
run;

```

Print the data set. PROC PRINT prints the ZSCORE data set with the standardized values. The TITLE statements specify two titles to be printed.

```

proc print data=zscore noobs;
  title 'Standardized Life Expectancies at Birth';
  title2 'by a Country's Birth Rate';
run;

```

Output: Listing

PROC STANDARD prints the variable name, mean, standard deviation, input frequency, and label of each variable to standardize for each BY group. Life expectancies for Bangladesh, Mozambique, and Russia are no longer missing. The missing values are replaced with the given mean (0).

Output 58.4 Life Expectancies by Birth Rate

Life Expectancies by Birth Rate				
The STANDARD Procedure				
PopulationRate=Stable				
Name	Mean	Standard Deviation	N	Label
Life50	67.400000	2.073644	5	1950 life expectancy
Life93	74.500000	4.888763	6	1993 life expectancy
PopulationRate=Rapid				
Name	Mean	Standard Deviation	N	Label
Life50	42.000000	5.707138	8	1950 life expectancy
Life93	59.100000	8.225300	10	1993 life expectancy
Standardized Life Expectancies at Birth by a Country's Birth Rate				
PopulationRate	Country	Life50	Life93	
Stable	France	-0.19290	0.51138	
Stable	Germany	0.28935	0.10228	
Stable	Japan	-1.63963	0.92048	
Stable	Russia	0.00000	-1.94323	
Stable	United Kingdom	0.77159	0.30683	
Stable	United States	0.77159	0.10228	
Rapid	Bangladesh	0.00000	-0.74161	
Rapid	Brazil	1.57697	0.96045	
Rapid	China	-0.17522	1.32518	
Rapid	Egypt	0.00000	0.10942	
Rapid	Ethiopia	-1.57697	-1.59265	
Rapid	India	-0.52566	-0.01216	
Rapid	Indonesia	-0.70088	-0.01216	
Rapid	Mozambique	0.00000	-1.47107	
Rapid	Philippines	1.05131	0.59572	
Rapid	Turkey	0.35044	0.83888	

STREAM Procedure

Overview: <i>STREAM</i> Procedure	2139
What Does the STREAM Procedure Do?	2139
Concepts: <i>STREAM</i> Procedure	2140
How Text and Macro Code Are Interpreted	2140
Tokenizer Limitations	2141
Syntax: <i>STREAM</i> Procedure	2142
PROC STREAM Statement	2142
Usage: <i>STREAM</i> Procedure	2145
Using Macro-Based Code in the Input Stream	2145
Beginning an Input Stream with Brackets or Parentheses	2146
Executing SAS Code	2146
Rich Text Format (RTF) File Output	2147
Using the READFILE Keyword	2148
Using %INCLUDE to Include a File in PROC STREAM	2149
Inserting a New Line into the Output Stream	2150
Ending the STREAM Procedure	2150

Overview: STREAM Procedure

What Does the STREAM Procedure Do?

The STREAM procedure enables you to process an input stream that consists of arbitrary text that can contain SAS macro specifications. The macros are executed and expanded while the other text in the input stream is preserved. The text stream is not validated as SAS syntax. The output stream is sent to an external file that is referenced by a fileref and that can be defined to use any traditional SAS output destination.

Concepts: STREAM Procedure

How Text and Macro Code Are Interpreted

The following example uses a macro called %DOIT to produce an HTML stream that contains a table:

```
%macro doit(nrows,ncols);
<table>
%do i=1 %to &nrows;
<tr>
%do j=1 %to &ncols;
<td>&j</td>
%end;
</tr>
%end;
</table>
%mend doit;
```

If this macro is executed within a SAS code stream, then HTML output will be produced but the output will be syntactically invalid because it is not valid SAS syntax. However, if this macro is executed through PROC STREAM, the HTML output will instead be written to a file and not validated as SAS syntax. The following is an example:

```
proc stream outfile=myfile; begin
%doit(2,3)
;;;

```

The following output is written to *myfile*:

```
<table> <tr> <td>1</td> <td>2</td> <td>3</td> </tr> <tr> <td>1</td> <td>2</td> <td>3</td> </tr>
</table>
```

You can use the %INCLUDE statement in the input stream. The types of input files that you can use include HTML files, RTF files, or any other type of text-based file. Note that you would not likely use %INCLUDE to include actual SAS code except for macro definitions and invocations. Any actual SAS code beyond macros is not executed, but is treated like any other text.

Text that follows a % and that is recognized by a macro is executed. Examples include %LET, %INCLUDE, and %SYSFUNC. Any other macros, such as user-defined macros, are also executed.

All macro variable references are resolved, but warnings are issued if the macro name is not recognized. Macro statements such as %PUT and %LIST send output to the log and do not produce tokens. You should avoid using these statements in an input stream, although they are allowed.

Tokenizer Limitations

The SAS tokenizer or word-scanner has a number of limitations that can be a problem for correctly processing syntax input streams other than SAS syntax input streams. These limitations are described in the following list.

- Input records that come from the %INCLUDE statement cannot exceed a record size of 32,767 bytes, which is the default value.
- The tokenizer does not provide accurate information about record size to PROC STREAM if the input record from %INCLUDE exceeds 32,767 bytes.
- %INCLUDE and %LET statements must begin on a statement boundary. That is, a semicolon must precede the statements, and the statements must end with a semicolon. Macro invocations do not have this requirement.
- Macro statements are completely consumed and executed by the tokenizer, and PROC STREAM is not aware of them. For example, %let xyz=2; is completely consumed. This behavior also holds true for macro statements that are not global in scope, such as %IF and %GOTO. These macro statements are flagged as errors but PROC STREAM is not aware of the error.
- Any token that starts with the letters a-z (uppercase or lowercase), or an underscore that is preceded by an &, is assessed by the macro processor to determine whether it is a macro variable to be resolved. If it is not a macro variable, a warning is issued. Currently, this warning cannot be suppressed.
- PL/I programming language comments (/* comment */) are typically completely absorbed by the tokenizer and are not seen by PROC STREAM. You can use the NOABSSCMT option in the PROC STREAM statement so that all text between and including the /* and */ are seen. This option is also needed if the input stream can contain occurrences of /* that are not enclosed in quotation marks. An example of this is /tmp/xyz/*, which is a Linux wildcard specification.
- No single token can exceed 32,767 characters.
- If the last token in a record is followed by a line feed that is preceded by an optional carriage return, and the first token in the next record begins in column 1, then the tokenizer introduces an artificial blank between these tokens. The blank allows for valid SAS syntax, because these tokens are not concatenated unless they are part of a string that is enclosed in quotation marks. For example, a record is 80 characters long. The string *abc*, preceded by a blank, is located in columns 78–80. In the next record, the string *def*, followed by a blank, appears in columns 1–3. In this case, PROC STREAM receives *abc* followed by *def*, which has a leading blank, even though a leading blank did not appear in the token stream. If, instead, *'abc'*, preceded by a blank, appears in columns 77–80, and *def'*, followed by a blank, appears in columns 1–4, then PROC STREAM receives *'abcdef'*.

- Non-printable characters (except for carriage return (CR) and line feed (LF)) cannot be properly tokenized.
- The input stream cannot be a binary stream such as a PDF or a JPG file.
- The normal behavior for the tokenizer is to provide a string, either with single or double quotation marks, as a single token. This is a problem if text, such as *don't do this*, is not enclosed in quotation marks, or if text has an escape sequence, as in *don't do this*. The QUOTING= option can be used to allow for quotation marks to be treated like any other special character, such as a hyphen or a slash.
- A single macro variable can expand into no more than 64,000 characters. However, there is no limit to what a macro can return as model text, apart from resource limitations such as memory and disk space.

Syntax: STREAM Procedure

```
PROC STREAM OUTFILE=fileref <options>; BEGIN
text-1
<text-2 ...>
;;;;
```

Statement	Task
PROC STREAM	Process an input stream that consists of arbitrary text that can contain SAS macro specifications

PROC STREAM Statement

Enables you to process an input stream that consists of arbitrary text that can contain SAS macro specifications.

Syntax

```
PROC STREAM OUTFILE=fileref <options>; BEGIN
text-1
<text-2 ...>
;;;;
```

Required Arguments

OUTFILE=fileref

specifies the file where all tokens are written.

The LRECL specification for the fileref is used. If no LRECL is given, then the default value for the global LRECL= option, which is 32,767 bytes, is used. Unless you use the PRESCOL option, all tokens are streamed out with the proper number of intervening blanks between tokens. No token is broken between records. Also, no stream of tokens is broken between records unless there is at least one blank within them. For example, `<table>X</table>` will not be broken between records, but `<table> X </table>` can be broken where you see the blanks (before and after the X).

text

specifies the SAS statements or macros to use with PROC STREAM.

Optional Arguments

MOD

specifies that the output file is appended to instead of being overwritten.

NOABSSCMT

specifies whether comments are written to the output stream.

If PL/I programming language style comments appear (`/* comments */`), all text between the comment characters (`/*` and `*/`) appears in the output stream. If this option is omitted, the PL/I-style comments do not appear in the output stream. Note that if NOABSSCMT is set, it is strongly suggested that QUOTING= also be set, because single quotation marks (such as in the word don't) can commonly appear in comments.

PRESCOL

indicates that an attempt is made to preserve the columns of the original input file.

Using this option is not as successful if there is macro substitution or if the record size exceeds 32,767 bytes. In this case, macro expansion might affect column location.

The PRESCOL option improves the validity of RTF files that are included with the %INCLUDE macro.

QUOTING=SINGLE | DOUBLE | BOTH

specifies how quotation marks are handled.

SINGLE

specifies that single quotation marks are treated like any other character. If you use the SINGLE option and macro references occur within single quotation marks, such as `'&hello'`, the macro references are expanded.

DOUBLE

specifies that double quotation marks are treated like any other character.

BOTH

specifies that both SINGLE and DOUBLE options are used.

RESETDELIM='label'

indicates a special marker token.

This option is used when there is a need for statements to be expanded, such as the %INCLUDE and %LET statements. These statements must begin on a statement boundary. If your syntax does not allow for a statement boundary, then the given label, followed by a semicolon, can be introduced in the input stream to satisfy the tokenizer requirements. The label and semicolon are not sent to the output file.

In the following example, %INCLUDE is not preceded by a semicolon, and the code is not correct:

```
x y %include myfile; z
```

However, if you use the RESETDELIM= option, the expansion occurs as expected:

```
resetdelim='mylabel'
x y mylabel;
%include myfile; z
```

Mylabel; is not seen in the output file.

Mylabel must be a valid SAS name, that is, it must begin with a letter or underscore, and all subsequent characters must be letters, underscores, or digits. The length of *mylabel* must not exceed 32 characters.

The specification and usage is not case-sensitive. There is no default value for RESETDELIM=.

PROC STREAM checks for the existence of a macro variable called &STREAMDELIM. If that macro variable exists, it is used as is. If it does not exist, PROC STREAM verifies whether the RESETDELIM= option was given. If this is the case, the macro variable &STREAMDELIM is set to the value of the RESETDELIM= option. If the RESETDELIM= option is not given, PROC STREAM constructs a unique value for the &STREAMDELIM macro variable based on the current datetime value.

With this consideration, you can add &STREAMDELIM into the input for PROC STREAM. When this macro variable is seen, it is assumed that optional keywords will follow. A closing semicolon is then expected. All tokens from the &STREAMDELIM value up to and including the semicolon are not emitted to the output stream, but are instead special control information items for PROC STREAM.

If you need to provide a %INCLUDE statement in your input, but a semicolon does not precede it, then you must add the following statement before the %INCLUDE statement:

```
&STREAMDELIM;
```

This statement forces a semicolon to be seen by the tokenizer, but it is absorbed by PROC STREAM.

The following optional keywords can follow &STREAMDELIM:

NEWLINE

specifies that a new line is emitted to the output file.

READFILE *filename*

specifies that the given filename is opened, and its contents are read as is and written to the output file. There is no macro expansion of the contents of this file, and new lines are preserved. This differs from %INCLUDE, where macro expansion occurs and new lines are ignored. For an example of how to use the READFILE keyword, see [“Using the READFILE Keyword” on page 2148](#).

Usage: STREAM Procedure

Using Macro-Based Code in the Input Stream

Any macro-based code in the input stream is expanded, as shown in this example:

```
%let abc=123;
proc stream outfile=out; begin
<title>Run &abc</title>
<table>
<tr> <td>1<td>2</tr>
<tr> <td>3<td>4</tr>
</table>
;;;
```

The output file contains the following results:

```
<title>Run 123</title> <table> <tr> <td>1<td>2</tr> <tr> <td>3<td>4</tr> </table>
```

Note that there are no line breaks, but blanks are preserved, and a blank is inserted for each line break.

The fact that the occurrence of % and & in the input stream are not enclosed in quotation marks and are not escape sequences can cause problems. For example, *&* appears in HTML streams as an escape sequence for an ampersand. If you have a macro variable called *&*, then its value is substituted for *&*. If this type of macro variable is not present, then SAS issues a warning. You can avoid these problems by using the & as an escape character, as in %STR(&), as the following example shows:

```
<title>A %str(&)amp; B</title>
```

The following output is written to the output stream:

```
<title>A &amp; B</title>
```

Likewise, for %, any occurrence that is not to be misinterpreted by a macro should appear as %STR(%), as the following example shows:

```
<a href="www.sas.com/x%str(%20y)">the link</a>
```

The following output is written to the output stream:

```
<a href="www.sas.com/x%20y">the link</a>
```

If the escape sequence for %STR occurs within single quotation marks, then either use the QUOTING= option, or use the escape sequence with a single quotation mark, %STR('%').

Beginning an Input Stream with Brackets or Parentheses

When you start your input stream with brackets or parentheses, specifically {, [, or (, this causes the BEGIN keyword to be treated as a name. This is true even when the first input character is on the line after BEGIN in your code. For example, the following code would treat BEGIN as a name rather than as the keyword that identifies the start of an input stream:

```
filename dheader "c:\temp\dheader.txt";
proc stream outfile=dheader; begin
[SAS version 9.4]
;;;
```

To prevent this, use the &STREAMDELIM; statement. You can always use the &STREAMDELIM; statement after BEGIN to indicate the beginning of any input stream.

When you include this statement, the starting point of the input stream is interpreted correctly:

```
filename dheader "c:\temp\dheader.txt"
proc stream outfile=dheader; begin &streamdelim;
[SAS version 9.4]
;;;
```

Executing SAS Code

As described in a previous section, any SAS code that is encountered is not executed. The tokens are streamed out like any other tokens. An exception to this occurs when the %SYSFUNC or %SYSCALL macro functions are encountered. These macro functions do execute the specified function, as shown in the following example:

```
%let abc=%sysfunc(getoption(obs));
&abc
```

This statement causes the GETOPTION function to be called, which obtains the value of the OBS option. GETOPTION returns the value as a character string to be placed into the ABC macro variable. That value is then written to the output stream.

It is possible to execute a stream of SAS code by using the DOSUB function in %SYSFUNC. The DOSUB function is provided a fileref, and all of the SAS code within that file is executed.

In this example, the fileref MYCODE points to the following SAS code:

```
filename myhtml "c:\temp\temp.txt";

data _null_;
  file myhtml;
  put '<table>';
  put '<tr><td>1</td></tr>';
  put '<tr><td>2</td></tr>';
  put '</table>';
run;
```

If you then execute the following STREAM procedure, you can see that PROC STREAM uses *mycode* as an argument to DOSUB in the %SYSFUNC function:

```
filename myhtml "c:\temp\temp.txt";
filename new "c:\temp\new.html";

proc stream outfile=new; begin
%let abc=%sysfunc(dosub(mycode));
%include myhtml;
;;;
```

The output stream contains the following results:

```
<table><tr><td>1</td></tr><tr><td>2</td></tr></table>
```

Note: The DOSUB function is similar to the DOSUBL function, but DOSUB is passed a fileref for a file that contains SAS code. DOSUBL is passed a text string and executes the value as SAS code. For more information, see [“DOSUBL Function” in SAS Functions and CALL Routines: Reference](#).

Rich Text Format (RTF) File Output

You can use Microsoft Word to input data and create RTF output. If you enter Today is &sysdate.. and My name is &name.., Microsoft Word saves the data as an RTF file called Mytest.rtf. This file is approximately 30K in size, and contains a large amount of markup data. A section of that data contains the actual text that was entered, as seen here:

```
\fs20\lang1033\langfe1033\cgrid\langnp1033\langfenp1033 {\rtlch\fcs1 \af0 \ltrch\fcs0
\insrsid8922096
Today is &sysdate.. }{\rtlch\fcs1 \af0 \ltrch\fcs0 \insrsid14092174
\par }{\rtlch\fcs1 \af0 \ltrch\fcs0 \insrsid8922096
\par My name is &name..
\par
```

Note that the text as entered has macro substitutions.

When you execute this SAS program, a new RTF file is created called Newrtf.rtf. The difference in this output file is that the macro substitutions have taken place, but the markup language remains intact:

```
%let name=John;
filename oldrtf 'mytest.rtf' recfm=v lrecl=32767;
filename newrtf 'newrtf.rtf' recfm=v lrecl=32767;
proc stream outfile=newrtf quoting=both asis; begin
&streamdelim;
%include oldrtf;
;;;
```

Then when the Newrtf.rtf file is read by Microsoft Word, this is what is displayed:

```
Today is 30NOV12.

My name is John.
```

Using the READFILE Keyword

In HTML, the <PRE> ... </PRE> tags encapsulate a stream of text that has been preformatted, and is displayed as is, honoring new lines and spacing. You can read the contents of a file and have it preserved as is by using the READFILE keyword. The READFILE keyword follows the &STREAMDELIM macro variable:

```
filename fixed temp;
data _null_; file fixed;
    input; put _infile_;
    datalines4;
This is the first line of fixed text.
This is another line to be fixed.
This is the last line of fixed text.
;;;

%macro doit(nrows,ncols);
<table>
%do i=1 %to &nrows;
<tr>
%do j=1 %to &ncols;
<td>&j</td>
%end;
</tr>
%end;
%mend;

filename new temp;
proc stream outfile=new; begin
<PRE>
&streamdelim readfile fixed;
</PRE>
%doit(2,3)
;;;
```

```

data _null_; infile new; input; put _infile_; run;
filename fixed temp;
data _null_; file fixed;
    input; put _infile_;
    datalines4;
This is the first line of fixed text.
This is another line to be fixed.
This is the last line of fixed text.
;;;

%macro doit(nrows,ncols);
<table>
%do i=1 %to &nrows;
<tr>
%do j=1 %to &ncols;
<td>&j</td>
%end;
</tr>
%end;
%mend;

filename new temp;
proc stream outfile=new; begin
<PRE>
&streamdelim readfile fixed;
</PRE>
%doit(2,3)
;;;

data _null_; infile new; input; put _infile_; run;

```

This is the resulting file:

```

<PRE>
This is the first line of fixed text.
This is another line to be fixed.
This is the last line of fixed text.
</PRE> <table> <tr> <td>1</td> <td>2</td> <td>3</td> </tr>
<tr> <td>1</td> <td>2</td> <td>3</td> </tr> </table>

```

Using %INCLUDE to Include a File in PROC STREAM

To stream complex HTML and Javascript code, you can create the code in a separate file. Then use the %INCLUDE statement to include the file in your program.

The following example shows this approach.

```
filename temp1 temp;
```

```

data _null_;
  infile datalines;
  file temp1;
  input;
  l=length(_infile_);
  put @1 _infile_ $varying200. 1;
  datalines4;
<table>
<tr>
<td>abc</td><td>def</td>
</tr>
<tr>
<td>ghi</td><td>jkl</td>
</tr>
</table>
;;;

filename myfile "c:\temp\test1.html";

proc stream outfile=myfile prescol resetdelim='label'; begin
<html>
<h2>Test Example</h2>
label;
%include temp1;
</html>
;;;

```

Inserting a New Line into the Output Stream

To insert a new line into the output stream, add the keyword `NEWLINE` after the delimiter and before the semicolon, as shown in the following example:

```

proc stream outfile=abc resetdelim='mylabel'; begin
my item here;
mylabel newline;
my next item here
;;;

```

This example produces the following output:

```

my item here;
my next item here

```

There is no other way to ensure a predictable new line.

Ending the STREAM Procedure

The end of the `PROC STREAM` step is indicated by four semicolons with no blanks between them, written as the last line of the procedure. If the last item in the input

stream ends with a semicolon, use the label that is specified in RESETDELIM=, followed by a semicolon, immediately after the last item.

The following example is not coded correctly because the semicolon after here is not recognized:

```
proc stream outfile=abc; begin
my item here;
;;;
```

The following example is coded correctly:

```
proc stream outfile=abc resetdelim='mylabel'; begin
my item here;
mylabel;
;;;
```

RESETDELIM= is used in this example because it recognizes the semicolon after here.

SUMMARY Procedure

Overview: SUMMARY Procedure	2153
What Does the SUMMARY Procedure Do?	2153
Syntax: SUMMARY Procedure	2154
PROC SUMMARY Statement	2155
ATTRIB Statement	2168
BY Statement	2169
CLASS Statement	2173
FORMAT Statement	2179
FREQ Statement	2179
ID Statement	2180
LABEL Statement	2181
OUTPUT Statement	2183
TYPES Statement	2193
VAR Statement	2195
WAYS Statement	2195
WEIGHT Statement	2196
WHERE Statement	2203

Overview: SUMMARY Procedure

What Does the SUMMARY Procedure Do?

The SUMMARY procedure provides data summarization tools that compute descriptive statistics for variables across all observations or within groups of observations. The SUMMARY procedure is very similar to the MEANS procedure; for full syntax details, see [Chapter 35, “MEANS Procedure,” on page 1227](#). Except for the differences that are discussed here, all the PROC MEANS information also applies to PROC SUMMARY.

When preloading a character format, there must be at least one representative raw value that produces the label. If a range does not have any raw values, the range is considered unreachable and an error is generated. For example, you cannot preload a multi-label character format with both Other= and Low-High ranges. Other is a special keyword that refers to values or ranges that remain after all labels have been defined. For character formats, missing values are included in the low range so that there are no raw values left to occupy the Other range. See [for more information on the PRELOADFMT option](#).

PROC SUMMARY can be used in SAS Cloud Analytic Services (CAS). Running PROC SUMMARY with CAS actions has several advantages over processing within SAS. See [Chapter 35, “MEANS Procedure,” on page 1227](#) for information about running PROC MEANS and PROC SUMMARY in CAS.

The VARCHAR variable type is not supported in PROC SUMMARY. Any variables declared as VARCHAR will be converted to CHAR.

Calculated statistics can vary slightly, depending on the order in which observations are processed. Such variations are due to numerical errors that are introduced by floating-point arithmetic, the results of which should be considered approximate and not exact. The order of observation processing can be affected by nondeterministic effects of multithreaded or parallel processing. The order of processing can also be affected by inconsistent or nondeterministic ordering of observations that are produced by a data source, such as a DBMS that delivers query results through an ACCESS engine. For more information, see [“Threading in Base SAS” in SAS Programmer’s Guide: Essentials](#)

Syntax: SUMMARY Procedure

Supports: Threaded processing

Tips: Supports the Output Delivery System. For details, see [“Output Delivery System: Basic Concepts” in SAS Output Delivery System: User’s Guide](#).

You can use the ATTRIB, FORMAT, LABEL, and WHERE statements.

Full syntax descriptions are in [“Syntax” on page 1239](#).

PROC SUMMARY <options> <statistic-keyword(s)>;

ATTRIB *variable-list(s) attribute-list(s)*;

BY <DESCENDING> *variable-1* <<DESCENDING> *variable-2 ...*>
<NOTSORTED>;

CLASS *variable(s)* </ options>;

FORMAT *variable-1* <...*variable-n*> <*format*> <DEFAULT=*default-format*>;

FREQ *variable*;

ID *variable(s)*;

LABEL *variable-1=**label-1*<...*variable-n=**label-n*>;

OUTPUT <OUT=*SAS-data-set*> <*output-statistic-specification(s)*>
<*id-group-specification(s)*> <*maximum-id-specification(s)*>

<minimum-id-specification(s)> </ options> ;

TYPES *request(s);*

VAR *variable(s) </ WEIGHT=weight-variable>;*

WAYS *list;*

WEIGHT *variable;*

WHERE *where-expression-1*

< logical-operator where-expression-n>;

Statement	Task
PROC SUMMARY	Compute descriptive statistics for variables across all observations or within groups of observations
BY	Calculate separate statistics for each BY group
CLASS	Identify variables whose values define subgroups for the analysis
FREQ	Identify a variable whose values represent the frequency of each observation
ID	Include additional identification variables in the output data set
OUTPUT	Create an output data set that contains specified statistics and identification variables
TYPES	Identify specific combinations of class variables to use to subdivide the data
VAR	Identify the analysis variables and their order in the results
WAYS	Specify the number of ways to make unique combinations of class variables
WEIGHT	Identify a variable whose values weight each observation in the statistical calculations

PROC SUMMARY Statement

Computes descriptive statistics for variables across all observations or within groups of observations.

See: For full syntax details, see [“PROC MEANS Statement” on page 1240](#).

Syntax

PROC SUMMARY *<options>* *<statistic-keyword(s)>*;

Optional Arguments

ALPHA=*value*

specifies the confidence level to compute the confidence limits for the mean. The percentage for the confidence limits is $(1 - \text{value}) \times 100$. An example is (ALPHA=.05 results in a 95% confidence limit).

Default .05

Range between 0 and 1

Interaction To compute confidence limits specify the *statistic-keyword* CLM, LCLM, or UCLM.

See [“Confidence Limits” on page 1294](#)

[“Example 7: Computing a Confidence Limit for the Mean” on page 1318](#)

CHARTYPE

specifies that the `_TYPE_` variable in the output data set is a character representation of the binary value of `_TYPE_`. The length of the variable equals the number of class variables.

Interaction When you specify more than 32 class variables, `_TYPE_` automatically becomes a character variable.

See [“Output Data Set” on page 1297](#)

[“Example 10: Computing Output Statistics with Missing Class Variable Values” on page 1325](#)

CLASSDATA=SAS-*data-set*

specifies a data set that contains the combinations of values of the class variables that must be present in the output. Any combinations of values of the class variables that occur in the CLASSDATA= data set but not in the input data set appear in the output and have a frequency of zero.

Restriction The CLASSDATA= data set must contain all class variables. Their data type and format must match the corresponding class variables in the input data set.

Interaction If you use the EXCLUSIVE option, then PROC MEANS excludes any observation in the input data set whose combination of class variables is not in the CLASSDATA= data set.

Tip Use the CLASSDATA= data set to filter or to supplement the input data set.

See [“Example 4: Using a CLASSDATA= Data Set with Class Variables” on page 1306](#)

COMPLETETYPES

creates all possible combinations of class variables even if the combination does not occur in the input data set.

Interaction The PRELOADFMT option in the CLASS statement ensures that PROC MEANS writes all user-defined format ranges or values for the combinations of class variables to the output, even when a frequency is zero.

Tip Using COMPLETETYPES does not increase the memory requirements.

See [“Example 6: Using Preloaded Formats with Class Variables” on page 1314](#)

DATA=SAS-data-set

identifies the input SAS data set.

See [“Input Data Sets” on page 24](#)

DESCENDTYPES

orders observations in the output data set by descending _TYPE_ value.

Aliases DESCENDING
 DESCEND

Interaction Descending has no effect if you specify NWAY.

Tip Use DESCENDTYPES to make the overall total (_TYPE_=0) the last observation in each BY group.

See [“Output Data Set” on page 1297](#)

[“Example 9: Computing Different Output Statistics for Several Variables” on page 1322](#)

EXCLNPWGT

excludes observations with nonpositive weight values (zero or negative) from the analysis. By default, PROC MEANS treats observations with negative weights like observations with zero weights and counts them in the total number of observations.

Alias EXCLNPWGTS

See [“WEIGHT Statement” on page 2196](#)

[WEIGHT= option on the VAR statement on page 1282](#)

EXCLUSIVE

excludes from the analysis all combinations of the class variables that are not found in the CLASSDATA= data set.

Requirement If a CLASSDATA= data set is not specified, then this option is ignored.

See [“Example 4: Using a CLASSDATA= Data Set with Class Variables” on page 1306](#)

FW=field-width

specifies the field width to display the statistics in printed or displayed output. FW= has no effect on statistics that are saved in an output data set.

Default 12

Tip If PROC MEANS truncates column labels in the output, then increase the field width.

See [“Example 1: Computing Specific Descriptive Statistics” on page 1299](#)

[“Example 4: Using a CLASSDATA= Data Set with Class Variables” on page 1306](#)

[“Example 5: Using Multilabel Value Formats with Class Variables” on page 1309](#)

INCAS=(YES | NO)

specifies whether to allow in-CAS processing.

YES

Use in-CAS processing. YES is the default.

NO

Do not use in-CAS processing.

IDMIN

specifies that the output data set contain the minimum value of the ID variables.

Interaction Specify PRINTIDVARS to display the value of the ID variables in the output.

See [“ID Statement” on page 2180](#)

MAXDEC=number

specifies the maximum number of decimal places to display the statistics in the printed or displayed output. MAXDEC= has no effect on statistics that are saved in an output data set.

Default BEST. width for columnar format, typically about 7.

Range 0-8

See [“Example 2: Computing Descriptive Statistics with Class Variables” on page 1301](#)

[“Example 4: Using a CLASSDATA= Data Set with Class Variables” on page 1306](#)

MISSING

considers missing values as valid values to create the combinations of class variables. Special missing values that represent numeric values (the letters A through Z and the underscore (_) character) are each considered as a separate value.

Default If you omit MISSING, then PROC MEANS excludes the observations with a missing class variable value from the analysis.

See *SAS Programmer's Guide: Essentials* for a discussion of missing values that have special meaning.

[“Example 6: Using Preloaded Formats with Class Variables” on page 1314](#)

NOLABEL

suppresses variable labels.

```
Example  proc means data=sashelp.shoes maxdec=2 nolabels;
          var sales returns;
          run;
```

NONOBS

suppresses the column that displays the total number of observations for each unique combination of the values of the class variables. This column corresponds to the _FREQ_ variable in the output data set.

See [“The N Obs Statistic” on page 1297](#)

[“Example 5: Using Multilabel Value Formats with Class Variables” on page 1309](#)

[“Example 6: Using Preloaded Formats with Class Variables” on page 1314](#)

NOPRINT

suppresses all the output. See the `PRINT` option.

Tip Use NOPRINT when you want to create only an OUT= output data set.

NOTRAP

disables floating point exception (FPE) recovery during data processing. By default, PROC MEANS traps these errors and sets the statistic to missing.

In operating environments where the overhead of FPE recovery is significant, NOTRAP can improve performance. Note that normal SAS FPE handling is still in effect so that PROC MEANS terminates in the case of math exceptions.

NWAY

specifies that the output data set contain only statistics for the observations with the highest _TYPE_ and _WAY_ values. When you specify class variables, NWAY corresponds to the combination of all class variables.

Interaction If you specify a TYPES statement or a WAYS statement, then PROC MEANS ignores this option.

See [“Output Data Set” on page 1297](#)

[“Example 10: Computing Output Statistics with Missing Class Variable Values” on page 1325](#)

ORDER=DATA | FORMATTED | FREQ | UNFORMATTED

specifies the sort order to create the unique combinations for the values of the class variables in the output, where

DATA

orders values according to their order in the input data set.

Interaction If you use PRELOADFMT in the CLASS statement, then the order for the values of each class variable matches the order that PROC FORMAT uses to store the values of the associated user-defined format. If you use the CLASSDATA= option, then PROC MEANS uses the order of the unique values of each class variable in the CLASSDATA= data set to order the output levels. If you use both options, then PROC MEANS first uses the user-defined formats to order the output. If you omit EXCLUSIVE, then PROC MEANS appends after the user-defined format and the CLASSDATA= values the unique values of the class variables in the input data set based on the order in which they are encountered.

Tip By default, PROC FORMAT stores a format definition in sorted order. Use the NOTSORTED option to store the values or ranges of a user-defined format in the order in which you define them.

FORMATTED

orders values by their ascending formatted values. This order depends on your operating environment.

Aliases FMT

EXTERNAL

FREQ

orders values by descending frequency count so that levels with the most observations are listed first.

Interactions For multiway combinations of the class variables, PROC MEANS determines the order of a class variable combination from the individual class variable frequencies.

Use the ASCENDING option in the CLASS statement to order values by ascending frequency count.

UNFORMATTED

orders values by their unformatted values, which yields the same order as PROC SORT. This order depends on your operating environment.

Aliases UNFMT

INTERNAL

Default UNFORMATTED

See [“Ordering the Class Values” on page 1232](#)

PRINT

specifies whether PROC MEANS displays the statistical analysis. NOPRINT suppresses all the output.

Default PRINT

Tip Use NOPRINT when you want to create only an OUT= output data set.

See [“Example 8: Computing Output Statistics” on page 1320](#)

[“Example 12: Identifying the Top Three Extreme Values with the Output Statistics” on page 1330](#)

PRINTALLTYPES

displays all requested combinations of class variables (all _TYPE_ values) in the printed or displayed output. Normally, PROC MEANS shows only the NWAY type.

Alias PRINTALL

Interaction If you use the NWAY option, the TYPES statement, or the WAYS statement, then PROC MEANS ignores this option.

See [“Example 4: Using a CLASSDATA= Data Set with Class Variables” on page 1306](#)

PRINTIDVARS

displays the values of the ID variables in printed or displayed output.

Alias PRINTIDS

Interaction Specify IDMIN to display the minimum value of the ID variables.

See [“ID Statement” on page 2180](#)

QMARKERS=*number*

specifies the default number of markers to use for the P^2 quantile estimation method. The number of markers controls the size of fixed memory space.

Default The default value depends on which quantiles you request. For the median (P50), *number* is 7. For the quantiles (P25 and P50), *number* is 25. For the quantiles P1, P5, P10, P75 P90, P95, or P99, *number* is 105. If you request several quantiles, then PROC MEANS uses the largest value of *number*.

Range an odd integer greater than 3

- Tip Increase the number of markers above the defaults settings to improve the accuracy of the estimate; reduce the number of markers to conserve memory and computing time.
- See [“Quantiles” on page 1295](#)

QMETHOD=OS | P2

specifies the method that PROC MEANS uses to process the input data when it computes quantiles. If the number of observations is less than or equal to the QMARKERS= value and QNTLDEF=5, then both methods produce the same results.

OS

uses order statistics. This method is the same method that PROC UNIVARIATE uses.

.....
Note: This technique can be very memory-intensive.
.....

P2

uses the P² method to approximate the quantile.

- Default OS
- Restriction When QMETHOD=P2, PROC MEANS does not compute MODE or weighted quantiles.
- Tip When QMETHOD=P2, reliable estimations of some quantiles (P1,P5,P95,P99) might not be possible for some data sets.
- See [“Quantiles” on page 1295](#)

QNTLDEF=1 | 2 | 3 | 4 | 5

specifies the mathematical definition that PROC MEANS uses to calculate quantiles when QMETHOD=OS. To use QMETHOD=P2, you must use QNTLDEF=5.

- Alias PCTLDEF=
- Default 5
- See [“Quantile and Related Statistics” on page 2432](#)

statistic-keyword(s)

specifies which statistics to compute and the order to display them in the output. The available keywords in the PROC statement are listed in [“Statistic Keywords” on page 2165](#).

- Default N, MEAN, STD, MIN, and MAX
- Requirement To compute standard error, confidence limits for the mean, and the Student’s *t*-test, you must use the default value of the VARDEF= option, which is DF. To compute skewness or kurtosis, you must use VARDEF=N or VARDEF=DF.

- Tip** Use CLM or both LCLM and UCLM to compute a two-sided confidence limit for the mean. Use only LCLM or UCLM, to compute a one-sided confidence limit.
- See** The definitions of the keywords and the formulas for the associated statistics are listed in [“Keywords and Formulas” on page 2426](#).
- [“Example 1: Computing Specific Descriptive Statistics” on page 1299](#)
- [“Example 3: Using the BY Statement with Class Variables” on page 1304](#)

STACKODSOUTPUT

produces an ODS output object whose data set resembles the printed output.

The STACKODSOUTPUT option affects output data sets created by ODS OUTPUT statements, not the PROC MEANS OUTPUT statement.

Alias STACKODS

See [“Example 13: Using the STACKODSOUTPUT Option to Control Data” on page 1335](#)

SUMSIZE=*value*

specifies the amount of memory that is available for data summarization when you use class variables. *value* might be one of the following:

n

nK

nM

nG

specifies the amount of memory available in bytes, kilobytes, megabytes, or gigabytes, respectively. If *n* is 0, then PROC MEANS use the value of the SAS system option SUMSIZE=.

MAXIMUM**MAX**

specifies the maximum amount of memory that is available.

Default The value of the SUMSIZE= system option.

Note Specifying SUMSIZE=0 enables PROC MEANS to use the preferred global REALMEMSIZE option.

Tip For best results, do not make SUMSIZE= larger than the amount of physical memory that is available for the PROC step. If additional space is needed, then PROC MEANS uses utility files.

See The SAS system option SUMSIZE= in *SAS System Options: Reference*.

THREADS | NOTHEADS

enables or disables parallel processing of the input data set. This option overrides the SAS system option THREADS | NOTHEADS unless the system

option is restricted. (See Restriction.) For more information, see [Using Parallel Processing](#).

Default	value of SAS system option THREADS NOTHREADS.
Restriction	Your site administrator can create a restricted options table. A restricted options table specifies SAS system option values that are established at start-up and cannot be overridden. If the THREADS NOTHREADS system option is listed in the restricted options table, any attempt to set these system options is ignored and a warning message is written to the SAS log.
Interaction	PROC MEANS honors the SAS system option THREADS except when a BY statement is specified or the value of the SAS system option CPUCOUNT is less than 2. You can use THREADS in the PROC MEANS statement to force PROC MEANS to use parallel processing in these situations.
Note	If THREADS is specified (either as a SAS system option or in the PROC MEANS statement) and another program has the input data set open for reading, writing, or updating, then PROC MEANS might fail to open the input data set. In this case, PROC MEANS stops processing and writes a message to the SAS log.

VARDEF=DF | N | WDF | WEIGHT

specifies the divisor to use in the calculation of the variance and standard deviation. The following table shows the possible values for *divisor* and associated divisors.

Table 60.1 Possible Values for VARDEF=

Value	Divisor	Formula for Divisor
DF	Degrees of freedom	$n - 1$
N	Number of observations	n
WDF	Sum of weights minus one	$(\sum_i w_i) - 1$
WEIGHT WGT	Sum of weights	$\sum_i w_i$

The procedure computes the variance as $CSS/divisor$, where CSS is the corrected sums of squares and equals $\sum (x_i - \bar{x})^2$. When you weight the analysis variables, CSS equals $\sum w_i(x_i - \bar{x}_w)^2$, where $\bar{x}_w$ is the weighted mean.

Default	DF
Requirement	To compute the standard error of the mean, confidence limits for the mean, or the Student's <i>t</i> -test, use the default value of VARDEF=.

Tips When you use the WEIGHT statement and VARDEF=DF, the variance is an estimate of σ^2 , where the variance of the i th observation is $\text{var}(x_i) = \sigma^2/w_i$ and w_i is the weight for the i th observation. This method yields an estimate of the variance of an observation with unit weight.

When you use the WEIGHT statement and VARDEF=WGT, the computed variance is asymptotically (for large n) an estimate of $\sigma^2/\bar{w}$, where $\bar{w}$ is the average weight. This method yields an asymptotic estimate of the variance of an observation with average weight.

See [“Keywords and Formulas” on page 2426](#)
[“WEIGHT Statement” on page 2196](#)

Statistic Keywords

CLM

is a descriptive statistic keyword. For details, see [“*statistic-keyword\(s\)*” on page 2162](#).

CSS

is a descriptive statistic keyword. For details, see [“*statistic-keyword\(s\)*” on page 2162](#).

KURTOSIS

is a descriptive statistic keyword. For details, see [“*statistic-keyword\(s\)*” on page 2162](#).

Alias KURT

LCLM

is a descriptive statistic keyword. For details, see [“*statistic-keyword\(s\)*” on page 2162](#).

MAX

is a descriptive statistic keyword. For details, see [“*statistic-keyword\(s\)*” on page 2162](#).

MEAN

is a descriptive statistic keyword. For details, see [“*statistic-keyword\(s\)*” on page 2162](#).

MIN

is a descriptive statistic keyword. For details, see [“*statistic-keyword\(s\)*” on page 2162](#).

MODE

is a descriptive statistic keyword. For details, see [“*statistic-keyword\(s\)*” on page 2162](#).

N

is a descriptive statistic keyword. For details, see [“*statistic-keyword\(s\)*” on page 2162](#).

NMISS

is a descriptive statistic keyword. For details, see “[statistic-keyword\(s\)](#)” on page 2162.

RANGE

is a descriptive statistic keyword. For details, see “[statistic-keyword\(s\)](#)” on page 2162.

SKEWNESS

is a descriptive statistic keyword. For details, see “[statistic-keyword\(s\)](#)” on page 2162.

Alias SKEW

STDDEV

is a descriptive statistic keyword. For details, see “[statistic-keyword\(s\)](#)” on page 2162.

Alias STD

STDERR

is a descriptive statistic keyword. For details, see “[statistic-keyword\(s\)](#)” on page 2162.

SUM

is a descriptive statistic keyword. For details, see “[statistic-keyword\(s\)](#)” on page 2162.

SUMWGT

is a descriptive statistic keyword. For details, see “[statistic-keyword\(s\)](#)” on page 2162.

UCLM

is a descriptive statistic keyword. For details, see “[statistic-keyword\(s\)](#)” on page 2162.

USS

is a descriptive statistic keyword. For details, see “[statistic-keyword\(s\)](#)” on page 2162.

VAR

is a descriptive statistic keyword. For details, see “[statistic-keyword\(s\)](#)” on page 2162.

MEDIAN

is a quantile statistic keyword. For details, see “[statistic-keyword\(s\)](#)” on page 2162.

Alias P50

P1

is a quantile statistic keyword. For details, see “[statistic-keyword\(s\)](#)” on page 2162.

P5

is a quantile statistic keyword. For details, see “[statistic-keyword\(s\)](#)” on page 2162.

P10

is a quantile statistic keyword. For details, see [“*statistic-keyword\(s\)*”](#) on page 2162.

P20

is a quantile statistic keyword. For details, see [“*statistic-keyword\(s\)*”](#) on page 2162.

P40

is a quantile statistic keyword. For details, see [“*statistic-keyword\(s\)*”](#) on page 2162.

P70

is a quantile statistic keyword. For details, see [“*statistic-keyword\(s\)*”](#) on page 2162.

Q1

is a quantile statistic keyword. For details, see [“*statistic-keyword\(s\)*”](#) on page 2162.

Alias P25

Q3

is a quantile statistic keyword. For details, see [“*statistic-keyword\(s\)*”](#) on page 2162.

Alias P75

P90

is a quantile statistic keyword. For details, see [“*statistic-keyword\(s\)*”](#) on page 2162.

P95

is a quantile statistic keyword. For details, see [“*statistic-keyword\(s\)*”](#) on page 2162.

P99

is a quantile statistic keyword. For details, see [“*statistic-keyword\(s\)*”](#) on page 2162.

P30

is a quantile statistic keyword. For details, see [“*statistic-keyword\(s\)*”](#) on page 2162.

P60

is a quantile statistic keyword. For details, see [“*statistic-keyword\(s\)*”](#) on page 2162.

P80

is a quantile statistic keyword. For details, see [“*statistic-keyword\(s\)*”](#) on page 2162.

QRANGE

is a quantile statistic keyword. For details, see [“*statistic-keyword\(s\)*”](#) on page 2162.

PROBT

is a hypothesis testing keyword. For details, see “[statistic-keyword\(s\)](#)” on page 2162.

Alias PBT

T

is a hypothesis testing keyword. For details, see “[statistic-keyword\(s\)](#)” on page 2162.

Details

PRINT | NOPRINT

specifies whether PROC SUMMARY displays the descriptive statistics. By default, PROC SUMMARY does not display output, but PROC MEANS does display output.

Default NOPRINT

ATTRIB Statement

Associates a format, informat, label, and length with one or more variables.

Syntax

ATTRIB *variable-list(s) attribute-list(s);*

Arguments

variable-list(s)

names the variables that you want to associate with the attributes.

TIP List the variables in any form that SAS allows.

attribute-list(s)

specifies one or more attributes to assign to *variable-list*. Specify one or more of these attributes in the ATTRIB statement:

FORMAT=*format*

associates a format with variables in *variable-list*.

Tip The format can be either a standard SAS format or a format that is defined with the FORMAT procedure.

INFORMAT=informal

associates an informat with variables in *variable-list*.

Tip The informat can be either a standard SAS informat or an informat that is defined with the FORMAT procedure.

LABEL='label'

associates a label with variables in *variable-list*.

See Also

[“ATTRIB Statement” in SAS DATA Step Statements: Reference.](#)

BY Statement

Orders the output according to the BY groups.

Restriction: Do not specify a group variable in the BY statement. The BY variable cannot be the same as the group variable. A *group variable* is a variable in the input data set that is used to categorize chart variable values into groups. For more information, see [“Group Variables” on page 1760](#).

Syntax

```
BY <DESCENDING> variable-1
    <... <DESCENDING> variable-n>
    <NOTSORTED>;
```

Required Argument

variable

specifies the variable that the procedure uses to form BY groups. You can specify more than one variable. If you do not use the NOTSORTED option in the BY statement, then either the observations in the data set must be sorted by all the variables that you specify, or they must be indexed appropriately. Variables in a BY statement are called *BY variables*.

Optional Arguments

DESCENDING

specifies that the observations are sorted in descending order by the variable that immediately follows the word DESCENDING in the BY statement.

NOTSORTED

specifies that observations are not necessarily sorted in alphabetic or numeric order. The observations are grouped in another way (for example, chronological order).

The requirement for ordering or indexing observations according to the values of BY variables is suspended for BY-group processing when you use the NOTSORTED option. In fact, the procedure does not use an index if you specify NOTSORTED. The procedure defines a BY group as a set of contiguous observations that have the same values for all BY variables. If observations with the same values for the BY variables are not contiguous, then the procedure treats each contiguous set as a separate BY group.

You cannot use the NOTSORTED option in a PROC SORT step.

Note: You cannot use the GROUPFORMAT option, which is available in the BY statement in a DATA step, in a BY statement in any PROC step.

Details

BY-Group Processing

Procedures create output for each BY group. For example, the elementary statistics procedures and the scoring procedures perform separate analyses for each BY group. The reporting and ODS procedures produce a report for each BY group.

Note: All Base SAS procedures except PROC PRINT process BY groups independently. PROC PRINT can report the number of observations in each BY group as well as the number of observations in all BY groups. Similarly, PROC PRINT can sum numeric variables in each BY group and across all BY groups.

You can use only one BY statement in each PROC step. When you use a BY statement, the procedure expects an input data set that is sorted by the order of the BY variables or one that has an appropriate index. If your input data set does not meet these criteria, then an error occurs. Either sort it with the SORT procedure or create an appropriate index on the BY variables.

Depending on the order of your data, you might need to use the NOTSORTED or DESCENDING option in the BY statement in the PROC step.

- For more information about the BY statement, see [SAS DATA Step Statements: Reference](#).
- For more information about PROC SORT, see [Chapter 55, “SORT Procedure,” on page 2039](#).
- For more information about BY-group processing in CAS, see [“Grouping and Ordering” in SAS Cloud Analytic Services: DATA Step Programming](#).
- For more information about creating indexes, see [“INDEX CREATE Statement” on page 440](#).

Formatting BY-Variable Values

When a procedure is submitted with a BY statement, the following actions are taken with respect to processing of BY groups:

- 1 The procedure determines whether the data is sorted by the internal (unformatted) values of the BY variable(s).
- 2 The procedure determines whether a format has been applied to the BY variable(s). If the BY variable is numeric and has no user-applied format, then the BEST12. format is applied for the purpose of BY-group processing.
- 3 The procedure continues adding observations to the current BY group until both the internal and formatted values of the BY variable or variables change.

This process can have unexpected results if, for example, nonconsecutive internal BY values share the same formatted value. In this case, the formatted value is represented in different BY groups. Alternatively, if different consecutive internal BY values share the same formatted value, then these observations are grouped into the same BY group.

BY Variables That Have Different Lengths in Two Data Sets

When a procedure has a BY statement and two input data sets, one of the input data sets is called the primary data set and the other is called the secondary data set. The primary data set is usually, but not always, the DATA= data set. A BY statement always applies to the primary data set. The variables in the BY statement must appear in the primary data set.

Each procedure determines whether a BY statement applies to a secondary data set, and performs one of the following actions:

- The procedure might always apply the BY statement to the secondary data set. In this case, one or more variables in the BY statement must appear in the secondary data set.
- The procedure might never apply the BY statement to the secondary data set. In this case, the variables in the BY statement are not required in the secondary data set.
- The procedure might check whether the BY variables are in the secondary data set. If none of the BY variables are in the secondary data set, then BY processing does not occur for the secondary data set. If one or more of the BY variables are in the secondary data set, and they match the BY variables in the primary data set, then BY processing is done for the secondary data set. If some but not all BY variables are in the secondary data set, then the procedure might issue an error message and quit. Or, it might take some other action described in the documentation for that particular procedure.

If the BY statement is applied to the secondary data set, then each BY variable that exists on both the data sets must have the same type, character or numeric, in both data sets. The BY variables are required to have either the same formatted value or the same unformatted value. Formatted values match only if both the formatted lengths and the formatted values are the same. Unformatted values are not required to have the same length in order to match. The unformatted character

values match if the unformatted values are the same after stripping the trailing blanks. The unformatted doubles match if they have the same value.

A secondary data set does not need to have all of the BY variables that are in the primary data set. A procedure can define a subset of the BY variables for the secondary data set. For example, if the primary data set has the BY variables A,B,C,D, then the procedure can define the following BY variables on the secondary data set:

- A
- A,B
- A,B,C
- A,B,C,D

If both the primary and secondary data sets have the same number of BY variables, and all the BY variables have the same byte lengths and format lengths, then either the unformatted values or the formatted values in the BY buffer (for all of the BY variables) have to match. If they do not match, then each variable is compared. The formatted values of each variable are compared first. The formatted lengths have to match, and the formatted values have to match. If the formatted lengths and values do not match, then the unformatted values are compared even if the byte lengths are different.

If corresponding character variable lengths differ, then the longer character variable can contain only trailing blanks for the extra characters. If the lengths of the character variables are different, then the values match as long as they are the same after stripping the trailing blanks. For example, 'ABCD' in the primary data set matches 'ABCD ' in the secondary data set. If the secondary data set contained 'ABCDEF', then they would not match.

Base SAS Procedures That Support the BY Statement

COMPARE	SORT (required)
CORR	STANDARD
FREQ	SUMMARY
MEANS	TABULATE
PRINT	TRANSPOSE
RANK	UNIVARIATE
REPORT (nonwindowing environment only)	ODSTEXT
ODSLIST	ODSTABLE

Note: In the SORT procedure, the BY statement specifies how to sort the data. In the other procedures, the BY statement specifies how the data is currently sorted.

See Also

[“BY Statement” in SAS DATA Step Statements: Reference](#)

CLASS Statement

Specifies the variables whose values define the subgroup combinations for the analysis.

Alias: CLASSES

Note: CLASS statements without options use ORDER=INTERNAL, which is the default, or the value specified by the ORDER= option in the PROC MEANS statement. For example, in the following code, variables c and d would use ORDER=INTERNAL. If an ORDER= option had been specified in the PROC MEANS statement, then variables c and d would use the value specified by the ORDER= option in the PROC MEANS statement.

```
class a b / order=data;
class c d;
```

Tips: You can use multiple CLASS statements.

Some CLASS statement options are also available in the PROC MEANS statement. They affect all CLASS variables. Options that you specify in a CLASS statement apply only to the variables in that CLASS statement.

See: For information about how the CLASS statement groups formatted values, see [“Formatted Values” on page 62](#).

Examples: [“Example 2: Computing Descriptive Statistics with Class Variables” on page 1301](#)
[“Example 3: Using the BY Statement with Class Variables” on page 1304](#)
[“Example 4: Using a CLASSDATA= Data Set with Class Variables” on page 1306](#)
[“Example 5: Using Multilabel Value Formats with Class Variables” on page 1309](#)
[“Example 6: Using Preloaded Formats with Class Variables” on page 1314](#)
[“Example 7: Computing a Confidence Limit for the Mean” on page 1318](#)
[“Example 8: Computing Output Statistics” on page 1320](#)
[“Example 9: Computing Different Output Statistics for Several Variables” on page 1322](#)
[“Example 10: Computing Output Statistics with Missing Class Variable Values” on page 1325](#)
[“Example 11: Identifying an Extreme Value with the Output Statistics” on page 1327](#)
[“Example 12: Identifying the Top Three Extreme Values with the Output Statistics” on page 1330](#)

Syntax

CLASS *variable(s)* *</ options>*;

Required Argument

variable(s)

specifies one or more variables that the procedure uses to group the data. Variables in a CLASS statement are referred to as *class variables*. Class variables are numeric or character. Class variables can have continuous values, but they typically have a few discrete values that define levels of the variable. You do not have to sort the data by class variables.

Interaction Use the TYPES statement or the WAYS statement to control which class variables PROC MEANS uses to group the data.

Tip To reduce the number of class variable levels, use a FORMAT statement to combine variable values. When a format combines several internal values into one formatted value, PROC MEANS outputs the lowest internal value.

See [“Using Class Variables ” on page 1231](#)

Optional Arguments

ASCENDING

specifies to sort the class variable levels in ascending order.

Alias ASCEND

Interaction PROC MEANS issues a warning message if you specify both ASCENDING and DESCENDING and ignores both options.

See [“Example 10: Computing Output Statistics with Missing Class Variable Values” on page 1325](#)

DESCENDING

specifies to sort the class variable levels in descending order.

Alias DESCEND

Interaction PROC MEANS issues a warning message if you specify both ASCENDING and DESCENDING and ignores both options.

EXCLUSIVE

excludes from the analysis all combinations of the class variables that are not found in the preloaded range of user-defined formats.

Requirement You must specify PRELOADFMT to preload the class variable formats.

See [“Example 6: Using Preloaded Formats with Class Variables” on page 1314](#)

GROUPINTERNAL

specifies not to apply formats to the class variables when PROC MEANS groups the values to create combinations of class variables.

Interactions If you specify the PRELOADFMT option, then PROC MEANS ignores the GROUPINTERNAL option and uses the formatted values.

If you specify the ORDER=FORMATTED option, then PROC MEANS ignores the GROUPINTERNAL option and uses the formatted values.

Tip This option saves computer resources when the numeric class variables contain discrete values.

See [“Computer Resources” on page 2178](#)

MISSING

considers missing values as valid values for the class variable levels. Special missing values that represent numeric values (the letters A through Z and the underscore (_) character) are each considered as a separate value.

Default If you omit MISSING, then PROC MEANS excludes the observations with a missing class variable value from the analysis.

See *SAS Programmer's Guide: Essentials* for a discussion of missing values with special meanings.

[“Example 10: Computing Output Statistics with Missing Class Variable Values” on page 1325](#)

MLF

enables PROC MEANS to use the primary and secondary format labels for a given range or overlapping ranges to create subgroup combinations when a multilabel format is assigned to a class variable.

Requirement You must use PROC FORMAT and the MULTILABEL option in the VALUE statement to create a multilabel format.

Interactions If you use the OUTPUT statement with MLF, then the class variable contains a character string that corresponds to the formatted value. Because the formatted value becomes the internal value, the length of this variable is the number of characters in the longest format label.

Using MLF with ORDER=FREQ might not produce the order that you expect for the formatted values. You might not get the expected results when you use MLF with CLASSDATA and EXCLUSIVE because MLF processing requires that each TYPE be computed independently. Types other than NWAY might contain more levels than expected.

Note When the formatted values overlap, one internal class variable value maps to more than one class variable subgroup combination. Therefore, the sum of the N statistics for all subgroups is greater

than the number of observations in the data set (the overall N statistic).

Tip If you omit MLF, then PROC MEANS uses the primary format labels. This action corresponds to using the first external format value to determine the subgroup combinations.

See The MULTILABEL option in the VALUE statement of the FORMAT procedure [“Optional Arguments” on page 889](#).

[“Example 5: Using Multilabel Value Formats with Class Variables” on page 1309](#)

ORDER=DATA | FORMATTED | FREQ | UNFORMATTED

specifies the order to group the levels of the class variables in the output, where

DATA

orders values according to their order in the input data set.

Interaction If you use PRELOADFMT, then the order of the values of each class variable matches the order that PROC FORMAT uses to store the values of the associated user-defined format. If you use the CLASSDATA= option in the PROC statement, then PROC MEANS uses the order of the unique values of each class variable in the CLASSDATA= data set to order the output levels. If you use both options, then PROC MEANS first uses the user-defined formats to order the output. If you omit EXCLUSIVE in the PROC statement, then PROC MEANS appends after the user-defined format and the CLASSDATA= values the unique values of the class variables in the input data set based on the order in which they are encountered.

Tip By default, PROC FORMAT stores a format definition in sorted order. Use the NOTSORTED option to store the values or ranges of a user-defined format in the order in which you define them.

See [“Example 10: Computing Output Statistics with Missing Class Variable Values” on page 1325](#)

FORMATTED

orders values by their ascending formatted values. This order depends on your operating environment. If no format has been assigned to a class variable, then the default format, BEST12., is used.

Aliases FMT

EXTERNAL

See [“Example 5: Using Multilabel Value Formats with Class Variables” on page 1309](#)

FREQ

orders values by descending frequency count so that levels with the most observations are listed first.

Interactions For multiway combinations of the class variables, PROC MEANS determines the order of a level from the individual class variable frequencies.

Use the ASCENDING option to order values by ascending frequency count.

See [“Example 5: Using Multilabel Value Formats with Class Variables” on page 1309](#)

UNFORMATTED

orders values by their unformatted values, which yields the same order as PROC SORT. This order depends on your operating environment. This sort sequence is particularly useful for displaying dates chronologically.

Aliases UNFMT

INTERNAL

Default UNFORMATTED

Tip By default, all orders except FREQ are ascending. For descending orders, use the DESCENDING option.

See [“Ordering the Class Values” on page 1232](#)

PRELOADFMT

specifies that all formats are preloaded for the class variables.

Requirement PRELOADFMT has no effect unless you specify either COMPLETETYPES, EXCLUSIVE, or ORDER=DATA and you assign formats to the class variables.

Interactions The combination of PRELOADFMT and COMPLETETYPES generates values that might not appear within the input data set. In some cases, for a character class variable, extremely low or high sentinel values are generated. These values might not represent valid characters of the session encoding. These values should not be printed in their raw form but should be printed using the associated format. Use of the OTHER= value in the format definition can prevent the raw sentinel values from printing when using the format.

To limit PROC MEANS output to the combinations of formatted class variable values present in the input data set, use the EXCLUSIVE option in the CLASS statement.

To include all ranges and values of the user-defined formats in the output, even when the frequency is zero, use COMPLETETYPES in the PROC statement.

Note When preloading a character format, there must be at least one representative raw value that produces the label. If a range does not have any raw values, the range is considered unreachable and an error is generated. For example, you cannot preload a multi-

label character format with both Other= and Low-High ranges. Other is a special keyword that refers to values or ranges that remain after all labels have been defined. For character formats, missing values are included in the low range so that there are no raw values left to occupy the Other range.

See [“Example 6: Using Preloaded Formats with Class Variables” on page 1314](#)

Details

Comparison of the BY and CLASS Statements

Using the BY statement is similar to using the CLASS statement and the NWAY option in that PROC MEANS summarizes each BY group as an independent subset of the input data. Therefore, no overall summarization of the input data is available. However, unlike the CLASS statement, the BY statement requires that the data is pre-sorted or indexed by the variables in the BY statement.

When you use the NWAY option, PROC MEANS might encounter insufficient memory for the summarization of all the class variables. You can move some class variables to the BY statement. For maximum benefit, move class variables to the BY statement that are already sorted or that have the greatest number of unique values.

You can use the CLASS and BY statements together to analyze the data by the levels of class variables within BY groups. See [“Example 3: Using the BY Statement with Class Variables” on page 1304](#).

How PROC MEANS Handles Missing Values for Class Variables

By default, if an observation contains a missing value for any class variable, then PROC MEANS excludes that observation from the analysis. If you specify the MISSING option in the PROC statement, then the procedure considers missing values as valid levels for the combination of class variables.

Specifying the MISSING option in the CLASS statement enables you to control the acceptance of missing values for individual class variables.

Computer Resources

The total of unique class variable values that PROC MEANS allows depends on the amount of computer memory that is available. See [“Computational Resources” on page 1233](#) for more information.

The GROUPINTERNAL option can improve computer performance because the grouping process is based on the internal values of the class variables. If a numeric class variable is not assigned a format and you do not specify GROUPINTERNAL, then PROC MEANS uses the default format, BEST12., to format numeric values as character strings. Then PROC MEANS groups these numeric variables by their character values, which takes additional time and computer memory.

FORMAT Statement

Associates formats with variables.

Syntax

FORMAT *variable-1* <...*variable-n*> *format variable-1* <...*variable-n*> *format*;

Arguments

variable

names one or more variables for SAS to associate with a format. You must specify at least one *variable*.

Tip To disassociate a format from a variable, use the variable in a FORMAT statement without specifying a format in a DATA step or in PROC DATASETS. In a DATA step, place this FORMAT statement after the SET statement. See [“Removing a Format” in SAS DATA Step Statements: Reference](#). You can also use PROC DATASETS.

format

specifies the format that is listed for writing the values of the variables.

Tip Formats that are associated with variables by using a FORMAT statement behave like formats that are used with a colon modifier in a subsequent PUT statement. For more information about using a colon modifier, see [“PUT Statement: List” in SAS DATA Step Statements: Reference](#).

See [SAS Formats and Informats: Reference](#)

FREQ Statement

Specifies a numeric variable that contains the frequency of each observation.

Alias: FREQUENCY

Syntax

FREQ *variable*;

Required Argument

variable

specifies a numeric variable whose value represents the frequency of the observation. If you use the FREQ statement, then the procedure assumes that each observation represents n observations, where n is the value of *variable*. If n is not an integer, then SAS truncates it. If n is less than 1 or is missing, then the procedure does not use that observation to calculate statistics.

The sum of the frequency variable represents the total number of observations.

Note: The FREQ variable does not affect how PROC MEANS identifies multiple extremes when you use the IDGROUP syntax in the OUTPUT statement.

ID Statement

Includes additional variables in the output data set.

See: [Discussion of id-group-specification in “OUTPUT Statement” on page 2183.](#)

Syntax

ID *variable(s)*;

Required Argument

variable(s)

identifies one or more variables from the input data set whose maximum values for groups of observations PROC MEANS includes in the output data set.

Interaction Use IDMIN in the PROC statement to include the minimum value of the ID variables in the output data set.

Tip Use the PRINTIDVARS option in the PROC statement to include the value of the ID variable in the displayed output.

Details

Selecting the Values of the ID Variables

When you specify only one variable in the ID statement, the value of the ID variable for a given observation is the maximum (minimum) value found in the corresponding group of observations in the input data set. When you specify multiple variables in the ID statement, PROC MEANS selects the maximum value

by processing the variables in the ID statement in the order in which you list them. PROC MEANS determines which observation to use from all the ID variables by comparing the values of the first ID variable. If more than one observation contains the same maximum (minimum) ID value, then PROC MEANS uses the second and subsequent ID variable values as “tiebreakers.” In any case, all ID values are taken from the same observation for any given BY group or classification level within a type.

See [“Sorting Orders for Character Variables” on page 2043](#) for information about how PROC MEANS compares character values to determine the maximum value.

LABEL Statement

Assigns descriptive labels to variables.

Syntax

LABEL *variable-1* = '*text-string*' <...*variable-n* = '*text-string*'>;

LABEL *variable-1* = '' <...*variable-n*=''>; | *variable-1* = <...*variable-n*=>;

Arguments

variable

specifies the variable that you want to label. You can specify labels for multiple variables in a single LABEL statement:

```
data test;
  L = 5;
  W = 10;
  label L = 'Length' W = 'Width';
run;
```

text-string

specifies a label of up to 256 bytes.

```
data test;
  L = 5;
  label L = 'Length';
run;
```

Restrictions If the label includes a semicolon (;) or an equal sign (=), you must enclose the label in either single or double quotation marks.

```
label N = 'Number = ';
```

If the label includes single quotation marks ('), you must enclose the label in double quotation marks.

```
label Name = "Owner's Last Name";
```

There is partial support for variable name substitution with the JAVAIMG device. When the label text for the variable exceeds the space allowed, the variable name is used instead by procedures that are labeling an axis or a legend title. With the exception of the SAS/GRAPH GPLOT procedure, the variable name is not used when the JAVAIMG device is specified.

Notes

Quotation marks are not required unless the label includes a semicolon, quotation marks, or an equal sign.

The LABEL statement expects an equal sign (=) after the first variable. If any other character is found, an error occurs. The LABEL statement considers everything after the equal sign, following the first variable, as part of the label until the statement comes to another variable followed by an equal sign, regardless of quotation marks. In this example, the label for x is **this is x y 4 = this is y** because the next variable that is followed by an equal sign is z.

```
data test;
  x = 1;
  y = 1;
  z = 1;
  label x = 'this is x' y 4 = 'this is y' z = 'This is z';
run;
```

In this LABEL statement, the x variable has the same label:

```
label x = this is x y 4 = this is y z = This is z;
```

See

[“Character Constants and Character Variables” in SAS Programmer’s Guide: Essentials.](#)

”

removes a label from a variable. Enclose a single blank space in quotation marks to remove an existing label. You can remove labels from multiple variables in a single LABEL statement:

```
data test;
  L=5; W=10;
  label L = ' ' W = ' ';
run;
```

Alternatively, you can remove a label by specifying nothing after the equal sign:

```
data test;
  L=5; W=10;
  label L = W = ;
run;
```

OUTPUT Statement

Writes statistics to a new SAS data set.

Tip: You can use multiple OUTPUT statements to create several OUT= data sets.

Examples: [“Example 8: Computing Output Statistics” on page 1320](#)
[“Example 9: Computing Different Output Statistics for Several Variables” on page 1322](#)
[“Example 10: Computing Output Statistics with Missing Class Variable Values” on page 1325](#)
[“Example 11: Identifying an Extreme Value with the Output Statistics” on page 1327](#)
[“Example 12: Identifying the Top Three Extreme Values with the Output Statistics” on page 1330](#)

Syntax

```
OUTPUT <OUT=SAS-data-set>
<statistic-specifications>
<id-group-specifications> <maximum-id-specifications>
<minimum-id-specifications> </ options>;
```

Optional Arguments

OUT=SAS-data-set

names the new output data set. If SAS-data-set does not exist, then PROC MEANS creates it. If you omit OUT=, then the data set is named DATA n , where n is the smallest integer that makes the name unique.

Default DATA n

Tip You can use data set options with the OUT= option.

statistic-keyword<(variable-list)><=name(s)>

specifies the statistics to store in the OUT= data set and names one or more variables that contain the statistics.

statistic-keyword

specifies which statistic to store in the output data set.

The available statistic keywords are listed in [“Statistic Keywords” on page 2190](#).

By default the statistics in the output data set automatically inherit the analysis variable's format, informat, and label. However, statistics computed for N, NMISS, SUMWGT, USS, CSS, VAR, CV, T, PROBT, PRT, SKEWNESS, and KURTOSIS do not inherit the analysis variable's format because this

format might be invalid for these statistics (for example, dollar or datetime formats).

Default N, MEAN, STD, MIN, and MAX

Restriction If you omit *variable* and *name(s)*, then PROC MEANS allows the *statistic-keyword* only once in a single OUTPUT statement, unless you also use the AUTONAME option.

See The definitions of the keywords and the formulas for the associated statistics are listed in [“Keywords and Formulas” on page 2426](#).

[“Example 8: Computing Output Statistics” on page 1320](#)

[“Example 9: Computing Different Output Statistics for Several Variables” on page 1322](#)

[“Example 11: Identifying an Extreme Value with the Output Statistics” on page 1327](#)

[“Example 12: Identifying the Top Three Extreme Values with the Output Statistics” on page 1330](#)

variable-list

specifies the names of one or more numeric analysis variables whose statistics you want to store in the output data set.

Default all numeric analysis variables

names

specifies one or more names for the variables in the output data set that contain the analysis variable statistics. The first name contains the statistic for the first analysis variable; the second name contains the statistic for the second analysis variable; and so on.

Default the analysis variable name. If you specify AUTONAME, then the default is the combination of the analysis variable name and the *statistic-keyword*. If you use the CLASS statement and an OUTPUT statement without an *output-statistic-specification*, then the output data set contains five observations for each combination of class variables: the value of N, MIN, MAX, MEAN, and STD. If you use the WEIGHT statement or the WEIGHT option in the VAR statement, then the output data set also contains an observation with the sum of weights (SUMWGT) for each combination of class variables.

Interaction If you specify *variable-list*, then PROC MEANS uses the order in which you specify the analysis variables to store the statistics in the output data set variables.

Tip Use the AUTONAME option to have PROC MEANS generate unique names for multiple variables and statistics.

See [“Example 8: Computing Output Statistics” on page 1320](#)

IDGROUP (<MIN (*variable-list-1*) | MAX (*variable-list-1*) <...MIN (*variable-list-n*) | MAX (*variable-list-n*) >> <MISSING> <OBS> <LAST> OUT <[*n*]> (*id-variable-list*)=<*names*>)

combines the features and extends the ID statement, the IDMIN option in the PROC statement, and the MAXID and MINID options in the OUTPUT statement to create an OUT= data set that identifies multiple extreme values.

LAST

specifies that the OUT= data set contains values from the last observation (or the last *n* observations, if *n* is specified).

If you do not specify LAST, then the OUT= data set contains values from the first observation (or the first *n* observations, if *n* is specified). The OUT= data set might contain several observations because in addition to the value of the last (first) observation, the OUT= data set contains values from the last (first) observation of each subgroup level that is defined by combinations of class variable values.

Interaction When you specify MIN or MAX and when multiple observations contain the same extreme values, PROC MEANS uses the observation number to resolve which observation to save to the OUT= data set. If you specify LAST, then PROC MEANS uses the later observations to resolve any ties. If you do not specify LAST, then PROC MEANS uses the earlier observations to resolve any ties.

MAX(*variable-list*)

specifies the selection criteria to determine the maximum extreme values of one or more input data set variables specified in *variable-list*.

When you specify multiple selection variables, the ordering of observations for the selection of *n* extremes is done the same way that PROC SORT sorts data with multiple BY variables. PROC MEANS concatenates the variable values into a single key. The MAX(*variable-list*) selection criterion is similar to using PROC SORT and the DESCENDING option in the BY statement.

Default If you do not specify MIN or MAX, then PROC MEANS uses the observation number as the selection criterion to output observations.

Restriction If you specify criteria that are contradictory, then PROC MEANS uses only the first selection criterion.

Interaction When multiple observations contain the same extreme values in all the MIN or MAX variables, PROC MEANS uses the observation number to resolve which observation to write to the output. By default, PROC MEANS uses the first observation to resolve any ties. However, if you specify the LAST option, then PROC MEANS uses the last observation to resolve any ties.

MIN(*variable-list*)

specifies the selection criteria to determine the minimum extreme values of one or more input data set variables specified in *variable-list*.

When you specify multiple selection variables, the ordering of observations for the selection of n extremes is done the same way that PROC SORT sorts data with multiple BY variables. PROC MEANS concatenates the variable values into a single key.

- Default** If you do not specify MIN or MAX, then PROC MEANS uses the observation number as the selection criterion to output observations.
- Restriction** If you specify criteria that are contradictory, then PROC MEANS uses only the first selection criterion.
- Interaction** When multiple observations contain the same extreme values in all the MIN or MAX variables, PROC MEANS uses the observation number to resolve which observation to write to the output. By default, PROC MEANS uses the first observation to resolve any ties. However, if you specify the LAST option, then PROC MEANS uses the last observation to resolve any ties.

MISSING

specifies that missing values be used in selection criteria.

Alias MISS

OBS

includes an _OBS_ variable in the OUT= data set that contains the number of the observation in the input data set where the extreme value was found.

Interactions If you use WHERE processing, then the value of _OBS_ might not correspond to the location of the observation in the input data set.

If you use $[n]$ to write multiple extreme values to the output, then PROC MEANS creates n _OBS_ variables and uses the suffix n to create the variable names, where n is a sequential integer from 1 to n .

$[n]$

specifies the number of extreme values for each variable in *id-variable-list* to include in the OUT= data set. PROC MEANS creates n new variables and uses the suffix $_n$ to create the variable names, where n is a sequential integer from 1 to n .

By default, PROC MEANS determines one extreme value for each level of each requested type. If n is greater than one, then n extremes are generated for each level of each type. When n is greater than one and you request extreme value selection, the time complexity is $O(T * N \log_2 n)$, where T is the number of types requested and N is the number of observations in the input data set. By comparison, to group the entire data set, the time complexity is $O(N \log_2 N)$.

Default 1

Range an integer between 1 and 100

Example For example, to generate two minimum extreme values for each variable, use

```
idgroup(min(x) out[2] (x y z)=MinX MinY MinZ);
```

The OUT= data set contains the variables MinX_1, MinX_2, MinY_1, MinY_2, MinZ_1, and MinZ_2.

(*id-variable-list*)

identifies one or more input data set variables whose values PROC MEANS includes in the OUT= data set. PROC MEANS determines which observations to generate by the selection criteria that you specify (MIN, MAX, and LAST).

Alias IDGRP

Requirement You must specify the MIN | MAX selection criteria first and OUT(*id-variable-list*)= after the suboptions MISSING, OBS, and LAST.

Tips You can use *id-group-specification* to mimic the behavior of the ID statement and a *maximum-id-specification* or *minimum-id-specification* in the OUTPUT statement.

When you want the output data set to contain extreme values along with other ID variables, it is more efficient to include them in the *id-variable-list* than to request separate statistics. For example, the statement `output idgrp(max(x) out(x a b)= );` is more efficient than the statement `output idgrp(max(x) out(a b)= ) max(x)=;`

See [“Example 8: Computing Output Statistics” on page 1320](#)
[“Example 12: Identifying the Top Three Extreme Values with the Output Statistics” on page 1330](#)

names

specifies one or more names for variables in the OUT= data set.

Default If you omit *name*, then PROC MEANS uses the names of variables in the *id-variable-list*.

Tip Use the AUTONAME option to automatically resolve naming conflicts.

CAUTION

The IDGROUP syntax enables you to create output variables with the same name. When this action happens, only the first variable appears in the output data set. Use the AUTONAME option to automatically resolve these naming conflicts.

Note: If you specify fewer new variable names than the combination of analysis variables and identification variables, then the remaining output variables use

the corresponding names of the ID variables as soon as PROC MEANS exhausts the list of new variable names.

Alias IDGRP

MAXID *<(variable-1 <(id-variable-list-1)> <...variable-n <(id-variable-list-n)>>> = names*

specifies that one or more identification variables be associated with the maximum values of the analysis variables.

variable

identifies the numeric analysis variable whose maximum values PROC MEANS determines. PROC MEANS can determine several maximum values for a variable because, in addition to the overall maximum value, subgroup levels, which are defined by combinations of class variables values, also have maximum values.

Tip If you use an ID statement and omit *variable*, then PROC MEANS uses all analysis variables.

id-variable-list

identifies one or more variables whose values identify the observations with the maximum values of the analysis variable.

Default the ID statement variables

names

specifies the names for new variables that contain the values of the identification variable associated with the maximum value of each analysis variable.

Note If multiple observations contain the maximum value within a class level, then PROC MEANS saves the value of the ID variable for only the first of those observations in the output data set.

Tips If you use an ID statement, and omit *variable* and *id-variable*, then PROC MEANS associates all ID statement variables with each analysis variable. Thus, for each analysis variable, the number of variables that are created in the output data set equals the number of variables that you specify in the ID statement.

Use the AUTONAME option to automatically resolve naming conflicts.

See [“Example 11: Identifying an Extreme Value with the Output Statistics” on page 1327](#)

CAUTION

The MAXID syntax enables you to create output variables with the same name. When this action happens, only the first variable appears in the output data set. Use the AUTONAME option to automatically resolve these naming conflicts.

Note: If you specify fewer new variable names than the combination of analysis variables and identification variables, then the remaining output variables use the corresponding names of the ID variables as soon as PROC MEANS exhausts the list of new variable names.

MINID<(variable-1 <(id-variable-list-1)> <...variable-n <(id-variable-list-n)>>> = name(s)

see the description of maximum-id-specification. This option behaves in exactly the same way, except that PROC MEANS determines the minimum values instead of the maximum values.

When MINID is used without an explicit variable list, it is similar to the following more advanced IDGROUP syntax example:

```
IDGRP( min(x) missing out(id_variable)=idminx) idgrp( min(y) missing
out(id_variable)=idminy)
```

If one or more of the analysis variables has a missing value, the id_variable value corresponds to the observation with the missing value not the observation with the value for the MIN statistic.

Options to Use after a Slash Delimiter

AUTOLABEL

specifies that PROC MEANS appends the statistic name to the end of the variable label. If an analysis variable has no label, then PROC MEANS creates a label by appending the statistic name to the analysis variable name.

See [“Example 12: Identifying the Top Three Extreme Values with the Output Statistics” on page 1330](#)

AUTONAME

specifies that PROC MEANS creates a unique variable name for an output statistic when you do not assign the variable name in the OUTPUT statement. This action is accomplished by appending to the *statistic-keyword* to the end of the input variable name from which the statistic was derived.

For example, the statement `output min(x)=/autoname;` produces the x_Min variable in the output data set.

AUTONAME activates the SAS internal mechanism to automatically resolve conflicts in the variable names in the output data set. Duplicate variables do not generate errors.

As a result, the statement `output min(x)= min(x)=/autoname;` produces two variables, x_Min and x_Min2, in the output data set.

If the new variable name exceeds 32 characters, then the variable-name portion is truncated.

See [“Example 12: Identifying the Top Three Extreme Values with the Output Statistics” on page 1330](#)

KEEPLEN

specifies that statistics in the output data set inherit the length of the analysis variable that PROC MEANS uses to derive them.

CAUTION

You permanently lose numeric precision when the length of the analysis variable causes PROC MEANS to truncate or round the value of the statistic. However, the precision of the statistic matches that of the input.

LEVELS

includes a variable named `_LEVEL_` in the output data set. This variable contains a value from 1 to n that indicates a unique combination of the values of class variables (the values of `_TYPE_` variable).

See [“Output Data Set” on page 1297](#)

[“Example 8: Computing Output Statistics” on page 1320](#)

NOINHERIT

specifies that the variables in the output data set that contain statistics do not inherit the attributes (label and format) of the analysis variables that are used to derive them.

Interaction When no option is used (implied INHERIT) then the statistics inherit the attributes, label and format, of the input analysis variable(s). If the INHERIT option is used in the OUTPUT statement, then the statistics inherit the length of the input analysis variable(s), the label and format.

Tip By default, the output data set includes an output variable for each analysis variable and for five observations that contain N, MIN, MAX, MEAN, and STDDEV. Unless you specify NOINHERIT, this variable inherits the format of the analysis variable, which can be invalid for the N statistic (for example, datetime formats).

WAYS

includes a variable named `_WAY_` in the output data set. This variable contains a value from 1 to the maximum number of class variables that indicates how many class variables PROC MEANS combines to create the TYPE value.

See [“Output Data Set” on page 1297](#)

[“WAYS Statement” on page 2195](#)

[“Example 8: Computing Output Statistics” on page 1320](#)

Statistic Keywords

CLM

is a descriptive statistic keyword. For details, see [“*statistic-keyword*<\(variable-list\)><=name\(s\)>” on page 2183](#).

CSS

is a descriptive statistic keyword. For details, see “*statistic-keyword<(variable-list)><=name(s)>*” on page 2183.

KURTOSIS

is a descriptive statistic keyword. For details, see “*statistic-keyword<(variable-list)><=name(s)>*” on page 2183.

Alias KURT

LCLM

is a descriptive statistic keyword. For details, see “*statistic-keyword<(variable-list)><=name(s)>*” on page 2183.

MAX

is a descriptive statistic keyword. For details, see “*statistic-keyword<(variable-list)><=name(s)>*” on page 2183.

MEAN

is a descriptive statistic keyword. For details, see “*statistic-keyword<(variable-list)><=name(s)>*” on page 2183.

MIN

is a descriptive statistic keyword. For details, see “*statistic-keyword<(variable-list)><=name(s)>*” on page 2183.

MODE

is a descriptive statistic keyword. For details, see “*statistic-keyword<(variable-list)><=name(s)>*” on page 2183.

N

is a descriptive statistic keyword. For details, see “*statistic-keyword<(variable-list)><=name(s)>*” on page 2183.

NMISS

is a descriptive statistic keyword. For details, see “*statistic-keyword<(variable-list)><=name(s)>*” on page 2183.

RANGE

is a descriptive statistic keyword. For details, see “*statistic-keyword<(variable-list)><=name(s)>*” on page 2183.

SKEWNESS

is a descriptive statistic keyword. For details, see “*statistic-keyword<(variable-list)><=name(s)>*” on page 2183.

Alias SKEW

STDDEV

is a descriptive statistic keyword. For details, see “*statistic-keyword<(variable-list)><=name(s)>*” on page 2183.

Alias STD

STDERR

is a descriptive statistic keyword. For details, see “*statistic-keyword<(variable-list)><=name(s)>*” on page 2183.

SUM

is a descriptive statistic keyword. For details, see “*statistic-keyword<(variable-list)><=name(s)>*” on page 2183.

SUMWGT

is a descriptive statistic keyword. For details, see “*statistic-keyword<(variable-list)><=name(s)>*” on page 2183.

UCLM

is a descriptive statistic keyword. For details, see “*statistic-keyword<(variable-list)><=name(s)>*” on page 2183.

USS

is a descriptive statistic keyword. For details, see “*statistic-keyword<(variable-list)><=name(s)>*” on page 2183.

VAR

is a descriptive statistic keyword. For details, see “*statistic-keyword<(variable-list)><=name(s)>*” on page 2183.

MEDIAN

is a quantile statistic keyword. For details, see “*statistic-keyword<(variable-list)><=name(s)>*” on page 2183.

Alias P50

P1

is a quantile statistic keyword. For details, see “*statistic-keyword<(variable-list)><=name(s)>*” on page 2183.

P5

is a quantile statistic keyword. For details, see “*statistic-keyword<(variable-list)><=name(s)>*” on page 2183.

P10

is a quantile statistic keyword. For details, see “*statistic-keyword<(variable-list)><=name(s)>*” on page 2183.

P20

is a quantile statistic keyword. For details, see “*statistic-keyword<(variable-list)><=name(s)>*” on page 2183.

P40

is a quantile statistic keyword. For details, see “*statistic-keyword<(variable-list)><=name(s)>*” on page 2183.

P70

is a quantile statistic keyword. For details, see “*statistic-keyword<(variable-list)><=name(s)>*” on page 2183.

Q1

is a quantile statistic keyword. For details, see “*statistic-keyword<(variable-list)><=name(s)>*” on page 2183.

Alias P25

Q3

is a quantile statistic keyword. For details, see [“*statistic-keyword*<\(variable-list\)><=name\(s\)>” on page 2183](#).

Alias P75

P90

is a quantile statistic keyword. For details, see [“*statistic-keyword*<\(variable-list\)><=name\(s\)>” on page 2183](#).

P95

is a quantile statistic keyword. For details, see [“*statistic-keyword*<\(variable-list\)><=name\(s\)>” on page 2183](#).

P99

is a quantile statistic keyword. For details, see [“*statistic-keyword*<\(variable-list\)><=name\(s\)>” on page 2183](#).

P30

is a quantile statistic keyword. For details, see [“*statistic-keyword*<\(variable-list\)><=name\(s\)>” on page 2183](#).

P60

is a quantile statistic keyword. For details, see [“*statistic-keyword*<\(variable-list\)><=name\(s\)>” on page 2183](#).

P80

is a quantile statistic keyword. For details, see [“*statistic-keyword*<\(variable-list\)><=name\(s\)>” on page 2183](#).

QRANGE

is a quantile statistic keyword. For details, see [“*statistic-keyword*<\(variable-list\)><=name\(s\)>” on page 2183](#).

PROBT

is a hypothesis testing keyword. For details, see [“*statistic-keyword*<\(variable-list\)><=name\(s\)>” on page 2183](#).

Alias PBT

T

is a hypothesis testing keyword. For details, see [“*statistic-keyword*<\(variable-list\)><=name\(s\)>” on page 2183](#).

TYPES Statement

Identifies which of the possible combinations of class variables to generate.

Requirement: CLASS statement

See: [“Output Data Set” on page 1297](#)

Examples: [“Example 2: Computing Descriptive Statistics with Class Variables” on page 1301](#)
[“Example 5: Using Multilabel Value Formats with Class Variables” on page 1309](#)

“Example 12: Identifying the Top Three Extreme Values with the Output Statistics”
on page 1330

Syntax

TYPES *request(s)*;

Required Argument

request(s)

specifies which of the 2^k combinations of class variables PROC MEANS uses to create the types, where k is the number of class variables. A request includes one class variable name, several class variable names separated by asterisks, or ().

To request class variable combinations quickly, use a grouping syntax by placing parentheses around several variables and joining other variables or variable combinations. For example, the following statements illustrate grouping syntax:

Request	Equivalent to
types A*(B C) ;	types A*B A*C;
types (A B)*(C D) ;	types A*C A*D B*C B*D;
types (A B C)*D;	types A*D B*D C*D;

Interaction The CLASSDATA= option places constraints on the NWAY type. PROC MEANS generates all other types as if derived from the resulting NWAY type.

Tips Use () to request the overall total (_TYPE_=0).

If you do not need all types in the output data set, then use the TYPES statement to specify particular subtypes rather than applying a WHERE clause to the data set. Doing so saves time and computer memory.

Details

Order of Analyses in the Output

The analyses are written to the output in order of increasing values of the _TYPE_ variable, which is calculated by PROC MEANS. The _TYPE_ variable has a unique value for each combination of class variables; the values are determined by how

you specify the CLASS statement, not the TYPES statement. Therefore, if you specify

```
class A B C;
types (A B) *C;
```

then the B*C analysis (_TYPE_=3) is written first, followed by the A*C analysis (_TYPE_=5). However, if you specify

```
class B A C;
types (A B) *C;
```

then the A*C analysis comes first.

The _TYPE_ variable is calculated even if no output data set is requested. For more information about the _TYPE_ variable, see [“Output Data Set” on page 1297](#).

VAR Statement

Identifies the analysis variables and their order in the results.

- Default:** If you omit the VAR statement, then PROC SUMMARY produces a simple count of observations, whereas PROC MEANS tries to analyze all the numeric variables that are not listed in the other statements.
- Interaction:** If you specify statistics in the PROC SUMMARY statement and the VAR statement is omitted or if a numeric variable is not associated with a statistic in the OUTPUT statement, then PROC SUMMARY stops processing and an error message is written to the SAS log.
- Note:** See the VAR Statement in PROC MEANS for a full description of the VAR statement.
- See:** For complete syntax, see [“VAR Statement” on page 1282](#).

WAYS Statement

Specifies the number of ways to make unique combinations of class variables.

- Tip:** Use the TYPES statement to specify additional combinations of class variables.
- Example:** [“Example 6: Using Preloaded Formats with Class Variables” on page 1314](#)

Syntax

WAYS *list*;

Required Argument

list

specifies one or more integers that define the number of class variables to combine to form all the unique combinations of class variables. For example, you can specify 2 for all possible pairs and 3 for all possible triples. The *list* can be specified in the following ways:

- *m*
- *m1 m2 ... mn*
- *m1,m2,...,mn*
- *m TO n <BY increment>*
- *m1,m2 TO m3 <BY increment>, m4*

Range 0 to maximum number of class variables

See WAYS option

Example The following code is an example of creating two-way types for the classification variables A, B, and C. This WAYS statement is equivalent to specifying a*b, a*c, and b*c in the TYPES statement.

```
class A B C ; ways 2;
```

WEIGHT Statement

Specifies weights for analysis variables in the statistical calculations. You can use a WEIGHT statement and a FREQ statement in the same step of any procedure that supports both statements.

Alias: WGT

Restrictions: To compute weighted quantiles, use QMETHOD=OS in the PROC statement.
Skewness and kurtosis are not available with the WEIGHT statement.

Syntax

WEIGHT *variable*;

Required Argument

variable

specifies a numeric variable whose values weight the values of the analysis variables. The values of the variable do not have to be integers.

Different behavior for nonpositive values is discussed in the WEIGHT statement syntax under the individual procedure.

You can exclude the observations with missing weights from the analysis. Most SAS/STAT procedures, such as PROC GLM, have always excluded not only missing weights but also negative and zero weights from the analysis. You can achieve this same behavior in a Base SAS procedure that supports the WEIGHT statement by using the EXCLNPWGT option in the PROC statement.

Weight value	Procedure
0	Counts the observation in the total number of observations
Less than 0	Converts the weight value to zero and counts the observation in the total number of observations
Missing	Excludes the observation from the analysis

To exclude observations that contain negative and zero weights from the analysis, use EXCLNPWGT. Note that most SAS/STAT procedures, such as PROC GLM, exclude negative and zero weights by default.

CAUTION

Single extreme weight values can cause inaccurate results. When one (and only one) weight value is many orders of magnitude larger than the other weight values (for example, 49 weight values of 1 and one weight value of 1×10^{14}), certain statistics might not be within acceptable accuracy limits. The affected statistics are based on the second moment (such as standard deviation, corrected sum of squares, variance, and standard error of the mean). Under certain circumstances, no warning is written to the SAS log.

The procedure substitutes the value of the WEIGHT variable for w_i , which appears in [“Keywords and Formulas” on page 2426](#).

Example

Procedures That Support the WEIGHT Statement

- CORR
- FREQ
- MEANS or SUMMARY
- REPORT
- STANDARD

- TABULATE
- UNIVARIATE

Note: In PROC FREQ, the value of the variable in the WEIGHT statement represents the frequency of occurrence for each observation. For more information, see the information about PROC FREQ in the *Base SAS(R) 9.3 Procedures Guide: Statistical Procedures*.

Calculating Weighted Statistics

The procedures that support the WEIGHT statement also support the VARDEF= option, which lets you specify a divisor to use in the calculation of the variance and standard deviation.

By using a WEIGHT statement to compute moments, you assume that the i th observation has a variance that is equal to σ^2/w_i . When you specify VARDEF=DF (the default), the computed variance is a weighted least squares estimate of σ^2 . Similarly, the computed standard deviation is an estimate of σ . Note that the computed variance is not an estimate of the variance of the i th observation, because this variance involves the observation's weight, which varies from observation to observation.

If the values of your variable are counts that represent the number of occurrences of each observation, then use this variable in the FREQ statement rather than in the WEIGHT statement. In this case, because the values are counts, they should be integers. (The FREQ statement truncates any noninteger values.) The variance that is computed with a FREQ variable is an estimate of the common variance σ^2 of the observations.

Note: If your data comes from a stratified sample where the weights w_i represent the strata weights, then neither the WEIGHT statement nor the FREQ statement provides appropriate stratified estimates of the mean, variance, or variance of the mean. To perform the appropriate analysis, consider using PROC SURVEYMEANS, which is a SAS/STAT procedure that is documented in the *Base SAS(R) 9.3 Procedures Guide: Statistical Procedures*.

Use Weighted Statistics

As an example of the WEIGHT statement, suppose 20 people are asked to estimate the size of an object 30 cm wide. Each person is placed at a different distance from the object. As the distance from the object increases, the estimates should become less precise. The SAS data set SIZE contains the estimate (ObjectSize) in centimeters at each distance (Distance) in meters and the precision (Precision) for each estimate. Notice that the largest deviation (an overestimate by 20 cm) came

at the greatest distance (7.5 meters from the object). As a measure of precision, $1/\text{Distance}$, gives more weight to estimates that were made closer to the object and less weight to estimates that were made at greater distances. The PROC MEANS step computes the average estimate of the object size while ignoring the weights. Without a WEIGHT variable, PROC MEANS uses the default weight of 1 for every observation. Thus, the estimates of object size at all distances are given equal weight. The average estimate of the object size exceeds the actual size by 3.55 cm.

```
options nodate pageno=1 linesize=64 pagesize=60;

data size;
  input Distance ObjectSize @@;
  Precision=1/distance;
  datalines;
1.5 30 1.5 20 1.5 30 1.5 25
3   43 3   33 3   25 3   30
4.5 25 4.5 36 4.5 48 4.5 33
6   43 6   36 6   23 6   48
7.5 30 7.5 25 7.5 50 7.5 38
;

proc means data=size maxdec=3 n mean var stddev;
  var objectsize;
  title1 'Unweighted Analysis of the SIZE Data Set';
run;
```

Output

Unweighted Analysis of the SIZE Data Set

The MEANS Procedure

Analysis Variable : ObjectSize			
N	Mean	Variance	Std Dev
20	33.550	80.892	8.994

Use the WEIGHT Statement with Precision in PROC MEANS

The next two PROC MEANS steps use the precision measure (Precision) in the WEIGHT statement and show the effect of using different values of the VARDEF= option. The first PROC step creates an output data set that contains the variance and standard deviation. If you reduce the weighting of the estimates that are made at greater distances, the weighted average estimate of the object size is closer to the actual size. The variance of the i th observation is assumed to be $\text{var}(x_i) = \sigma^2/w_i$ and w_i is the weight for the i th observation. In the first PROC MEANS step, the

computed variance is an estimate of σ^2 . In the second PROC MEANS step, the computed variance is an estimate of $(n - 1/n)\sigma^2/\bar{w}$, where $\bar{w}$ is the average weight. For large n , this value is an approximate estimate of the variance of an observation with average weight.

```
proc means data=size maxdec=3 n mean var stddev;
  weight precision;
  var objectsize;
  output out=wtstats var=Est_SigmaSq std=Est_Sigma;
  title1 'Weighted Analysis Using Default VARDEF=DF';
run;

proc means data=size maxdec=3 n mean var std
           vardef=weight;
  weight precision;
  var objectsize;
  title1 'Weighted Analysis Using VARDEF=WEIGHT';
run;
```

Output

Weighted Analysis Using Default VARDEF=DF

The MEANS Procedure

Analysis Variable : ObjectSize			
N	Mean	Variance	Std Dev
20	31.088	20.678	4.547

Weighted Analysis Using VARDEF=WEIGHT

The MEANS Procedure

Analysis Variable : ObjectSize			
N	Mean	Variance	Std Dev
20	31.088	64.525	8.033

Create a Data Set with the Weighted Variance and Weighted Standard Deviation of Each Observation

In the following statements, DATA step combines the output data set that contains the variance and the standard deviation from the weighted analysis with the original data set. The variance of each observation is computed by dividing

Est_SigmaSq (the estimate of σ^2 from the weighted analysis when VARDEF=DF) by each observation's weight (Precision). The standard deviation of each observation is computed by dividing Est_Sigma (the estimate of σ from the weighted analysis when VARDEF=DF) by the square root of each observation's weight (Precision).

```
data wtsize(drop=_freq_ _type_);
    set size;
    if _n_=1 then set wtstats;
    Est_VarObs=est_sigmasq/precision;
    Est_StdObs=est_sigma/sqrt(precision);

proc print data=wtsize noobs;
    title 'Weighted Statistics';
    by distance;
    format est_varobs est_stdobs
           est_sigmasq est_sigma precision 6.3;
run;
```

Output

Weighted Statistics

Distance=1.5

ObjectSize	Precision	Est_SigmaSq	Est_Sigma	Est_VarObs	Est_StdObs
30	0.667	20.678	4.547	31.017	5.569
20	0.667	20.678	4.547	31.017	5.569
30	0.667	20.678	4.547	31.017	5.569
25	0.667	20.678	4.547	31.017	5.569

Distance=3

ObjectSize	Precision	Est_SigmaSq	Est_Sigma	Est_VarObs	Est_StdObs
43	0.333	20.678	4.547	62.035	7.876
33	0.333	20.678	4.547	62.035	7.876
25	0.333	20.678	4.547	62.035	7.876
30	0.333	20.678	4.547	62.035	7.876

Distance=4.5

ObjectSize	Precision	Est_SigmaSq	Est_Sigma	Est_VarObs	Est_StdObs
25	0.222	20.678	4.547	93.052	9.646
36	0.222	20.678	4.547	93.052	9.646
48	0.222	20.678	4.547	93.052	9.646
33	0.222	20.678	4.547	93.052	9.646

Distance=6					
ObjectSize	Precision	Est_SigmaSq	Est_Sigma	Est_VarObs	Est_StdObs
43	0.167	20.678	4.547	124.07	11.139
36	0.167	20.678	4.547	124.07	11.139
23	0.167	20.678	4.547	124.07	11.139
48	0.167	20.678	4.547	124.07	11.139

Distance=7.5					
ObjectSize	Precision	Est_SigmaSq	Est_Sigma	Est_VarObs	Est_StdObs
30	0.133	20.678	4.547	155.09	12.453
25	0.133	20.678	4.547	155.09	12.453
50	0.133	20.678	4.547	155.09	12.453
38	0.133	20.678	4.547	155.09	12.453

WHERE Statement

Subsets the input data set by specifying certain conditions that each observation must meet before it is available for processing.

Syntax

WHERE *where-expression-1*
 < *logical-operator where-expression-n*>;

Arguments

where-expression

is an arithmetic or logical expression that generally consists of a sequence of operands and operators.

Tips The operands and operators described in the next several sections are also valid for the WHERE= data set option.

You can specify multiple where-expressions.

logical-operator

can be AND, AND NOT, OR, or OR NOT.

Example

Procedures That Support the WHERE Statement

You can use the WHERE statement with any of the following Base SAS procedures that read a SAS data set:

COMPARE	REPORT
CORR	SORT
DATASETS (APPEND statement)	SQL
FREQ	STANDARD
MEANS or SUMMARY	TABULATE
PRINT	TRANSPOSE
RANK	UNIVARIATE

Details

- The COMPARE procedure and the APPEND statement in PROC DATASETS accept more than one input data set. For more information, see the documentation for the specific procedure.
- To subset the output data set, use the WHERE= data set option:

```
proc report data=debate nowd  
            out=onlyfr (where= (year='1')) ;  
run;
```

For more information about WHERE=, see [SAS DATA Step Statements: Reference](#).

Temporarily Dissociating a Format from a Variable

In this example, PROC PRINT prints only those observations that meet the condition of the WHERE expression.

```
options nodate pageno=1 linesize=64
      pagesize=40;

proc print data=debate noobs;
  where gpa>3.5;
  title 'Team Members with a GPA';
  title2 'Greater than 3.5';
run;
```

Output

**Team Members with a GPA
Greater than 3.5**

Name	Gender	Year	GPA
Capiccio	m	Freshman	3.598
Tucker	m	Freshman	3.901
Bagwell	f	Sophomore	3.722
Gold	f	Junior	3.609
Syme	f	Junior	3.883
Baglione	f	Senior	4.000
Carr	m	Senior	3.750
Hall	m	Senior	3.574

TABULATE Procedure

Overview: TABULATE Procedure	2207
What Does the TABULATE Procedure Do?	2208
Simple Tables	2209
Complex Tables	2209
PROC TABULATE and the Output Delivery System	2210
Concepts: TABULATE Procedure	2211
Terminology: TABULATE Procedure	2211
Threaded Processing of Input Data Sets	2214
Syntax: TABULATE Procedure	2214
PROC TABULATE Statement	2216
ATTRIB Statement	2229
BY Statement	2230
CLASS Statement	2234
CLASSLEV Statement	2241
FORMAT Statement	2244
FREQ Statement	2244
KEYLABEL Statement	2245
KEYWORD Statement	2246
LABEL Statement	2248
TABLE Statement	2250
VAR Statement	2264
WEIGHT Statement	2267
WHERE Statement	2269
Usage: TABULATE Procedure	2271
Statistics That Are Available in PROC TABULATE	2271
Formatting Class Variables	2273
Formatting Values in Tables	2274
Calculating Percentages	2275
Using ODS Styles with PROC TABULATE	2278
SAS Cloud Analytic Services Processing for PROC TABULATE	2292
In-Database Processing for PROC TABULATE	2293
Substituting BY Line Values in a Text String	2294
Results: TABULATE Procedure	2295
Missing Values	2295
Understanding the Order of Headings with ORDER=DATA	2304
Examples: TABULATE Procedure	2306

Example 1: Creating a Basic Two-Dimensional Table	2306
Example 2: Specifying Class Variable Combinations to Appear in a Table	2308
Example 3: Using Preloaded Formats with Class Variables	2311
Example 4: Using Multilabel Formats	2315
Example 5: Customizing Row and Column Headings	2318
Example 6: Summarizing Information with the Universal Class Variable ALL ...	2320
Example 7: Eliminating Row Headings	2323
Example 8: Creating Multipage Tables	2325
Example 9: Reporting on Multiple-Response Survey Data	2328
Example 10: Reporting on Multiple-Choice Survey Data	2332
Example 11: Calculating Various Percentage Statistics	2339
Example 12: Using Denominator Definitions to Display Basic Frequency Counts and Percentages	2343
Example 13: Specifying Style Overrides for ODS Output	2357
Example 14: Style Precedence	2362
Example 15: Using the NOCELLMERGE Option	2366
References	2369

Overview: TABULATE Procedure

What Does the TABULATE Procedure Do?

The TABULATE procedure displays descriptive statistics in tabular format, using some or all of the variables in a data set. You can create a variety of tables ranging from simple to highly customized.

PROC TABULATE computes many of the same statistics that are computed by other descriptive statistical procedures such as MEANS, FREQ, and REPORT. PROC TABULATE provides the following features:

- simple but powerful methods to create tabular reports
- flexibility in classifying the values of variables and establishing hierarchical relationships between the variables
- mechanisms for labeling and formatting variables and procedure-generated statistics
- the ability to create accessible output tables when used with the ACCESSIBLETABLE system option. PROC TABULATE also provides the ability to add captions to tables.

For more information about creating accessible tables with PROC TABULATE, see [“Creating Accessible Tables” in *Creating Accessible SAS Viya Platform Output Using ODS and ODS Graphics*](#).

Simple Tables

The following output shows a simple table that was produced by PROC TABULATE. The data set “ENERGY” on page 2512 contains data on expenditures of energy by two types of customers, residential and business, in individual states in the Northeast (1) and West (4) regions of the United States. The table sums expenditures for states within a geographic division. The RTS option provides enough space to display the column headings without hyphenating them.

```
proc tabulate data=energy;
  class region division type;
  var expenditures;
  table region*division, type*expenditures /
    rts=20;
run;
```

Output 61.1 Simple Table Produced by PROC TABULATE

		Type	
		1	2
		Expenditures	Expenditures
		Sum	Sum
Region	Division		
1	1	7477.00	5129.00
	2	19379.00	15078.00
4	3	5476.00	4729.00
	4	13959.00	12619.00

Complex Tables

The following output is a more complicated table using the same data set that was used to create [Output 61.301 on page 2209](#). The statements that create this report do the following:

- customize column and row headings
- apply a format to all table cells
- sum expenditures for residential and business customers
- compute subtotals for each division
- compute totals for all regions

For an explanation of the program that produces this report, see “[Example 6: Summarizing Information with the Universal Class Variable ALL](#)” on page 2320.

Output 61.2 Complex Table Produced by PROC TABULATE

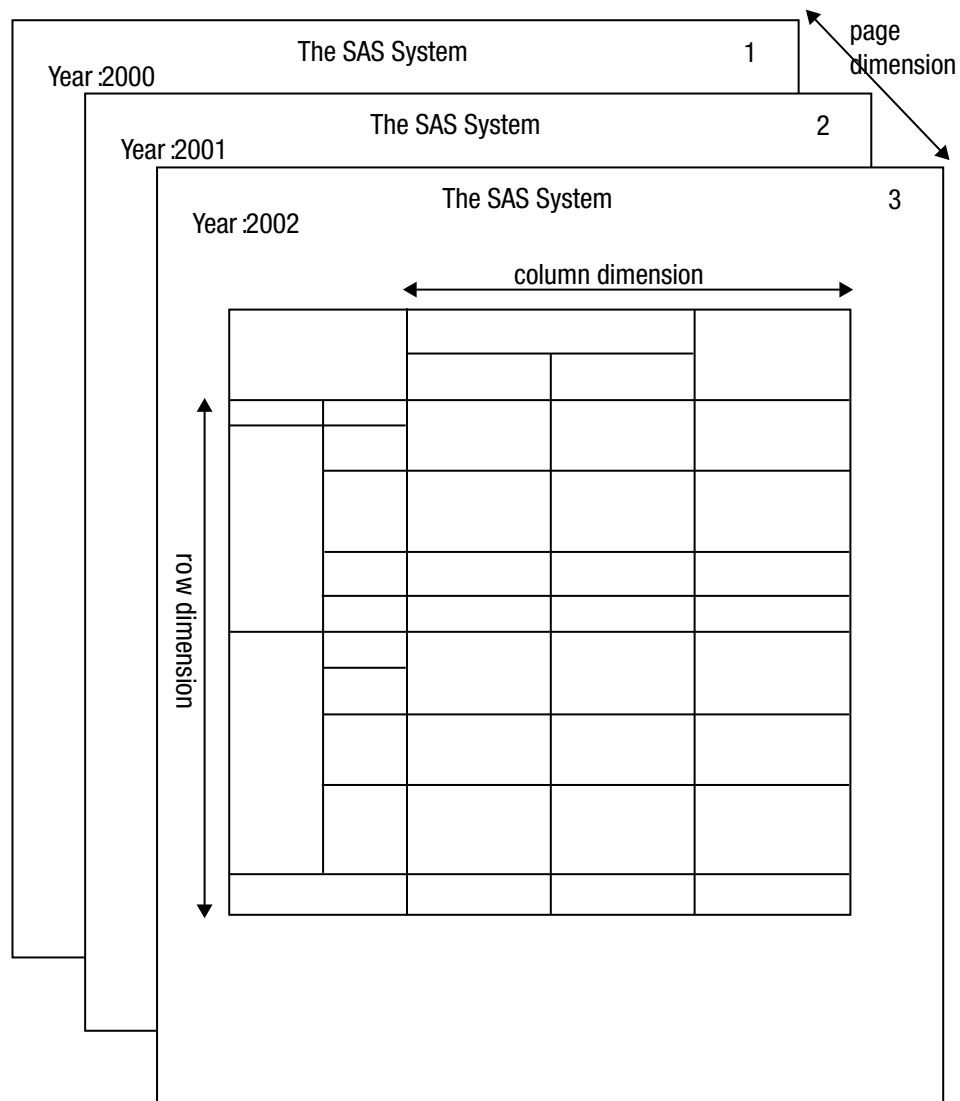
Energy Expenditures for Each Region (millions of dollars)				
		Customer Base		All Customers
		Residential Customers	Business Customers	
Region	Division			
Northeast	New England	7,477	5,129	12,606
	Middle Atlantic	19,379	15,078	34,457
	Subtotal	26,856	20,207	47,063
West	Division			
	Mountain	5,476	4,729	10,205
	Pacific	13,959	12,619	26,578
	Subtotal	19,435	17,348	36,783
Total for All Regions		\$46,291	\$37,555	\$83,846

PROC TABULATE and the Output Delivery System

The following output shows a table that is created in HTML5. You can use the Output Delivery System with PROC TABULATE to create customized output in HTML5, Rich Text Format (RTF), Word, Portable Document Format (PDF), and other output formats. For an explanation of the program that produces this table, see [“Example 13: Specifying Style Overrides for ODS Output”](#) on page 2357.

Output 61.3 HTML Table Produced by PROC TABULATE

Energy Expenditures (millions of dollars)				
Region=Northeast				
Region by Division by Type		Type		Total
		Residential Customers	Business Customers	
		Expenditures	Expenditures	Expenditures
		Sum	Sum	Sum
Region	Division			
Northeast	New England	\$7,477	\$5,129	\$12,606
	Middle Atlantic	\$19,379	\$15,078	\$34,457
	Total	\$26,856	\$20,207	\$47,063
Total	Division			
	New England	\$7,477	\$5,129	\$12,606
	Middle Atlantic	\$19,379	\$15,078	\$34,457
	Total	\$26,856	\$20,207	\$47,063

Figure 61.2 PROC TABULATE Table Dimensions

In addition, the following terms frequently appear in discussions of PROC TABULATE:

category

the combination of unique values of class variables. The TABULATE procedure creates a separate category for each unique combination of values that exists in the observations of the data set. Each category that is created by PROC TABULATE is represented by one or more cells in the table where the pages, rows, and columns that describe the category intersect.

The table in [Figure 61.59 on page 2211](#) contains three class variables: Region, Division, and Type. These class variables form the eight categories listed in the following table. (For convenience, the categories are described in terms of their formatted values.)

Table 61.1 Categories Created from Three Class Variables

Region	Division	Type
Northeast	New England	Residential Customers
Northeast	New England	Business Customers
Northeast	Middle Atlantic	Residential Customers
Northeast	Middle Atlantic	Business Customers
West	Mountain	Residential Customers
West	Mountain	Business Customers
West	Pacific	Residential Customers
West	Pacific	Business Customers

continuation message

the text that appears below the table if it spans multiple physical pages.

nested variable

a variable whose values appear in the table with each value of another variable.

In [Figure 61.59 on page 2211](#), Division is nested under Region.

page dimension text

the text that appears above the table if the table has a page dimension. However, if you specify `BOX=_PAGE_` in the TABLE statement, then the text that would appear above the table appears in the box. In [Figure 61.60 on page 2212](#), the word **Year:**, followed by the value, is the page dimension text.

Page dimension text has a style. The default style is *Beforecaption*. For more information about using styles, see [“Using ODS Styles with PROC TABULATE” on page 2278](#).

subtable

the group of cells that is produced by crossing a single element from each dimension of the TABLE statement when one or more dimensions contain concatenated elements.

[Figure 61.59 on page 2211](#) contains no subtables. For an illustration of a table that consists of multiple subtables, see [“Example 12: Using Denominator Definitions to Display Basic Frequency Counts and Percentages” on page 2343](#).

Threaded Processing of Input Data Sets

The THREADED option enables or disables parallel processing of the input data set. Threaded processing achieves a degree of parallelism in the processing operations. This parallelism is intended to reduce the real time to completion for a given operation and therefore limit the cost of additional CPU resources. For more information, see [Support for Parallel Processing](#).

The value of the SAS system option CPUCOUNT= affects the performance of the threaded sort. CPUCOUNT= suggests how many system CPUs are available for use by the threaded procedures.

For more information, see the “[THREADS System Option](#)” in *SAS System Options: Reference* and the “[CPUCOUNT= System Option](#)” in *SAS System Options: Reference*.

Calculated statistics can vary slightly, depending on the order in which observations are processed. Such variations are due to numerical errors that are introduced by floating-point arithmetic, the results of which should be considered approximate and not exact. The order of observation processing can be affected by nondeterministic effects of multithreaded or parallel processing. The order of processing can also be affected by inconsistent or nondeterministic ordering of observations that are produced by a data source, such as a DBMS that delivers query results through an ACCESS engine. For more information, see “[Threading in Base SAS](#)” in *SAS Programmer's Guide: Essentials*.

Syntax: TABULATE Procedure

- | | |
|---------------------|--|
| Requirements: | At least one TABLE statement is required in a PROC TABULATE procedure step. Depending on the variables that appear in the TABLE statement, a CLASS statement, a VAR statement, or both are required. |
| Supports: | Threaded processing |
| Accessibility note: | Use the ACCESSIBLECHECK and ACCESSIBLETABLE system options to check for and create accessible tables. For information about creating accessible PROC TABULATE tables, see “ Creating Accessible Tables ” in <i>Creating Accessible SAS Viya Platform Output Using ODS and ODS Graphics</i> . |
| Tips: | <p>You can use the ATTRIB, FORMAT, LABEL, and WHERE statements with the PROC TABULATE procedure.</p> <p>For in-database processing to occur, your data must reside within a supported version of a DBMS that has been properly configured for SAS in-database processing. For more information, see “In-Database Processing for PROC TABULATE” on page 2293.</p> |

PROC TABULATE <options> <STYLE=style-override(s)>;

ATTRIB *variable-list(s) attribute-list(s);*
BY <DESCENDING> *variable-1*
 <<DESCENDING> *variable-n ...*> <NOTSORTED>;
CLASS *variable(s)* </ options> <STYLE=style-override(s)>;
CLASSLEV *variable(s)* </ STYLE=style-override(s)>;
FORMAT *variable-1* <...*variable-n*> <format> <DEFAULT=default-format>;
FREQ *variable*;
KEYLABEL *keyword-1*=*'description-1'* <*keyword-2*=*'description-2'* ...>;
KEYWORD *keyword(s)* </ STYLE=style-override(s)>;
LABEL *variable-1*=*label-1*<...*variable-n*=*label-n*>;
TABLE <<*page-expression*,> *row-expression*,>
 column-expression </ table-options>;
VAR *analysis-variable(s)* </ options> <STYLE=style-override(s)>;
WEIGHT *variable*;
WHERE *where-expression-1*
 < *logical-operator where-expression-n*>;

Statement	Task	Example
PROC TABULATE	Display descriptive statistics in tabular format	Ex. 1, Ex. 2, Ex. 3, Ex. 4, Ex. 6, Ex. 11, Ex. 13, Ex. 14
BY	Create a separate table for each BY group	
CLASS	Identify variables in the input data set as class variables	Ex. 3, Ex. 4
CLASSLEV	Specify a style for class variable level value headings	Ex. 13, Ex. 14
FREQ	Identify a variable in the input data set whose values represent the frequency of each observation	
KEYLABEL	Specify a label for a keyword	
KEYWORD	Specify a style for keyword headings	Ex. 13
TABLE	Describe the table to create	Ex. 1, Ex. 3, Ex. 4, Ex. 5, Ex. 6, Ex. 7, Ex. 8, Ex. 9, Ex. 10, Ex. 11, Ex. 12, Ex. 13, Ex. 14
VAR	Identify variables in the input data set as analysis variables	Ex. 13

Statement	Task	Example
<code>WEIGHT</code>	Identify a variable in the input data set whose values weight each observation in the statistical calculations	

PROC TABULATE Statement

Displays descriptive statistics in tabular format.

Syntax

PROC TABULATE *<options>*;

Summary of Optional Arguments

CONTENTS=*link-text* *<#BYLINE>* *<#BYVAL>* *<#BYVAR>*
specifies the text for the entries in the table of contents.

DATA=*SAS-data-set*
specifies the input data set.

NOTHREADS
disables parallel processing of the input data set. This option overrides the SAS system option `THREADS` | `NOTHREADS` unless the system option is restricted.

OUT=*SAS-data-set*
specifies the output data set.

THREADS
enables parallel processing of the input data set. This option overrides the SAS system option `THREADS` | `NOTHREADS` unless the system option is restricted.

TRAP
enables floating point exception recovery during data processing beyond the recovery that is provided by normal SAS FPE handling.

Control the statistical analysis

ALPHA=*value*
specifies the confidence level for the confidence limits.

EXCLNPWGT
excludes observations with nonpositive weights.

QMARKERS=*number*
specifies the sample size to use for the P^2 quantile estimation method.

QMETHOD=`OS` | `P2`

specifies the quantile estimation method.

QNTLDEF=1 | 2 | 3 | 4 | 5

specifies the mathematical definition to calculate quantiles.

VARDEF=DF | N | WDF | WEIGHT

specifies the variance divisor to use in the calculation of the variance and standard deviation.

Customize the appearance of the table

FORMAT=*format-name*

specifies a default format for each cell in the table.

FORMCHAR= <(position(s))>=*'formatting-character(s)'*

defines the characters to use to construct the table outlines and dividers.

NOSEPS

eliminates horizontal separator lines from the row titles and the body of the table.

ORDER=DATA | FORMATTED | FREQ | UNFORMATTED

orders the values of a class variable according to the specified order.

STYLE=*style-override(s)*

specifies one or more style overrides to use for specific areas of the table.

Identify categories of data that are of interest

CLASSDATA=*SAS-data-set*

specifies a secondary data set that contains the combinations of values of class variables to include in tables and output data sets.

EXCLUSIVE

excludes from tables and output data sets all combinations of class variable values that are not in the CLASSDATA= data set.

MISSING

considers missing values as valid values for class variables.

Optional Arguments

ALPHA=*value*

specifies the confidence level to compute the confidence limits for the mean.

The percentage for the confidence limits is $(1 - \text{value}) \times 100$. For example,

ALPHA= .05 results in a 95% confidence limit.

Default .05

Range 0-1

Interaction To compute confidence limits specify the *statistic-keyword* LCLM or UCLM.

CLASSDATA=*SAS-data-set*

specifies a data set that contains the combinations of values of the class variables that must be present in the output. Any combination of values of the class variables appear in each table or output data set and have a frequency of zero if they meet the following criteria:

- 1 occur in the CLASSDATA= data set
- 2 but not in the input data set

Restriction The CLASSDATA= data set must contain all class variables. Their data type and format must match the corresponding class variables in the input data set.

Interaction If you use the EXCLUSIVE option, then PROC TABULATE excludes any observations in the input data set whose combinations of values of class variables are not in the CLASSDATA= data set.

Tip Use the CLASSDATA= data set to filter or supplement the input data set.

Example [“Example 2: Specifying Class Variable Combinations to Appear in a Table” on page 2308](#)

CONTENTS=[link-text](#) **<#BYLINE> <#BYVAL> <#BYVAR>**

specifies the text for the entries in the table of contents created by default or by options settings in ODS destinations that support the STYLE= option.

These options are available if a BY statement is in effect:

#BYLINE

substitutes the entire BY line without leading or trailing blanks for #BYLINE in the text string. The BY line uses the format *variable-name=value*.

#BYVAL_{*n*}

#BYVAL(*BY-variable-name*)

substitutes the current value of the specified BY variable for #BYVAL in the text string.

Specify the variable with one of the following:

n

specifies a variable by its position in the BY statement. For example, #BYVAL2 specifies the second variable in the BY statement.

BY-variable-name

specifies a variable from the BY statement by its name. For example, #BYVAL(YEAR) specifies the BY variable, YEAR. *Variable-name* is not case sensitive.

#BYVAR_{*n*}

#BYVAR(*BY-variable-name*)

substitutes the name of the BY-variable or the label associated with the variable (whatever the BY line would normally display) for #BYVAR in the text string.

Specify the variable with one of the following:

n

specifies a variable by its position in the BY statement. For example, #BYVAR2 specifies the second variable in the BY statement.

BY-variable-name

specifies a variable from the BY statement by its name. For example, #BYVAR(SITES) specifies the BY variable, SITES. *Variable-name* is not case sensitive.

See [“Substituting BY Line Values in a Text String” on page 2294](#)

DATA=SAS-data-set

specifies the input data set.

See [“Input Data Sets” on page 24](#)

EXCLNPWGT

excludes observations with nonpositive weight values (zero or negative) from the analysis. By default, PROC TABULATE treats observations with negative weights like observations with zero weights and counts them in the total number of observations.

Alias EXCLNPWGTS

See [“WEIGHT=weight-variable” on page 2267](#) and [“WEIGHT Statement” on page 2267](#)

EXCLUSIVE

excludes from the tables and the output data sets all combinations of the class variable that are not found in the CLASSDATA= data set.

Requirement If a CLASSDATA= data set is not specified, then the EXCLUSIVE option is ignored.

Example [“Example 2: Specifying Class Variable Combinations to Appear in a Table” on page 2308](#)

FORMAT=format-name

specifies a default format for the value in each table cell. You can use any SAS or user-defined format.

Alias F=

Default If you omit FORMAT=, then PROC TABULATE uses BEST12.2 as the default format.

Interaction Formats that are specified in a TABLE statement override the format that is specified with FORMAT=.

Tip The FORMAT= option is especially useful for controlling the number of print positions that are used to print a table.

Examples [“Example 1: Creating a Basic Two-Dimensional Table” on page 2306](#)
[“Example 6: Summarizing Information with the Universal Class Variable ALL” on page 2320](#)

FORMCHAR= <(position(s))>='formatting-character(s)'

defines the characters to use for constructing the table outlines and dividers.

position(s)

identifies the position of one or more characters in the SAS formatting-character string. A space or a comma separates the positions.

Default Omitting *position(s)* is the same as specifying all 20 possible SAS formatting characters, in order.

formatting-character(s)

lists the characters to use for the specified positions.

PROC TABULATE assigns characters in *formatting-character(s)* to *position(s)*, in the order in which they are listed. For example, the following option assigns the asterisk (*) to the third formatting character, the number sign (#) to the seventh character, and does not alter the remaining characters:

```
formchar(3,7)='*#'
```

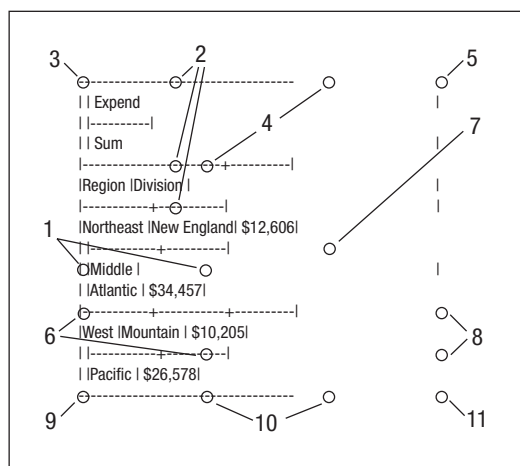
PROC TABULATE uses 11 of the 20 formatting characters that SAS provides. [Table 61.114 on page 2220](#) shows the formatting characters that PROC TABULATE uses. [Figure 61.61 on page 2221](#) illustrates the use of each formatting character in the output from PROC TABULATE.

Table 61.2 *Formatting Characters Used by PROC TABULATE*

Position	Default	Used to Draw
1		Right and left borders and the vertical separators between columns
2	-	Top and bottom borders and the horizontal separators between rows
3	-	Top character in the left border
4	-	Top character in a line of characters that separate columns
5	-	Top character in the right border
6		Leftmost character in a row of horizontal separators
7	+	Intersection of a column of vertical characters and a row of horizontal characters
8		Rightmost character in a row of horizontal separators
9	-	Bottom character in the left border
10	-	Bottom character in a line of characters that separate columns

Position	Default	Used to Draw
11	-	Bottom character in the right border

Figure 61.3 Formatting Characters in PROC TABULATE Output



- Restriction** The FORMCHAR= option affects only the traditional SAS monospace output destination.
- Interaction** The SAS system option FORMCHAR= specifies the default formatting characters. The system option defines the entire string of formatting characters. The FORMCHAR= option in a procedure can redefine selected characters.
- Tips** You can use any character in *formatting-characters*, including hexadecimal characters. If you use hexadecimal characters, then you must put x after the closing quotation mark. For example, the following option assigns the hexadecimal character 2-D to the third formatting character, assigns the hexadecimal character 7C to the seventh character, and does not alter the remaining characters:
- ```
formchar (3, 7) = '2D7C'x
```
- Specifying all blanks for *formatting-character(s)* produces tables with no outlines or dividers.
- ```
formchar (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11) = '          '
(11 blanks)
```
- See** For more examples using formatting output, see *PROC TABULATE by Example, Second Edition*.
- For information about which hexadecimal codes to use for which characters, consult the documentation for your hardware.

MISSING

considers missing values as valid values to create the combinations of class variables. Special missing values that are used to represent numeric values (the

letters A through Z and the underscore (_) character) are each considered as a separate value. A heading for each missing value appears in the table.

Default If you omit MISSING, then PROC TABULATE does not include observations with a missing value for any class variable in the report.

See [“Including Observations with Missing Class Variables” on page 2299](#)
[Creating Special Missing Values](#) for a discussion of missing values that have special meaning.

NOSEPS

eliminates horizontal separator lines from the row titles and the body of the table. Horizontal separator lines remain between nested column headings.

Restriction The NOSEPS option affects only the traditional SAS monospace output destination.

Tip If you want to replace the separator lines with blanks rather than remove them, then use option **“FORMCHAR=**
<(position(s))>=’formatting-character(s)’ ” on page 2219.

NOTHEADS

disables parallel processing of the input data set. This option overrides the SAS system option THREADS | NOTHEADS unless the system option is restricted.

The default is the value of SAS system option THREADS | NOTHEADS.

Restriction Your site administrator can create a restricted options table. A restricted options table specifies SAS system option values that are established at start-up and cannot be overridden. If the THREADS | NOTHEADS system option is listed in the restricted options table, any attempt to set these system options is ignored and a warning message is written to the SAS log.

Interaction PROC TABULATE uses the value of the SAS system option THREADS except when a BY statement is specified or the value of the SAS system option CPUCOUNT is less than 2. In those cases, you can specify the THREADS option in the PROC TABULATE statement to force PROC TABULATE to use parallel processing. When multi-threaded processing, also known as parallel processing, is in effect, observations might be returned in an unpredictable order. However, the observations are sorted correctly if a BY statement is specified.

See To enable parallel processing of the input data set, see **“THREADS ”** on page 2227.

For more information, see [Support for Parallel Processing](#).

ORDER=DATA | FORMATTED | FREQ | UNFORMATTED

specifies the sort order to create the unique combinations of the values of the class variables, which form the headings of the table, according to the specified order.

DATA

orders values according to their order in the input data set.

Interaction If you use PRELOADFMT in the CLASS statement, then the order for the values of each class variable matches the order that PROC FORMAT uses to store the values of the associated user-defined format. If you use the CLASSDATA= option, then PROC TABULATE uses the order of the unique values of each class variable in the CLASSDATA= data set to order the output levels. If you use both options, then PROC TABULATE first uses the user-defined formats to order the output. If you omit EXCLUSIVE, then PROC TABULATE appends after the user-defined format and the CLASSDATA= values the unique values of the class variables in the input data set in the same order in which they are encountered.

Tips By default, PROC FORMAT stores a format definition in sorted order. Use the NOTSORTED option to store the values or ranges of a user-defined format in the order in which you define them.

When you are using in-database procedures with the ORDER=DATA option, the results can vary. DBMS tables have no inherent order for the rows. The order of rows written to a database table from a SAS procedure is not likely to be preserved.

FORMATTED

orders values by their ascending formatted values. If no format has been assigned to a numeric class variable, then the default format, BEST12., is used. This order depends on your operating environment.

Aliases FMT

EXTERNAL

FREQ

orders values by descending frequency count.

Interaction Use the ASCENDING option in the CLASS statement to order values by ascending frequency count.

UNFORMATTED

orders values by their unformatted values, which yields the same order as PROC SORT. This order depends on your operating environment. This sort sequence is particularly useful for displaying dates chronologically.

Aliases UNFMT

INTERNAL

Default UNFORMATTED

Interaction If you use the PRELOADFMT option in the CLASS statement, then PROC TABULATE orders the levels by the order of the values in the user-defined format.

Example [“Understanding the Order of Headings with ORDER=DATA” on page 2304](#)

OUT=SAS-data-set

names the output data set. If *SAS-data-set* does not exist, then PROC TABULATE creates it.

The number of observations in the output data set depends on the number of categories of data that are used in the tables and the number of subtables that are generated. The output data set contains these variables (in the following order):

by variables	variables that are listed in the BY statement.
class variables	variables that are listed in the CLASS statement.
TYPE	a character variable that shows which combination of class variables produced the summary statistics in that observation. Each position in _TYPE_ represents one variable in the CLASS statement. If that variable is in the category that produced the statistic, then the position contains a 1. Otherwise, the position contains a 0. In simple PROC TABULATE steps that do not use the universal class variable ALL, all values of _TYPE_ contain only 1s because the only categories that are being considered involve all class variables. If you use the variable ALL, then your tables will contain data for categories that do not include all the class variables, and positions of _TYPE_ will, therefore, include both 1s and 0s.
PAGE	The logical page that contains the observation.
TABLE	The number of the table that contains the observation.
statistics	statistics that are calculated for each observation in the data set.

Example [“Example 3: Using Preloaded Formats with Class Variables” on page 2311](#)

QMARKERS=number

specifies the default number of markers to use for the P^2 quantile estimation method. The number of markers controls the size of fixed memory space.

Default The default value depends on which quantiles you request. For the median (P50), *number* is 7. For the quartiles (P25 and P75), *number* is 25. For the quantiles P1, P5, P10, P90, P95, or P99, *number* is 105. If you request several quantiles, then PROC TABULATE uses the largest default value of *number*.

Range any odd integer greater than 3

Tip Increase the number of markers above the default settings to improve the accuracy of the estimates; reduce the number of markers to conserve memory and computing time.

See [“Quantiles” on page 1295](#)

QMETHOD=OS | P2

specifies the method PROC TABULATE uses to process the input data when it computes quantiles. If the number of observations is less than or equal to the QMARKERS= value and QNTLDEF=5, then both methods produce the same results.

OS

uses order statistics. PROC UNIVARIATE uses this technique.

Note: This technique can be very memory-intensive.

P2

uses the P^2 method to approximate the quantile.

Alias HIST

Default OS

Restriction When QMETHOD=P2, PROC TABULATE will not compute MODE or weighted quantiles.

Tip When QMETHOD=P2, reliable estimates of some quantiles (P1, P5, P95, P99) might not be possible for some types of data.

See [“Quantiles” on page 1295](#)

QNTLDEF=1 | 2 | 3 | 4 | 5

specifies the mathematical definition that the procedure uses to calculate quantiles when QMETHOD=OS is specified. When QMETHOD=P2, you must use QNTLDEF=5.

Alias PCTLDEF=

Default 5

See [“Quantile and Related Statistics” on page 2432](#)

STYLE=style-override(s)

specifies one or more style overrides to use for the data cells of a table. For example, the following statement specifies that the background color for data cells is red:

```
proc tabulate data=one style=[backgroundcolor=red];
```

style-override

specifies one or more style attributes or style elements to override the default style element and attributes in a specific area of a report.

You can specify a style override in three ways:

- Specify a style element. A style element is a collection of style attributes that apply to a particular part of the output for a SAS program.
- Specify a style attribute. A style attribute is a name-value pair that describes a single behavioral or visual aspect of a piece of output. This is the most specific method of changing the appearance of your output.
- Specify the PARENT value. The PARENT value specifies that the data cell use the style element of its parent heading.

style-override has the following form:

```
PARENT | style-element-name | [style-attribute-name-1=style-attribute-value-1
<style-attribute-name-2=style-attribute-value-2 ...>]
```

<PARENT>

specifies that the data cell use the style element of its parent heading.

The parent style element of a data cell is one of the following:

- the style element of the leaf heading above the column that contains the data cell, if the table specifies no row dimension, or if the table specifies the style element in the column dimension expression
- the style element of the leaf heading above the row that contains the cell, if the table specifies the style element in the row dimension expression
- the Beforecaption style element, if the table specifies the style element in the page dimension expression
- undefined, otherwise

Note: In this usage, the angle brackets around the word PARENT are required. Braces or square brackets cannot be substituted in the syntax.

Note: The parent of a heading (not applicable to STYLE= in the PROC TABULATE statement) is the heading under which the current heading is nested.

style-attribute-name

specifies the attribute to change.

See [For information about using styles with PROC TABULATE, see “Using ODS Styles with PROC TABULATE” on page 2278.](#)

For a table of default style attributes and style elements for each ODS destination, see [“Style Elements and Style Attributes for Table Regions” on page 2286.](#)

style-attribute-value

specifies a value for the attribute. Each attribute has a different set of valid values. A SAS format can also be used as an attribute value for conditional formatting.

See For information about using styles with PROC TABULATE, see [“Using ODS Styles with PROC TABULATE” on page 2278](#).

For information about using SAS formats as style attribute values, see [“Using a Format to Assign a Style Attribute” on page 2290](#).

For a table of default style attributes and style elements for each ODS destination, see [“Style Elements and Style Attributes for Table Regions” on page 2286](#).

style-element-name

is the name of a style element that is part of an ODS style template.

See For information about using styles with PROC TABULATE, see [“Using ODS Styles with PROC TABULATE” on page 2278](#).

For a table of default style attributes and style elements for each ODS destination, see [“Style Elements and Style Attributes for Table Regions” on page 2286](#).

Alias S=

Tips The STYLE= option can be used in other statements, or in dimension expressions, to specify style elements for other parts of a table.

To specify a style element for data cells with missing values, use STYLE= in the TABLE statement MISSTEXT= option.

You can use braces ({ and }) instead of square brackets ([and]).

See For information about using styles with PROC TABULATE, see [“Using ODS Styles with PROC TABULATE” on page 2278](#).

Example [“Example 13: Specifying Style Overrides for ODS Output” on page 2357](#)

THREADS

enables parallel processing of the input data set. This option overrides the SAS system option THREADS | NOTHEADS unless the system option is restricted.

The default is the value of SAS system option THREADS | NOTHEADS.

Restriction Your site administrator can create a restricted options table. A restricted options table specifies SAS system option values that are established at start-up and cannot be overridden. If the THREADS | NOTHEADS system option is listed in the restricted options table, any attempt to set these system options is ignored and a warning message is written to the SAS log.

Interaction PROC TABULATE uses the value of the SAS system option THREADS except when a BY statement is specified or the value of the SAS system option CPUCOUNT is less than 2. In those cases, you can specify the THREADS option in the PROC TABULATE statement to force PROC TABULATE to use parallel processing. When multi-threaded processing, also known as parallel processing, is in effect, observations might be returned in an unpredictable order. However, the observations are sorted correctly if a BY statement is specified.

See To disable parallel processing of the input data set, see [“NOTHEADS” on page 2222](#).

For more information, see [Support for Parallel Processing](#).

TRAP

enables floating point exception (FPE) recovery during data processing beyond the recovery that is provided by normal SAS FPE handling. Note that without the TRAP option, normal SAS FPE handling is still in effect so that PROC TABULATE terminates in the case of math exceptions.

VARDEF=DF | N | WDF | WEIGHT

specifies the divisor to use in the calculation of the variance and standard deviation. The following table shows the possible values for *divisor* and the associated divisors.

DF

Degrees of freedom

The following is the formula for the divisor:

$$n - 1$$

N

Number of observations

The following is the formula for the divisor:

$$n$$

WDF

Sum of weights minus one

The following is the formula for the divisor:

$$(\sum_i w_i) - 1$$

WEIGHT

Sum of weights

The following is the formula for the divisor:

$$\sum_i w_i$$

Alias WGT

The procedure computes the variance as $CSS/divisor$, where CSS is the corrected sums of squares and equals $\sum (x_i - \bar{x})^2$. When you weight the analysis variables, CSS equals $\sum w_i(x_i - \bar{x}_w)^2$ where $\bar{x}_w$ is the weighted mean.

Default DF

Requirement To compute standard error of the mean, use the default value of VARDEF=.

Tips When you use the WEIGHT statement and VARDEF=DF, the variance is an estimate of σ^2 , where the variance of the i th observation is $var(x_i) = \sigma^2/w_i$ and w_i is the weight for the i th observation. This yields an estimate of the variance of an observation with unit weight.

When you use the WEIGHT statement and VARDEF=WGT, the computed variance is asymptotically (for large n) an estimate of $\sigma^2/\bar{w}$, where $\bar{w}$ is the average weight. This yields an asymptotic estimate of the variance of an observation with average weight.

ATTRIB Statement

Associates a format, informat, label, and length with one or more variables.

Syntax

ATTRIB *variable-list(s) attribute-list(s);*

Arguments

variable-list(s)

names the variables that you want to associate with the attributes.

TIP List the variables in any form that SAS allows.

attribute-list(s)

specifies one or more attributes to assign to *variable-list*. Specify one or more of these attributes in the ATTRIB statement:

FORMAT=format

associates a format with variables in *variable-list*.

Tip The format can be either a standard SAS format or a format that is defined with the FORMAT procedure.

INFORMAT=informal

associates an informat with variables in *variable-list*.

Tip The informat can be either a standard SAS informat or an informat that is defined with the FORMAT procedure.

LABEL='label'

associates a label with variables in *variable-list*.

See Also

[“ATTRIB Statement” in SAS DATA Step Statements: Reference.](#)

BY Statement

Orders the output according to the BY groups.

Restriction: Do not specify a group variable in the BY statement. The BY variable cannot be the same as the group variable. A *group variable* is a variable in the input data set that is used to categorize chart variable values into groups. For more information, see [“Group Variables” on page 1760](#).

Syntax

```
BY <DESCENDING> variable-1
    <... <DESCENDING> variable-n>
    <NOTSORTED>;
```

Required Argument

variable

specifies the variable that the procedure uses to form BY groups. You can specify more than one variable. If you do not use the NOTSORTED option in the BY statement, then either the observations in the data set must be sorted by all the variables that you specify, or they must be indexed appropriately. Variables in a BY statement are called *BY variables*.

Optional Arguments

DESCENDING

specifies that the observations are sorted in descending order by the variable that immediately follows the word DESCENDING in the BY statement.

NOTSORTED

specifies that observations are not necessarily sorted in alphabetic or numeric order. The observations are grouped in another way (for example, chronological order).

The requirement for ordering or indexing observations according to the values of BY variables is suspended for BY-group processing when you use the NOTSORTED option. In fact, the procedure does not use an index if you specify NOTSORTED. The procedure defines a BY group as a set of contiguous observations that have the same values for all BY variables. If observations with the same values for the BY variables are not contiguous, then the procedure treats each contiguous set as a separate BY group.

You cannot use the NOTSORTED option in a PROC SORT step.

Note: You cannot use the GROUPFORMAT option, which is available in the BY statement in a DATA step, in a BY statement in any PROC step.

Details

BY-Group Processing

Procedures create output for each BY group. For example, the elementary statistics procedures and the scoring procedures perform separate analyses for each BY group. The reporting and ODS procedures produce a report for each BY group.

Note: All Base SAS procedures except PROC PRINT process BY groups independently. PROC PRINT can report the number of observations in each BY group as well as the number of observations in all BY groups. Similarly, PROC PRINT can sum numeric variables in each BY group and across all BY groups.

You can use only one BY statement in each PROC step. When you use a BY statement, the procedure expects an input data set that is sorted by the order of the BY variables or one that has an appropriate index. If your input data set does not meet these criteria, then an error occurs. Either sort it with the SORT procedure or create an appropriate index on the BY variables.

Depending on the order of your data, you might need to use the NOTSORTED or DESCENDING option in the BY statement in the PROC step.

- For more information about the BY statement, see [SAS DATA Step Statements: Reference](#).
- For more information about PROC SORT, see [Chapter 55, “SORT Procedure,” on page 2039](#).
- For more information about BY-group processing in CAS, see [“Grouping and Ordering” in SAS Cloud Analytic Services: DATA Step Programming](#).
- For more information about creating indexes, see [“INDEX CREATE Statement” on page 440](#).

Formatting BY-Variable Values

When a procedure is submitted with a BY statement, the following actions are taken with respect to processing of BY groups:

- 1 The procedure determines whether the data is sorted by the internal (unformatted) values of the BY variable(s).
- 2 The procedure determines whether a format has been applied to the BY variable(s). If the BY variable is numeric and has no user-applied format, then the BEST12. format is applied for the purpose of BY-group processing.
- 3 The procedure continues adding observations to the current BY group until both the internal and formatted values of the BY variable or variables change.

This process can have unexpected results if, for example, nonconsecutive internal BY values share the same formatted value. In this case, the formatted value is represented in different BY groups. Alternatively, if different consecutive internal BY values share the same formatted value, then these observations are grouped into the same BY group.

BY Variables That Have Different Lengths in Two Data Sets

When a procedure has a BY statement and two input data sets, one of the input data sets is called the primary data set and the other is called the secondary data set. The primary data set is usually, but not always, the DATA= data set. A BY statement always applies to the primary data set. The variables in the BY statement must appear in the primary data set.

Each procedure determines whether a BY statement applies to a secondary data set, and performs one of the following actions:

- The procedure might always apply the BY statement to the secondary data set. In this case, one or more variables in the BY statement must appear in the secondary data set.
- The procedure might never apply the BY statement to the secondary data set. In this case, the variables in the BY statement are not required in the secondary data set.
- The procedure might check whether the BY variables are in the secondary data set. If none of the BY variables are in the secondary data set, then BY processing does not occur for the secondary data set. If one or more of the BY variables are in the secondary data set, and they match the BY variables in the primary data set, then BY processing is done for the secondary data set. If some but not all BY variables are in the secondary data set, then the procedure might issue an error message and quit. Or, it might take some other action described in the documentation for that particular procedure.

If the BY statement is applied to the secondary data set, then each BY variable that exists on both the data sets must have the same type, character or numeric, in both data sets. The BY variables are required to have either the same formatted value or the same unformatted value. Formatted values match only if both the formatted lengths and the formatted values are the same. Unformatted values are not required to have the same length in order to match. The unformatted character

values match if the unformatted values are the same after stripping the trailing blanks. The unformatted doubles match if they have the same value.

A secondary data set does not need to have all of the BY variables that are in the primary data set. A procedure can define a subset of the BY variables for the secondary data set. For example, if the primary data set has the BY variables A,B,C,D, then the procedure can define the following BY variables on the secondary data set:

- A
- A,B
- A,B,C
- A,B,C,D

If both the primary and secondary data sets have the same number of BY variables, and all the BY variables have the same byte lengths and format lengths, then either the unformatted values or the formatted values in the BY buffer (for all of the BY variables) have to match. If they do not match, then each variable is compared. The formatted values of each variable are compared first. The formatted lengths have to match, and the formatted values have to match. If the formatted lengths and values do not match, then the unformatted values are compared even if the byte lengths are different.

If corresponding character variable lengths differ, then the longer character variable can contain only trailing blanks for the extra characters. If the lengths of the character variables are different, then the values match as long as they are the same after stripping the trailing blanks. For example, 'ABCD' in the primary data set matches 'ABCD ' in the secondary data set. If the secondary data set contained 'ABCDEF', then they would not match.

Base SAS Procedures That Support the BY Statement

COMPARE	SORT (required)
CORR	STANDARD
FREQ	SUMMARY
MEANS	TABULATE
PRINT	TRANSPOSE
RANK	UNIVARIATE
REPORT (nonwindowing environment only)	ODSTEXT
ODSLIST	ODSTABLE

Note: In the SORT procedure, the BY statement specifies how to sort the data. In the other procedures, the BY statement specifies how the data is currently sorted.

See Also

[“BY Statement” in SAS DATA Step Statements: Reference](#)

CLASS Statement

Identifies class variables for the table. Class variables determine the categories that PROC TABULATE uses to calculate statistics.

Alias: CLASSES

Note: CLASS statements without options use the internal default or the value specified by an option in the PROC TABULATE statement. For example, in the following code, variables c and d would use the internal default. If an ORDER= option had been specified in the PROC TABULATE statement, then variables c and d would use the value specified by the ORDER= option in the PROC TABULATE statement.

```
class a b / order=data;
class c d;
```

Tips: You can use multiple CLASS statements.
Some CLASS statement options are also available in the PROC TABULATE statement. They affect all CLASS variables rather than just the ones that you specify in a CLASS statement.

Examples: [“Example 3: Using Preloaded Formats with Class Variables” on page 2311](#)
[“Example 4: Using Multilabel Formats” on page 2315](#)

Syntax

CLASS *variable(s)* *</ options>*;

Summary of Optional Arguments

ASCENDING

specifies to sort the class variable values in ascending order.

DESCENDING

specifies to sort the class variable values in descending order.

EXCLUSIVE

excludes from tables and output data sets all combinations of class variables that are not found in the preloaded range of user-defined formats.

GROUPINTERNAL

specifies not to apply formats to the class variables when PROC TABULATE groups the values to create combinations of class variables.

MISSING

considers missing values as valid class variable levels. Special missing values that represent numeric values (the letters A through Z and the underscore (_) character) are each considered as a separate value.

MLF

enables PROC TABULATE to use the format label or labels for a given range or overlapping ranges to create subgroup combinations when a multilabel format is assigned to a class variable.

ORDER=DATA | FORMATTED | FREQ | UNFORMATTED

specifies the order to group the levels of the class variables in the output.

PRELOADFMT

specifies that all formats are preloaded for the class variables.

STYLE=style-override

specifies one or more style overrides to use for page dimension text and class variable name headings.

Required Argument

variable(s)

specifies one or more variables that the procedure uses to group the data. Variables in a CLASS statement are referred to as *class variables*. Class variables can be numeric or character. Class variables can have continuous values, but they typically have a few discrete values that define the classifications of the variable. You do not have to sort the data by class variables.

Interaction If a variable name and a statistic name are the same, enclose the statistic name in single or double quotation marks.

Options to Use after a Slash Delimiter

ASCENDING

specifies to sort the class variable values in ascending order.

Alias ASCEND

Interaction PROC TABULATE issues a warning message if you specify both ASCENDING and DESCENDING and ignores both options.

DESCENDING

specifies to sort the class variable values in descending order.

Alias DESCEND

Default ASCENDING

Interaction PROC TABULATE issues a warning message if you specify both ASCENDING and DESCENDING and ignores both options.

EXCLUSIVE

excludes from tables and output data sets all combinations of class variables that are not found in the preloaded range of user-defined formats.

Requirement You must specify the PRELOADFMT option in the CLASS statement to preload the class variable formats.

Example [“Example 3: Using Preloaded Formats with Class Variables” on page 2311](#)

GROUPINTERNAL

specifies not to apply formats to the class variables when PROC TABULATE groups the values to create combinations of class variables.

Interactions If you specify the PRELOADFMT option in the CLASS statement, then PROC TABULATE ignores the GROUPINTERNAL option and uses the formatted values.

If you specify the ORDER=FORMATTED option, then PROC TABULATE ignores the GROUPINTERNAL option and uses the formatted values.

Tip The GROUPINTERNAL option saves computer resources when the class variables contain discrete numeric values.

MISSING

considers missing values as valid class variable levels. Special missing values that represent numeric values (the letters A through Z and the underscore (_) character) are each considered as a separate value.

Default If you omit the MISSING option, then PROC TABULATE excludes the observations with any missing CLASS variable values from tables and output data sets.

See [“Example: Creating Special Missing Values” in SAS Programmer’s Guide: Essentials](#) for a discussion of missing values with special meanings.

MLF

enables PROC TABULATE to use the format label or labels for a given range or overlapping ranges to create subgroup combinations when a multilabel format is assigned to a class variable.

Note: When the formatted values overlap, one internal class variable value maps to more than one class variable subgroup combination. Therefore, the sum of the N statistics for all subgroups is greater than the number of observations in the data set (the overall N statistic).

Requirement You must use PROC FORMAT and the MULTILABEL option in the VALUE statement to create a multilabel format.

- Interactions** Using MLF with ORDER=FREQ might not produce the order that you expect for the formatted values.
- When you specify MLF, the formatted values of the class variable become internal values. Therefore, specifying ORDER=FORMATTED produces the same results as specifying ORDER=UNFORMATTED.
- Tip** If you omit MLF, then PROC TABULATE uses the primary format labels, which correspond to the first external format value, to determine the subgroup combinations.
- See** [“MULTILABEL” on page 913](#) in the VALUE statement of the FORMAT procedure.
- Example** [“Example 4: Using Multilabel Formats” on page 2315](#)

ORDER=DATA | FORMATTED | FREQ | UNFORMATTED

specifies the order to group the levels of the class variables in the output.

DATA

orders values according to their order in the input data set.

- Interaction** If you use PRELOADFMT, then the order for the values of each class variable matches the order that PROC FORMAT uses to store the values of the associated user-defined format. If you use the CLASSDATA= option in the PROC statement, then PROC TABULATE uses the order of the unique values of each class variable in the CLASSDATA= data set to order the output levels. If you use both options, then PROC TABULATE first uses the user-defined formats to order the output. If you omit EXCLUSIVE in the PROC statement, then PROC TABULATE places, in the order in which they are encountered, the unique values of the class variables that are in the input data set after the user-defined format and the CLASSDATA= values.
- Tip** By default, PROC FORMAT stores a format definition in sorted order. Use the NOTSORTED option to store the values or ranges of a user-defined format in the order in which you define them.

FORMATTED

orders values by their ascending formatted values. This order depends on your operating environment.

Aliases FMT

EXTERNAL

FREQ

orders values by descending frequency count.

- Interaction** Use the ASCENDING option to order values by ascending frequency count.

UNFORMATTED

orders values by their unformatted values, which yields the same order as PROC SORT. This order depends on your operating environment. This sort sequence is particularly useful for displaying dates chronologically.

Aliases UNFMT

INTERNAL

Default UNFORMATTED

Interaction If you use the PRELOADFMT option in the CLASS statement, then PROC TABULATE orders the levels by the order of the values in the user-defined format.

Tip By default, all orders except FREQ are ascending. For descending orders, use the DESCENDING option.

Example [“Understanding the Order of Headings with ORDER=DATA” on page 2304](#)

PRELOADFMT

specifies that all formats are preloaded for the class variables.

Requirement PRELOADFMT has no effect unless you specify EXCLUSIVE, ORDER=DATA, or PRINTMISS and you assign formats to the class variables. If you specify PRELOADFMT without also specifying EXCLUSIVE, ORDER=DATA, or PRINTMISS, then SAS writes a warning message to the SAS log.

Interactions To limit PROC TABULATE output to the combinations of formatted class variable values present in the input data set, use the EXCLUSIVE option in the CLASS statement.

To include all ranges and values of the user-defined formats in the output, use the PRINTMISS option in the TABLE statement. Use care when you use PRELOADFMT with PRINTMISS. This feature creates all possible combinations of formatted class variables. Some of these combinations might not make sense.

Note When preloading a character format, there must be at least one representative raw value that produces the label. If a range does not have any raw values, the range is considered unreachable and an error is generated. For example, you cannot preload a multi-label character format with both Other= and Low-High ranges. Other is a special keyword that refers to values or ranges that remain after all labels have been defined. For character formats, missing values are included in the low range so that there are no raw values left to occupy the Other range.

Example [“Example 3: Using Preloaded Formats with Class Variables” on page 2311](#)

STYLE=style-override

specifies one or more style overrides to use for page dimension text and class variable name headings. For example, the following statement specifies that the background color for page dimension text and class variable name headings is light green:

```
class region division prodtype / style=[background=lightgreen];
```

style-override

specifies one or more style attributes or style elements to override the default style element and attributes in a specific area of a report.

You can specify a style override in three ways:

- Specify a style element. A style element is a collection of style attributes that apply to a particular part of the output for a SAS program.
- Specify a style attribute. A style attribute is a name-value pair that describes a single behavioral or visual aspect of a piece of output. This is the most specific method of changing the appearance of your output.
- Specify the PARENT value. The PARENT value specifies that the data cell use the style element of its parent heading.

style-override has the following form:

PARENT | *style-element-name* | [*style-attribute-name-1*=*style-attribute-value-1*

<*style-attribute-name-2*=*style-attribute-value-2* ...>]

<PARENT>

specifies that the data cell use the style element of its parent heading.

The parent style element of a data cell is one of the following:

- the style element of the leaf heading above the column that contains the data cell, if the table specifies no row dimension, or if the table specifies the style element in the column dimension expression
- the style element of the leaf heading above the row that contains the cell, if the table specifies the style element in the row dimension expression
- the Beforecaption style element, if the table specifies the style element in the page dimension expression
- undefined, otherwise

Note: In this usage, the angle brackets around the word PARENT are required. Braces or square brackets cannot be substituted in the syntax.

Note: The parent of a heading (not applicable to STYLE= in the PROC TABULATE statement) is the heading under which the current heading is nested.

style-attribute-name

specifies the attribute to change.

See For information about using styles with PROC TABULATE, see [“Using ODS Styles with PROC TABULATE” on page 2278](#).

For a table of default style attributes and style elements for each ODS destination, see [“Style Elements and Style Attributes for Table Regions” on page 2286](#).

style-attribute-value

specifies a value for the attribute. Each attribute has a different set of valid values. A SAS format can also be used as an attribute value for conditional formatting.

See For information about using styles with PROC TABULATE, see [“Using ODS Styles with PROC TABULATE” on page 2278](#).

For information about using SAS formats as style attribute values, see [“Using a Format to Assign a Style Attribute” on page 2290](#).

For a table of default style attributes and style elements for each ODS destination, see [“Style Elements and Style Attributes for Table Regions” on page 2286](#).

style-element-name

is the name of a style element that is part of an ODS style template.

See For information about using styles with PROC TABULATE, see [“Using ODS Styles with PROC TABULATE” on page 2278](#).

For a table of default style attributes and style elements for each ODS destination, see [“Style Elements and Style Attributes for Table Regions” on page 2286](#).

Alias S=

Tips To override a style element that is specified for page dimension text in the CLASS statement, you can specify a style element in the TABLE statement page dimension expression.

To override a style element that is specified for a class variable name heading in the CLASS statement, you can specify a style element in the related TABLE statement dimension expression.

If a page dimension expression contains multiple nested elements, then the Beforecaption style element is the style element of the first element in the nesting.

The use of STYLE= in the CLASS statement differs slightly from its use in the PROC TABULATE statement. In the CLASS statement, inheritance is different for rows and columns. For rows, the parent heading is located to the left of the current heading. For columns, the parent heading is located above the current heading.

See For information about using styles with PROC TABULATE, see [“Using ODS Styles with PROC TABULATE” on page 2278](#).

Example [“Example 13: Specifying Style Overrides for ODS Output” on page 2357](#)

Details

How PROC TABULATE Handles Missing Values for Class Variables

By default, if an observation contains a missing value for any class variable, then PROC TABULATE excludes that observation from all tables that it creates. CLASS statements apply to all TABLE statements in the PROC TABULATE step. Therefore, if you define a variable as a class variable, then PROC TABULATE omits observations that have missing values for that variable from every table even if the variable does not appear in the TABLE statement for one or more tables.

If you specify the MISSING option in the PROC TABULATE statement, then the procedure considers missing values as valid levels for all class variables. If you specify the MISSING option in a CLASS statement, then PROC TABULATE considers missing values as valid levels for the class variables that are specified in that CLASS statement.

CLASSLEV Statement

Specifies a style element for class variable level value headings.

Aliases: CLASSLEVEL
CLASSLEVELS

Examples: [“Example 13: Specifying Style Overrides for ODS Output” on page 2357](#)
[“Example 14: Style Precedence” on page 2362](#)

Syntax

CLASSLEV *variable(s)* </ STYLE=*style-override(s)*>;

Required Argument

variable(s)

specifies one or more class variables from the CLASS statement for which you want to specify a style element.

Options to Use after a Slash Delimiter

STYLE=style-override

specifies one or more style overrides for class variable level value headings. For example, the following statement specifies that the background color for class variable level value headings is yellow:

```
classlev region division prodtype / style=[background=yellow];
```

style-override

specifies one or more style attributes or style elements to override the default style element and attributes in a specific area of a report.

You can specify a style override in three ways:

- Specify a style element. A style element is a collection of style attributes that apply to a particular part of the output for a SAS program.
- Specify a style attribute. A style attribute is a name-value pair that describes a single behavioral or visual aspect of a piece of output. This is the most specific method of changing the appearance of your output.
- Specify the PARENT value. The PARENT value specifies that the data cell use the style element of its parent heading.

style-override has the following form:

PARENT | *style-element-name* | [*style-attribute-name-1*=*style-attribute-value-1*

<*style-attribute-name-2*=*style-attribute-value-2* ...>]

<PARENT>

specifies that the data cell use the style element of its parent heading.

The parent style element of a data cell is one of the following:

- the style element of the leaf heading above the column that contains the data cell, if the table specifies no row dimension, or if the table specifies the style element in the column dimension expression
- the style element of the leaf heading above the row that contains the cell, if the table specifies the style element in the row dimension expression
- the Beforecaption style element, if the table specifies the style element in the page dimension expression
- undefined, otherwise

Note: In this usage, the angle brackets around the word PARENT are required. Braces or square brackets cannot be substituted in the syntax.

Note: The parent of a heading (not applicable to STYLE= in the PROC TABULATE statement) is the heading under which the current heading is nested.

style-attribute-name

specifies the attribute to change.

See For information about using styles with PROC TABULATE, see [“Using ODS Styles with PROC TABULATE” on page 2278](#).

For a table of default style attributes and style elements for each ODS destination, see [“Style Elements and Style Attributes for Table Regions” on page 2286](#).

style-attribute-value

specifies a value for the attribute. Each attribute has a different set of valid values. A SAS format can also be used as an attribute value for conditional formatting.

See For information about using styles with PROC TABULATE, see [“Using ODS Styles with PROC TABULATE” on page 2278](#).

For information about using SAS formats as style attribute values, see [“Using a Format to Assign a Style Attribute” on page 2290](#).

For a table of default style attributes and style elements for each ODS destination, see [“Style Elements and Style Attributes for Table Regions” on page 2286](#).

style-element-name

is the name of a style element that is part of an ODS style template.

See For information about using styles with PROC TABULATE, see [“Using ODS Styles with PROC TABULATE” on page 2278](#).

For a table of default style attributes and style elements for each ODS destination, see [“Style Elements and Style Attributes for Table Regions” on page 2286](#).

Alias S=

Tips The use of STYLE= in the CLASSLEV statement differs slightly from its use in the PROC TABULATE statement. In the CLASSLEV statement, inheritance is different for rows and columns. For rows, the parent heading is located to the left of the current heading. For columns, the parent heading is located above the current heading.

To override a style element that is specified in the CLASSLEV statement, you can specify a style element in the related TABLE statement dimension expression.

You can use braces ({ and }) instead of square brackets ([and]).

See For information about using styles with PROC TABULATE, see [“Using ODS Styles with PROC TABULATE” on page 2278](#).

Example [“Example 13: Specifying Style Overrides for ODS Output” on page 2357](#)

FORMAT Statement

Associates formats with variables.

Syntax

FORMAT *variable-1* <...*variable-n*> *format variable-1* <...*variable-n*> *format*;

Arguments

variable

names one or more variables for SAS to associate with a format. You must specify at least one *variable*.

Tip To disassociate a format from a variable, use the variable in a FORMAT statement without specifying a format in a DATA step or in PROC DATASETS. In a DATA step, place this FORMAT statement after the SET statement. See [“Removing a Format” in SAS DATA Step Statements: Reference](#). You can also use PROC DATASETS.

format

specifies the format that is listed for writing the values of the variables.

Tip Formats that are associated with variables by using a FORMAT statement behave like formats that are used with a colon modifier in a subsequent PUT statement. For more information about using a colon modifier, see [“PUT Statement: List” in SAS DATA Step Statements: Reference](#).

See [SAS Formats and Informats: Reference](#)

FREQ Statement

Specifies a numeric variable that contains the frequency of each observation.

Tip: The effects of the FREQ and WEIGHT statements are similar except when calculating degrees of freedom.

Syntax

FREQ *variable*;

Required Argument

variable

specifies a numeric variable whose value represents the frequency of the observation. If you use the FREQ statement, then the procedure assumes that each observation represents

n observations, where n is the value of *variable*. If *variable* is not an integer, then SAS truncates it. If *variable* is less than 1 or is missing, then the procedure does not use that observation to calculate statistics. If a FREQ statement does not appear, then each observation has a default frequency of 1.

The sum of the frequency variable represents the total number of observations.

KEYLABEL Statement

Labels a keyword for the duration of the PROC TABULATE step. PROC TABULATE uses the label anywhere that the specified keyword would otherwise appear.

Syntax

KEYLABEL *keyword-1*='description-1' <*keyword-2*='description-2' ...>;

Required Arguments

keyword

specifies a statistic keyword.

keyword can be one of the keywords for statistics that is discussed in [“Statistics That Are Available in PROC TABULATE” on page 2271](#) or is the universal class variable ALL. (See [“Elements That You Can Use in a Dimension Expression” on page 2260](#).)

description

specifies the keyword label.

Length	256 characters
Restriction	Each keyword can have only one label in a particular PROC TABULATE step. If you request multiple labels for the same keyword, then PROC TABULATE uses the last one that is specified in the step.
Requirement	You must enclose <i>description</i> in quotation marks

KEYWORD Statement

Specifies a style element for keyword headings.

Alias: KEYWORDS

Example: [“Example 13: Specifying Style Overrides for ODS Output” on page 2357](#)

Syntax

KEYWORD *keyword(s)* **</ STYLE=***style-override(s)***>;**

Required Argument

keyword

specifies the keyword statistic.

keyword can be one of the keywords for statistics that is discussed in [“Statistics That Are Available in PROC TABULATE” on page 2271](#) or is the universal class variable ALL. (See [“Elements That You Can Use in a Dimension Expression” on page 2260](#).)

Options to Use after a Slash Delimiter

STYLE=*style-override*

specifies one or more style overrides for the keyword headings.

For example, the following statement specifies that the background color for keyword headings is linen:

```
keyword all sum / style=[background=linen];
```

style-override

specifies one or more style attributes or style elements to override the default style element and attributes in a specific area of a report.

You can specify a style override in three ways:

- Specify a style element. A style element is a collection of style attributes that apply to a particular part of the output for a SAS program.
- Specify a style attribute. A style attribute is a name-value pair that describes a single behavioral or visual aspect of a piece of output. This is the most specific method of changing the appearance of your output.
- Specify the PARENT value. The PARENT value specifies that the data cell use the style element of its parent heading.

style-override has the following form:

```
PARENT | style-element-name | [style-attribute-name-1=style-attribute-value-1
```

`<style-attribute-name-2=style-attribute-value-2 ...>`]

<PARENT>

specifies that the data cell use the style element of its parent heading.

The parent style element of a data cell is one of the following:

- the style element of the leaf heading above the column that contains the data cell, if the table specifies no row dimension, or if the table specifies the style element in the column dimension expression
- the style element of the leaf heading above the row that contains the cell, if the table specifies the style element in the row dimension expression
- the Beforecaption style element, if the table specifies the style element in the page dimension expression
- undefined, otherwise

Note: In this usage, the angle brackets around the word PARENT are required. Braces or square brackets cannot be substituted in the syntax.

Note: The parent of a heading (not applicable to STYLE= in the PROC TABULATE statement) is the heading under which the current heading is nested.

style-attribute-name

specifies the attribute to change.

See For information about using styles with PROC TABULATE, see [“Using ODS Styles with PROC TABULATE” on page 2278](#).

For a table of default style attributes and style elements for each ODS destination, see [“Style Elements and Style Attributes for Table Regions” on page 2286](#).

style-attribute-value

specifies a value for the attribute. Each attribute has a different set of valid values. A SAS format can also be used as an attribute value for conditional formatting.

See For information about using styles with PROC TABULATE, see [“Using ODS Styles with PROC TABULATE” on page 2278](#).

For information about using SAS formats as style attribute values, see [“Using a Format to Assign a Style Attribute” on page 2290](#).

For a table of default style attributes and style elements for each ODS destination, see [“Style Elements and Style Attributes for Table Regions” on page 2286](#).

style-element-name

is the name of a style element that is part of an ODS style template.

See For information about using styles with PROC TABULATE, see [“Using ODS Styles with PROC TABULATE” on page 2278](#).

For a table of default style attributes and style elements for each ODS destination, see [“Style Elements and Style Attributes for Table Regions” on page 2286](#).

Alias S=

Tips The use of STYLE= in the KEYWORD statement differs slightly from its use in the PROC TABULATE statement. In the KEYWORD statement, inheritance is different for rows and columns. For rows, the parent heading is located to the left of the current heading. For columns, the parent heading is located above the current heading.

To override a style element that is specified in the KEYWORD statement, you can specify a style element in the related TABLE statement dimension expression.

See For information about using styles with PROC TABULATE, see [“Using ODS Styles with PROC TABULATE” on page 2278](#).

Example [“Example 13: Specifying Style Overrides for ODS Output” on page 2357](#)

LABEL Statement

Assigns descriptive labels to variables.

Syntax

LABEL *variable-1* = '*text-string*' <...*variable-n* = '*text-string*'>;

LABEL *variable-1* = '' <...*variable-n*=''>; | *variable-1* = <...*variable-n*=>;

Arguments

variable

specifies the variable that you want to label. You can specify labels for multiple variables in a single LABEL statement:

```
data test;
  L = 5;
  W = 10;
  label L = 'Length' W = 'Width';
run;
```

text-string

specifies a label of up to 256 bytes.

```
data test;
  L = 5;
  label L = 'Length';
run;
```

Restrictions If the label includes a semicolon (;) or an equal sign (=), you must enclose the label in either single or double quotation marks.

```
label N = 'Number = ';
```

If the label includes single quotation marks ('), you must enclose the label in double quotation marks.

```
label Name = "Owner's Last Name";
```

There is partial support for variable name substitution with the JAVAIMG device. When the label text for the variable exceeds the space allowed, the variable name is used instead by procedures that are labeling an axis or a legend title. With the exception of the SAS/GRAPH GPLOT procedure, the variable name is not used when the JAVAIMG device is specified.

Notes Quotation marks are not required unless the label includes a semicolon, quotation marks, or an equal sign.

The LABEL statement expects an equal sign (=) after the first variable. If any other character is found, an error occurs. The LABEL statement considers everything after the equal sign, following the first variable, as part of the label until the statement comes to another variable followed by an equal sign, regardless of quotation marks. In this example, the label for x is **this is x y 4 = this is y** because the next variable that is followed by an equal sign is z.

```
data test;
  x = 1;
  y = 1;
  z = 1;
  label x = 'this is x' y 4 = 'this is y' z = 'This is z';
run;
```

In this LABEL statement, the x variable has the same label:

```
label x = this is x y 4 = this is y z = This is z;
```

See [“Character Constants and Character Variables” in SAS Programmer’s Guide: Essentials.](#)

”

removes a label from a variable. Enclose a single blank space in quotation marks to remove an existing label. You can remove labels from multiple variables in a single LABEL statement:

```
data test;
  L=5; W=10;
  label L = ' ' W = ' ';
run;
```

Alternatively, you can remove a label by specifying nothing after the equal sign:

```
data test;
  L=5; W=10;
  label L = W = ;
run;
```

TABLE Statement

Describes a table to be printed.

Alias: TABLES

Requirement: All variables in the TABLE statement must appear in either the VAR statement or the CLASS statement.

Tips: To create several tables use multiple TABLE statements.
Use of variable name list shortcuts is now supported within the TABLE statement. For more information, see [“Shortcuts for Specifying Lists of Variable Names” on page 61](#).

Syntax

TABLE *<<page-expression> row-expression,>
column-expression </ table-options>;*

Summary of Optional Arguments

BOX=*<label=value> <STYLE=style-override(s)>*

specifies text and a style override for the empty box above the row titles.

BOX=*value*

specifies text for the empty box above the row titles.

CAPTION= *text<#BYLINE> <#BYVAL><#BYVAR>*

specifies a caption text string to add before each table.

CONDENSE

prints as many complete logical pages as possible on a single printed page or, if possible, prints multiple pages of tables that are too wide to fit on a page one below the other on a single page, instead of on separate pages.

CONTENTS=*link-text <#BYLINE> <#BYVAL> <#BYVAR>*

enables you to name the link in the HTML table of contents that points to the ODS output of the table that is produced by using the TABLE statement.

FORMAT_PRECEDENCE=PAGE | ROW | COLUMN

specifies whether the format that is specified for the page dimension (PAGE), row dimension (ROW), or column dimension (COLUMN) is applied to the contents of the table cells.

FUZZ=number

supplies a numeric value against which table cell values other than frequency counts are compared to eliminate trivial values (absolute values less than the FUZZ= value) from printing.

INDENT=number-of-spaces

specifies the number of spaces to indent nested row headings, and suppresses the row headings for class variables.

MISSTEXT='text'

supplies up to 256 characters of text to be printed for table cells that contain missing values.

MISSTEXT={<label='text'> <STYLE=style-override(s)>}

supplies up to 256 characters of text to be printed and specifies a style override for table cells that contain missing values.

NOCELLMERGE

specifies that data cells are not merged with other data cells in the table.

NOCONTINUED

suppresses the continuation message, **continued**, that is displayed at the bottom of tables that span multiple pages.

page-expression

defines the pages in a table.

PRINTMISS

prints all values that occur for a class variable each time headings for that variable are printed, even if there are no data for some of the cells that these headings create.

ROW=CONSTANT | FLOAT

specifies whether all title elements in a row crossing are allotted space even when they are blank.

row-expression

defines the rows in the table.

RTSPACE=number

specifies the number of print positions to allot to all of the headings in the row dimension, including spaces that are used to print outlining characters for the row headings.

STYLE_PRECEDENCE=PAGE | ROW | COLUMN

specifies whether the style that is specified for the page dimension (PAGE), row dimension (ROW), or column dimension (COLUMN or COL) is applied to the contents of the table cells.

STYLE=style-override

specifies one or more style overrides to use for parts of the table other than table cells.

Required Argument

column-expression

defines the columns in the table. For information about constructing dimension expressions, see [“Details” on page 2260](#).

Restriction A column dimension is the last dimension in a TABLE statement. A row dimension or a row dimension and a page dimension can precede a column dimension.

Optional Arguments

page-expression

defines the pages in a table. For information about constructing dimension expressions, see [“Details” on page 2260](#).

Restrictions A page dimension is the first dimension in a table statement. Both a row dimension and a column dimension must follow a page dimension.

If the ACCESSIBLETABLE=ON system option is specified and the TABULATE table has page dimension text, then that text is the accessible caption. Otherwise use the CAPTION=/ CONTENTS= text.

Accessibility note You can use the ACCESSIBLECHECK and ACCESSIBLETABLE system options to check for and create accessible tables starting with SAS 9.4M6. For information about creating accessible PROC TABULATE tables, see [“Creating Accessible Tables” in Creating Accessible SAS Viya Platform Output Using ODS and ODS Graphics](#). If the ACCESSIBLETABLE=ON system option is specified and the TABULATE table has page dimension text, then that text is the accessible caption.

Example [“Example 8: Creating Multipage Tables” on page 2325](#)

row-expression

defines the rows in the table. For information about constructing dimension expressions, see [“Details” on page 2260](#).

Restriction A row dimension is the next to last dimension in a table statement. A column dimension must follow a row dimension. A page dimension can precede a row dimension.

Table Options

BOX=value

specifies text for the empty box above the row titles.

Value can be one of the following:

PAGE

writes the page-dimension text in the box. If the page-dimension text does not fit, then it is placed in its default position above the box, and the box remains empty.

BOX={<label=value> <STYLE=style-override(s)>}

specifies text and a style override for the empty box above the row titles.

Value can be one of the following:

PAGE

writes the page-dimension text in the box. If the page-dimension text does not fit, then it is placed in its default position above the box, and the box remains empty.

'string'

writes the quoted string in the box. Any string that does not fit in the box is truncated.

variable

writes the name (or label, if the variable has one) of a variable in the box. Any name or label that does not fit in the box is truncated.

For details about the arguments of the STYLE= option and how it is used, see [STYLE= on page 2258](#) in the TABLE statement.

Examples [“Example 8: Creating Multipage Tables” on page 2325](#)

[“Example 13: Specifying Style Overrides for ODS Output” on page 2357](#)

CAPTION= text<#BYLINE> <#BYVAL><#BYVAR>

specifies a caption text string to add before each table. If no string is specified, then the caption defaults to the text specified by the CONTENTS= option in the TABLE statement.

This option makes table captions both visual and accessible if the ACCESSIBLETABLE system option is specified.

If the NOACCESSIBLETABLE system option is specified, then no caption is displayed, however the caption text is still accessible.

These options are available if a BY statement is in effect:

#BYLINE

substitutes the entire BY line without leading or trailing blanks for #BYLINE in the text string. The BY line uses the format *variable-name=value*.

#BYVAL_n**#BYVAL(BY-variable-name)**

substitutes the current value of the specified BY variable for #BYVAL in the text string.

Specify the variable with one of the following:

n

specifies a variable by its position in the BY statement. For example, #BYVAL2 specifies the second variable in the BY statement.

BY-variable-name

specifies a variable from the BY statement by its name. For example, #BYVAL(YEAR) specifies the BY variable, YEAR. *Variable-name* is not case sensitive.

#BYVAR n **#BYVAR(*BY-variable-name*)**

substitutes the name of the BY-variable or the label associated with the variable (whatever the BY line would normally display) for #BYVAR in the text string.

Specify the variable with one of the following:

 n

specifies a variable by its position in the BY statement. For example, #BYVAR2 specifies the second variable in the BY statement.

BY-variable-name

specifies a variable from the BY statement by its name. For example, #BYVAR(SITES) specifies the BY variable, SITES. *Variable-name* is not case sensitive.

Restriction The ACCESSIBLETABLE system option must be specified to enable captions.

Tip You can use the PROC DOCUMENT OBBNOTE option to display or edit the caption.

See [“Creating Accessible Tables” in *Creating Accessible SAS Viya Platform Output Using ODS and ODS Graphics*](#)

CONDENSE

A *logical page* is all the rows and columns that fall within one of the following:

- a page-dimension category (with no BY-group processing)
- a BY group with no page dimension
- a page-dimension category within a single BY group

Restriction The CONDENSE option has no effect on the pages that are generated by the BY statement. The first table for a BY group always begins on a new page.

Example [“Example 8: Creating Multipage Tables” on page 2325](#)

CONTENTS=*link-text* <#BYLINE> <#BYVAL> <#BYVAR>

enables you to name the link in the HTML table of contents that points to the ODS output of the table that is produced by using the TABLE statement.

These options are available if a BY statement is in effect:

#BYLINE

substitutes the entire BY line without leading or trailing blanks for #BYLINE in the text string. The BY line uses the format *variable-name=value*.

#BYVAL n **#BYVAL(*BY-variable-name*)**

substitutes the current value of the specified BY variable for #BYVAL in the text string.

Specify the variable with one of the following:

n

specifies a variable by its position in the BY statement. For example, #BYVAL2 specifies the second variable in the BY statement.

BY-variable-name

specifies a variable from the BY statement by its name. For example, #BYVAL(YEAR) specifies the BY variable, YEAR. *Variable-name* is not case sensitive.

#BYVAR*n***#BYVAR(*BY-variable-name*)**

substitutes the name of the BY-variable or the label associated with the variable (whatever the BY line would normally display) for #BYVAR in the text string.

Specify the variable with one of the following:

n

specifies a variable by its position in the BY statement. For example, #BYVAR2 specifies the second variable in the BY statement.

BY-variable-name

specifies a variable from the BY statement by its name. For example, #BYVAR(SITES) specifies the BY variable, SITES. *Variable-name* is not case sensitive.

See [“Substituting BY Line Values in a Text String” on page 2294](#)

[“Creating Accessible Tables” in *Creating Accessible SAS Viya Platform Output Using ODS and ODS Graphics*](#)

FORMAT_PRECEDENCE=PAGE | ROW | COLUMN

specifies whether the format that is specified for the page dimension (PAGE), row dimension (ROW), or column dimension (COLUMN) is applied to the contents of the table cells.

PAGE

specifies that the format that is specified for the page dimension is applied to the contents of the table cells.

ROW

specifies that the format that is specified for the row dimension is applied to the contents of the table cells.

COLUMN

specifies that the format that is specified for the column dimension is applied to the contents of the table cells.

Alias COL

Default COLUMN

FUZZ=*number*

supplies a numeric value against which table cell values other than frequency counts are compared to eliminate trivial values (absolute values less than the FUZZ= value) from printing. A number whose absolute value is less than the

FUZZ= value is treated as zero when printed. The default value is the smallest representable floating-point number on the computer that you are using.

INDENT=*number-of-spaces*

specifies the number of spaces to indent nested row headings, and suppresses the row headings for class variables.

Restriction In the HTML5, RTF, and Printer destinations, the INDENT= option suppresses the row headings for class variables but does not indent nested row headings.

Tip When there are no crossings in the row dimension, there is nothing to indent, so the value of *number-of-spaces* has no effect. However, in such cases INDENT= still suppresses the row headings for class variables.

MISSTEXT='text'

supplies up to 256 characters of text to be printed for table cells that contain missing values.

MISSTEXT={<label='text'> <STYLE=style-override(s)>}

supplies up to 256 characters of text to be printed and specifies a style override for table cells that contain missing values. For details about the arguments of the STYLE= option and how it is used, see [STYLE= on page 2258](#) in the TABLE statement.

Interaction A style element that is specified in a dimension expression overrides a style element that is specified in the MISSTEXT= option for any given cells.

Examples [“Providing Text for Cells That Contain Missing Values” on page 2302](#)
[“Example 13: Specifying Style Overrides for ODS Output” on page 2357](#)

NOCELLMERGE

specifies that data cells are not merged with other data cells in the table.

Note: The NOCELLMERGE option works with the ODS formatted destinations. These include the ODS MARKUP family, ODS RTF, and the ODS PRINTER family destinations.

Restriction The NOCELLMERGE option does not work with the traditional monospace output.

Interactions If you specify ROW=FLOAT or INDENT=0, PROC TABULATE produces single unmerged data rows. The NOCELLMERGE option is then ignored because there are no rows that need to be merged.

If the NOCELLMERGE option is in effect, the style of the empty data cells will be the default style of the data cell. The style of the empty data cells might not be the same style as the style of the formatted data cells.

Example [“Example 15: Using the NOCELLMERGE Option” on page 2366](#)

NOCONTINUED

suppresses the continuation message, **continued**, that is displayed at the bottom of tables that span multiple pages. The text is rendered with the AFTERCAPTION style element.

Note: Because HTML browsers do not break pages, NOCONTINUED has no effect on the HTML5 destination.

PRINTMISS

prints all values that occur for a class variable each time headings for that variable are printed, even if there are no data for some of the cells that these headings create. Consequently, PRINTMISS creates row and column headings that are the same for all logical pages of the table, within a single BY group.

Default If you omit the PRINTMISS option, then PROC TABULATE suppresses a row or column for which there are no data, unless you use the CLASSDATA= option in the PROC TABULATE statement.

Restriction If an entire logical page contains only missing values, then that page is not printed regardless of the PRINTMISS option.

See [“CLASSDATA=SAS-data-set” on page 2217](#)

Example [“Providing Headings for All Categories” on page 2301](#)

ROW=CONSTANT | FLOAT

specifies whether all title elements in a row crossing are allotted space even when they are blank.

CONSTANT

allots space to all row titles even if the title has been blanked out. (For example, N=' '.)

Alias CONST

FLOAT

divides the row title space equally among the nonblank row titles in the crossing.

Default CONSTANT

Example [“Example 7: Eliminating Row Headings” on page 2323](#)

RTSPACE=number

specifies the number of print positions to allot to all of the headings in the row dimension, including spaces that are used to print outlining characters for the row headings. PROC TABULATE divides this space equally among all levels of row headings.

Alias RTS=

Default one-fourth of the value of the SAS system option LINESIZE=

- Restriction** The RTSPACE= option affects only the traditional SAS monospace output destination.
- Interaction** By default, PROC TABULATE allots space to row titles that are blank. Use ROW=FLOAT in the TABLE statement to divide the space among only nonblank titles.
- See** For more examples of controlling the space for row titles, see *PROC TABULATE by Example, Second Edition*.
- Example** [“Example 1: Creating a Basic Two-Dimensional Table” on page 2306](#)

STYLE=style-override

specifies one or more style overrides to use for parts of the table other than table cells. For example, the following statement specifies that the background color for missing values is red and the background color the box is orange:

```
table (region all)*(division all),
      (prodtype all)*(actual*f=dollar10.) /
      misstext=[label='Missing' style=[background=red]]
      box=[label='Region by Division and Type' style=[backgroundcolor=orange]];
```

style-override

specifies one or more style attributes or style elements to override the default style element and attributes in a specific area of a report.

You can specify a style override in three ways:

- Specify a style element. A style element is a collection of style attributes that apply to a particular part of the output for a SAS program.
- Specify a style attribute. A style attribute is a name-value pair that describes a single behavioral or visual aspect of a piece of output. This is the most specific method of changing the appearance of your output.
- Specify the PARENT value. The PARENT value specifies that the data cell use the style element of its parent heading.

style-override has the following form:

PARENT | *style-element-name* | [*style-attribute-name-1*=*style-attribute-value-1*

<*style-attribute-name-2*=*style-attribute-value-2* ...>]

<PARENT>

specifies that the data cell use the style element of its parent heading.

The parent style element of a data cell is one of the following:

- the style element of the leaf heading above the column that contains the data cell, if the table specifies no row dimension, or if the table specifies the style element in the column dimension expression
- the style element of the leaf heading above the row that contains the cell, if the table specifies the style element in the row dimension expression
- the Beforecaption style element, if the table specifies the style element in the page dimension expression

- undefined, otherwise

Note: In this usage, the angle brackets around the word PARENT are required. Braces or square brackets cannot be substituted in the syntax.

Note: The parent of a heading (not applicable to STYLE= in the PROC TABULATE statement) is the heading under which the current heading is nested.

style-attribute-name

specifies the attribute to change.

See For information about using styles with PROC TABULATE, see [“Using ODS Styles with PROC TABULATE” on page 2278](#).

For a table of default style attributes and style elements for each ODS destination, see [“Style Elements and Style Attributes for Table Regions” on page 2286](#).

style-attribute-value

specifies a value for the attribute. Each attribute has a different set of valid values. A SAS format can also be used as an attribute value for conditional formatting.

See For information about using styles with PROC TABULATE, see [“Using ODS Styles with PROC TABULATE” on page 2278](#).

For information about using SAS formats as style attribute values, see [“Using a Format to Assign a Style Attribute” on page 2290](#).

For a table of default style attributes and style elements for each ODS destination, see [“Style Elements and Style Attributes for Table Regions” on page 2286](#).

style-element-name

is the name of a style element that is part of an ODS style template.

See For information about using styles with PROC TABULATE, see [“Using ODS Styles with PROC TABULATE” on page 2278](#).

For a table of default style attributes and style elements for each ODS destination, see [“Style Elements and Style Attributes for Table Regions” on page 2286](#).

Alias S=

Tips To override a style element specification that is made as an option in the TABLE statement, specify STYLE= in a dimension expression of the TABLE statement.

You can use braces ({ and }) instead of square brackets ([and]).

See For information about using styles with PROC TABULATE, see [“Using ODS Styles with PROC TABULATE” on page 2278](#).

Example [“Example 13: Specifying Style Overrides for ODS Output” on page 2357](#)

STYLE_PRECEDENCE=PAGE | ROW | COLUMN

specifies whether the style that is specified for the page dimension (PAGE), row dimension (ROW), or column dimension (COLUMN or COL) is applied to the contents of the table cells.

PAGE

specifies that the style that is specified for the page dimension is applied to the contents of the table cells.

ROW

specifies that the style that is specified for the row dimension is applied to the contents of the table cells.

COLUMN

specifies that the style that is specified for the column dimension is applied to the contents of the table cells.

Alias COL

Default COLUMN

Example [“Example 14: Style Precedence” on page 2362](#)

Details

What Are Dimension Expressions?

A dimension expression defines the content and appearance of a dimension (the columns, rows, or pages in the table) by specifying the combination of variables, variable values, and statistics that make up that dimension. A TABLE statement consists of from one to three dimension expressions separated by commas. Options can follow the dimension expressions.

If all three dimensions are specified, then the leftmost dimension expression defines pages, the middle dimension expression defines rows, and the rightmost dimension expression defines columns. If two dimensions are specified, then the left dimension expression defines rows, and the right dimension expression defines columns. If a single dimension is specified, then the dimension expression defines columns.

A dimension expression consists of one or more elements and operators.

Elements That You Can Use in a Dimension Expression

analysis variables

(See the [VAR statement on page 2264](#).)

class variables

(See the [CLASS statement on page 2234](#).)

the universal class variable ALL

summarizes all of the categories for class variables in the same parenthetical group or dimension (if the variable ALL is not contained in a parenthetical group).

Note: If the input data set contains a variable named ALL, then enclose the name of the universal class variable in quotation marks.

Examples [“Example 6: Summarizing Information with the Universal Class Variable ALL” on page 2320](#)

[“Example 8: Creating Multipage Tables” on page 2325](#)

[“Example 12: Using Denominator Definitions to Display Basic Frequency Counts and Percentages” on page 2343](#)

keywords for statistics

See [“Statistics That Are Available in PROC TABULATE” on page 2271](#) for a list of available statistics. Use the asterisk (*) operator to associate a statistic keyword with a variable. The N statistic (number of nonmissing values) can be specified in a dimension expression without associating it with a variable.

Default For analysis variables, the default statistic is SUM. Otherwise, the default statistic is N.

Restriction Statistic keywords other than N must be associated with an analysis variable.

Interaction Statistical keywords should be enclosed by single or double quotation marks to ensure that the keyword element is treated as a statistical keyword and not treated as a variable. By default, SAS treats these keywords as variables.

Example

```
n
Region*n
Sales*max
```

Examples [“Example 9: Reporting on Multiple-Response Survey Data” on page 2328](#)

[“Example 12: Using Denominator Definitions to Display Basic Frequency Counts and Percentages” on page 2343](#)

format modifiers

define how to format values in cells. Use the asterisk (*) operator to associate a format modifier with the element (an analysis variable or a statistic) that produces the cells that you want to format. Format modifiers have the form

f=format

Tip Format modifiers have no effect on CLASS variables.

See For more information about specifying formats in tables, see [“Formatting Values in Tables” on page 2274](#).

Example `Sales*f=dollar8.2`

Example [“Example 6: Summarizing Information with the Universal Class Variable ALL” on page 2320](#)

labels

temporarily replace the names of variables and statistics. Labels affect only the variable or statistic that immediately precedes the label. Labels have the form

statistic-keyword-or-variable-name='label-text'

Tip PROC TABULATE eliminates the space for blank column headings from a table but by default does not eliminate the space for blank row headings unless all row headings are blank. Use ROW=FLOAT in the TABLE statement to remove the space for blank row headings.

Example `Region='Geographical Region'`
`Sales*max='Largest Sale'`

Examples [“Example 5: Customizing Row and Column Headings” on page 2318](#)
[“Example 7: Eliminating Row Headings” on page 2323](#)

style specifications

specify style elements and style attributes for page dimension text, headings, or data cells. For details, see [“Specifying Style Attributes and Style Elements in Dimension Expressions” on page 2263](#).

Operators That You Can Use in a Dimension Expression

asterisk *

creates categories from the combination of values of the class variables and constructs the appropriate headings for the dimension. If one of the elements is an analysis variable, then the statistics for the analysis variable are calculated for the categories that are created by the class variables. This process is called crossing.

Example `Region*Division`
`Quarter*Sales*f=dollar8.2`

Example [“Example 1: Creating a Basic Two-Dimensional Table” on page 2306](#)

(blank)

places the output for each element immediately after the output for the preceding element. This process is called concatenation.

Example `n Region*Sales ALL`

Example [“Example 6: Summarizing Information with the Universal Class Variable ALL” on page 2320](#)

parentheses ()

group elements and associate an operator with each concatenated element in the group.

Example `Division*(Sales*max Sales*min)
(Region ALL)*Sales`

Example [“Example 6: Summarizing Information with the Universal Class Variable ALL” on page 2320](#)

angle brackets <>

specify denominator definitions, which determine the value of the denominator in the calculation of a percentage. For a discussion of how to construct denominator definitions, see [“Calculating Percentages ” on page 2275](#).

Examples [“Example 9: Reporting on Multiple-Response Survey Data” on page 2328](#)

[“Example 12: Using Denominator Definitions to Display Basic Frequency Counts and Percentages” on page 2343](#)

Specifying Style Attributes and Style Elements in Dimension Expressions

You can specify one or more style elements or style attributes in a dimension expression to control the appearance of non-LISTING output. You can modify the appearance of the following areas:

- analysis variable name headings
- class variable name headings
- class variable level value headings
- data cells
- keyword headings
- page dimension text

Specifying a style attribute or style element in a dimension expression is useful when you want to override a style attribute or style element that you have specified in another statement, such as the PROC TABULATE, CLASS, CLASSLEV, KEYWORD, TABLE, or VAR statements.

The syntax for specifying style elements and style attributes in a dimension expression is

`[STYLE<(CLASSLEV)>=<style-element-name | PARENT>`

`[style-attribute-name-1=style-attribute-value-1< style-attribute-name-2=style-attribute-value-2 ...>]]`

These are some examples of style attributes in dimension expressions:

■

```
dept={label='Department '
      style=[color=red] }, N
```

- `dept*[style=MyDataStyle], N`
- `dept*[format=12.2 style=MyDataStyle], N`

Note: When used in a dimension expression, the `STYLE=` option must be enclosed within square brackets ([and]) or braces ({ and }).

(CLASSLEV)

assigns a style element to a class variable level value heading. For example, the following TABLE statement specifies that the level value heading for the class variable, DEPT, has a foreground color of yellow:

```
table dept=[style(classlev)=
             [color=yellow]]*sales;
```

Note: The CLASSLEV option is used only in dimension expressions.

For an example that shows how to specify style elements within dimension expressions, see [“Example 13: Specifying Style Overrides for ODS Output” on page 2357](#). For information about using styles with PROC TABULATE, see [“Using ODS Styles with PROC TABULATE” on page 2278](#).

VAR Statement

Identifies numeric variables to use as analysis variables.

Aliases: VARIABLE
 VARIABLES

Tip: You can use multiple VAR statements.

Example: [“Example 13: Specifying Style Overrides for ODS Output” on page 2357](#)

Syntax

VAR *analysis-variable(s)* </ options>;

Required Argument

analysis-variable(s);

identifies the analysis variables in the table. Analysis variables are numeric variables for which PROC TABULATE calculates statistics. The values of an analysis variable can be continuous or discrete.

If an observation contains a missing value for an analysis variable, then PROC TABULATE omits that value from calculations of all statistics except N (the number of observations with nonmissing variable values) and NMISS (the number of observations with missing variable values). For example, the missing

value does not increase the SUM, and it is not counted when you are calculating statistics such as the MEAN.

Interaction If a variable name and a statistic name are the same, enclose the statistic name in single or double quotation marks.

Options to Use after a Slash Delimiter

STYLE=style-override(s)

specifies one or more style overrides for analysis variable name headings.

For example, the following statement specifies that the background color for analysis variable name headings is tan:

```
var actual / style=[background=tan];
```

style-override

specifies one or more style attributes or style elements to override the default style element and attributes in a specific area of a report.

You can specify a style override in three ways:

- Specify a style element. A style element is a collection of style attributes that apply to a particular part of the output for a SAS program.
- Specify a style attribute. A style attribute is a name-value pair that describes a single behavioral or visual aspect of a piece of output. This is the most specific method of changing the appearance of your output.
- Specify the PARENT value. The PARENT value specifies that the data cell use the style element of its parent heading.

style-override has the following form:

PARENT | *style-element-name* | [*style-attribute-name-1*=*style-attribute-value-1*

<style-attribute-name-2=style-attribute-value-2 ...>]

<PARENT>

specifies that the data cell use the style element of its parent heading.

The parent style element of a data cell is one of the following:

- the style element of the leaf heading above the column that contains the data cell, if the table specifies no row dimension, or if the table specifies the style element in the column dimension expression
- the style element of the leaf heading above the row that contains the cell, if the table specifies the style element in the row dimension expression
- the Beforecaption style element, if the table specifies the style element in the page dimension expression
- undefined, otherwise

Note: In this usage, the angle brackets around the word PARENT are required. Braces or square brackets cannot be substituted in the syntax.

Note: The parent of a heading (not applicable to STYLE= in the PROC TABULATE statement) is the heading under which the current heading is nested.

style-attribute-name

specifies the attribute to change.

See For information about using styles with PROC TABULATE, see [“Using ODS Styles with PROC TABULATE” on page 2278](#).

For a table of default style attributes and style elements for each ODS destination, see [“Style Elements and Style Attributes for Table Regions” on page 2286](#).

style-attribute-value

specifies a value for the attribute. Each attribute has a different set of valid values. A SAS format can also be used as an attribute value for conditional formatting.

See For information about using styles with PROC TABULATE, see [“Using ODS Styles with PROC TABULATE” on page 2278](#).

For information about using SAS formats as style attribute values, see [“Using a Format to Assign a Style Attribute” on page 2290](#).

For a table of default style attributes and style elements for each ODS destination, see [“Style Elements and Style Attributes for Table Regions” on page 2286](#).

style-element-name

is the name of a style element that is part of an ODS style template.

See For information about using styles with PROC TABULATE, see [“Using ODS Styles with PROC TABULATE” on page 2278](#).

For a table of default style attributes and style elements for each ODS destination, see [“Style Elements and Style Attributes for Table Regions” on page 2286](#).

Alias S=

Tips To override a style element that is specified in the VAR statement, you can specify a style element in the related TABLE statement dimension expression.

The use of STYLE= in the VAR statement differs slightly from its use in the PROC TABULATE statement. In the VAR statement, inheritance is different for rows and columns. For rows, the parent heading is located to the left of the current heading. For columns, the parent heading is located above the current heading.

See For information about using styles with PROC TABULATE, see [“Using ODS Styles with PROC TABULATE” on page 2278](#).

Example [“Example 13: Specifying Style Overrides for ODS Output” on page 2357](#)

WEIGHT=weight-variable

specifies a numeric variable whose values weight the values of the variables that are specified in the VAR statement. The variable does not have to be an integer.

PROC TABULATE responds to weight values in accordance with the following table.

Table 61.3 PROC TABULATE Response to Weight Values

Weight Value	PROC TABULATE Response
0	Counts the observation in the total number of observations
Less than 0	Converts the value to zero and counts the observation in the total number of observations
Missing	Excludes the observation

To exclude observations that contain negative and zero weights from the analysis, use EXCLNPWGT. Note that most SAS/STAT procedures, such as PROC GLM, exclude negative and zero weights by default.

Restriction To compute weighted quantiles, use QMETHOD=OS in the PROC statement.

Note Prior to Version 7 of SAS, the procedure did not exclude the observations with missing weights from the count of observations.

Tips When you use the WEIGHT= option, consider which value of the VARDEF= option is appropriate. See the discussion of [“VARDEF=DF | N | WDF | WEIGHT” on page 2228](#).

Use the WEIGHT option in multiple VAR statements to specify different weights for the analysis variables.

WEIGHT Statement

Specifies weights for analysis variables in the statistical calculations.

Alias: WGT

See: For information about calculating weighted statistics and for an example that uses the WEIGHT statement, see [“Calculating Weighted Statistics” on page 2198](#).

Syntax

WEIGHT *variable*;

Required Argument

variable

specifies a numeric variable whose values weight the values of the analysis variables. The values of the variable do not have to be integers.

PROC TABULATE responds to weight values in accordance with the following table.

Table 61.4 PROC TABULATE Response to Weight Values

Weight Value	PROC TABULATE Response
0	Counts the observation in the total number of observations
Less than 0	Converts the value to zero and counts the observation in the total number of observations
Missing	Excludes the observation

To exclude observations that contain negative and zero weights from the analysis, use EXCLNPWGT. Note that most SAS/STAT procedures, such as PROC GLM, exclude negative and zero weights by default.

Note: Prior to Version 7 of SAS, the procedure did not exclude the observations with missing weights from the count of observations.

- Restrictions

To compute weighted quantiles, use QMETHOD=OS in the PROC statement.

PROC TABULATE will not compute MODE when a weight variable is active. Instead, try using PROC UNIVARIATE when MODE needs to be computed and a weight variable is active.
- Interaction

If you use the WEIGHT= option in a VAR statement to specify a weight variable, then PROC TABULATE uses this variable instead to weight those VAR statement variables.
- Tip

When you use the WEIGHT statement, consider which value of the VARDEF= option is appropriate. See the discussion of [“VARDEF=DF | N | WDF | WEIGHT”](#) on page 2228 and the calculation of weighted statistics in the [“Keywords and Formulas”](#) on page 2426 section of this document.

WHERE Statement

Subsets the input data set by specifying certain conditions that each observation must meet before it is available for processing.

Syntax

WHERE *where-expression-1*
< *logical-operator where-expression-n*>;

Arguments

where-expression

is an arithmetic or logical expression that generally consists of a sequence of operands and operators.

Tips The operands and operators described in the next several sections are also valid for the WHERE= data set option.

You can specify multiple where-expressions.

logical-operator

can be AND, AND NOT, OR, or OR NOT.

Example

Procedures That Support the WHERE Statement

You can use the WHERE statement with any of the following Base SAS procedures that read a SAS data set:

COMPARE	REPORT
CORR	SORT
DATASETS (APPEND statement)	SQL
FREQ	STANDARD

MEANS or SUMMARY	TABULATE
PRINT	TRANSPOSE
RANK	UNIVARIATE

Details

- The COMPARE procedure and the APPEND statement in PROC DATASETS accept more than one input data set. For more information, see the documentation for the specific procedure.
- To subset the output data set, use the WHERE= data set option:

```
proc report data=debate nowd  
             out=onlyfr(where=(year='1'));  
run;
```

For more information about WHERE=, see [SAS DATA Step Statements: Reference](#).

Temporarily Dissociating a Format from a Variable

In this example, PROC PRINT prints only those observations that meet the condition of the WHERE expression.

```
options nodate pageno=1 linesize=64  
       pagesize=40;  
  
proc print data=debate noobs;  
  where gpa>3.5;  
  title 'Team Members with a GPA';  
  title2 'Greater than 3.5';  
run;
```

Output

**Team Members with a GPA
Greater than 3.5**

Name	Gender	Year	GPA
Capiccio	m	Freshman	3.598
Tucker	m	Freshman	3.901
Bagwell	f	Sophomore	3.722
Gold	f	Junior	3.609
Syme	f	Junior	3.883
Baglione	f	Senior	4.000
Carr	m	Senior	3.750
Hall	m	Senior	3.574

Usage: TABULATE Procedure

Statistics That Are Available in PROC TABULATE

Use the following keywords to request statistics in the TABLE statement or to specify statistic keywords in the KEYWORD or KEYLABEL statement.

Note: If a variable name (class or analysis) and a statistic name are the same, then enclose the statistic name in single quotation marks (for example, 'MAX').

Descriptive statistic keywords

COLPCTN	PCTSUM
COLPCTSUM	RANGE
CSS	REPPCTN
CV	REPPCTSUM

KURTOSIS KURT	ROWPCTN
LCLM	ROWPCTSUM
MAX	SKEWNESS SKEW
MEAN	STDDEV STD
MIN	STDERR
MODE	SUM
N	SUMWGT
NMISS	UCLM
PAGEPCTN	USS
PAGEPCTSUM	VAR
PCTN	
Quantile statistic keywords	
MEDIAN P50	Q3 P75
P1	P70
P5	P80
P10	P90
P20	P95
P30	P99
P40	
P60	
Q1 P25	QRANGE
Hypothesis testing keywords	
PROBT PRT	T

These statistics, the formulas that are used to calculate them, and their data requirements are discussed in the [“Keywords and Formulas” on page 2426](#) section of this document.

To compute standard error of the mean (STDERR) or Student’s *t*-test, you must use the default value of the VARDEF= option, which is DF. The VARDEF= option is specified in the PROC TABULATE statement.

To compute weighted quantiles, you must use QMETHOD=OS in the PROC TABULATE statement.

Use both LCLM and UCLM to compute a two-sided confidence limit for the mean. Use only LCLM or UCLM to compute a one-sided confidence limit. Use the ALPHA= option in the PROC TABULATE statement to specify a confidence level.

Formatting Class Variables

Use the FORMAT statement to assign a format to a class variable for the duration of a PROC TABULATE step. When you assign a format to a class variable, PROC TABULATE uses the formatted values to create categories, and it uses the formatted values in headings. If you do not specify a format for a class variable, and the variable does not have any other format assigned to it, then the default format, BEST12., is used, unless the GROUPINTERNAL option is specified.

User-defined formats are particularly useful for grouping values into fewer categories. For example, if you have a class variable, Age, with values that range from 1 to 99, then you could create a user-defined format that groups the ages so that your tables contain a manageable number of categories. The following PROC FORMAT step creates a format that condenses all possible values of age into six groups of values.

```
proc format;
  value agefmt 0-29='Under 30'
              30-39='30-39'
              40-49='40-49'
              50-59='50-59'
              60-69='60-69'
              other='70 or over';
run;
```

For information about creating user-defined formats, see [Chapter 26, “FORMAT Procedure,” on page 863](#).

By default, PROC TABULATE includes in a table only those formats for which the frequency count is not zero and for which values are not missing. To include missing values for all class variables in the output, use the MISSING option in the PROC TABULATE statement, and to include missing values for selected class variables, use the MISSING option in a CLASS statement. To include formats for which the frequency count is zero, use the PRELOADFMT option in a CLASS statement and the PRINTMISS option in the TABLE statement, or use the CLASSDATA= option in the PROC TABULATE statement.

Formatting Values in Tables

The formats for data in table cells serve two purposes. They determine how PROC TABULATE displays the values, and they determine the width of the columns. The default format for values in table cells is 12.2. You can modify the format for printing values in table cells by doing the following:

- changing the default format with the FORMAT= option in the PROC TABULATE statement
- crossing elements in the TABLE statement with the F= format modifier

Note: You cannot modify the format for printing values in table cells by using the FORMAT or the ATTRIB statement. If you use these statements, the analysis variable formats that they contain will be ignored.

PROC TABULATE determines the format to use for a particular cell from the following default order of precedence for formats:

- 1 If no other formats are specified, then PROC TABULATE uses the default format (12.2).
- 2 The FORMAT= option in the PROC TABULATE statement changes the default format. If no format modifiers affect a cell, then PROC TABULATE uses this format for the value in that cell.
- 3 A format modifier in the page dimension applies to the values in all the table cells on the logical page unless you specify another format modifier for a cell in the row or column dimension.
- 4 A format modifier in the row dimension applies to the values in all the table cells in the row unless you specify another format modifier for a cell in the column dimension.
- 5 A format modifier in the column dimension applies to the values in all the table cells in the column.

You can change this order of precedence by using the FORMAT_PRECEDENCE= option in the TABLE statement. For more information, see [“TABLE Statement” on page 2250](#). For example, if you specify FORMAT_PRECEDENCE=ROW and specify a format modifier in the row dimension, then that format overrides all other specified formats for the table cells.

Calculating Percentages

Calculating the Percentage of the Value in a Single Table Cell

The following statistics print the percentage of the value in a single table cell in relation to the total of the values in a group of cells. No denominator definitions are required. However, an analysis variable can be used as a denominator definition for percentage sum statistics.

- REPPCTN and REPPCTSUM statistics—print the percentage of the value in a single table cell in relation to the total of the values in the report.
- COLPCTN and COLPCTSUM statistics—print the percentage of the value in a single table cell in relation to the total of the values in the column.
- ROWPCTN and ROWPCTSUM statistics—print the percentage of the value in a single table cell in relation to the total of the values in the row.
- PAGEPCTN and PAGEPCTSUM statistics—print the percentage of the value in a single table cell in relation to the total of the values in the page.

These statistics calculate the most commonly used percentages. See [“Example 11: Calculating Various Percentage Statistics” on page 2339](#) for an example.

Using PCTN and PCTSUM

PCTN and PCTSUM statistics can be used to calculate these same percentages. They enable you to manually define denominators. PCTN and PCTSUM statistics print the percentage of the value in a single table cell in relation to the value (used in the denominator of the calculation of the percentage) in another table cell or to the total of the values in a group of cells. By default, PROC TABULATE summarizes the values in all N cells (for PCTN) or all SUM cells (for PCTSUM) and uses the summarized value for the denominator. You can control the value that PROC TABULATE uses for the denominator with a denominator definition.

You place a denominator definition in angle brackets (< and >) next to the PCTN or PCTSUM statistic. The denominator definition specifies which categories to sum for the denominator.

This section illustrates how to specify denominator definitions in a simple table. [“Example 12: Using Denominator Definitions to Display Basic Frequency Counts and Percentages” on page 2343](#) illustrates how to specify denominator definitions in a table that consists of multiple subtables. For more examples of denominator definitions, see *PROC TABULATE by Example, Second Edition*.

Specifying a Denominator for the PCTN Statistic

The following PROC TABULATE step calculates the N statistic and three different versions of PCTN using the data set “ENERGY” on page 2512.

```
proc tabulate data=energy;
  class division type;
  table division*
    (n='Number of customers'
     pctn<type>='% of row' 1
     pctn<division>='% of column' 2
     pctn='% of all customers'), 3
    type/rts=50;
  title 'Number of Users in Each Division';
run;
```

The TABLE statement creates a row for each value of Division and a column for each value of Type. Within each row, the TABLE statement nests four statistics: N and three different calculations of PCTN. (See the following figure.) Each occurrence of PCTN uses a different denominator definition.

Figure 61.4 Three Different Uses of the PCTN Statistic with Frequency Counts Highlighted

Number of Users in Each Division			
		Type	
		1	2
Division			
1	Number of customers	6.00	6.00
	% of row 1	50.00	50.00
	% of column 2	27.27	27.27
	% of all customers 3	13.64	13.64
2	Number of customers	3.00	3.00
	% of row	50.00	50.00
	% of column	13.64	13.64
	% of all customers	6.82	6.82
3	Number of customers	8.00	8.00
	% of row	50.00	50.00
	% of column	36.36	36.36
	% of all customers	18.18	18.18
4	Number of customers	5.00	5.00
	% of row	50.00	50.00
	% of column	22.73	22.73
	% of all customers	11.36	11.36

- 1 <type> sums the frequency counts for all occurrences of Type within the same value of Division. Thus, for Division=1, the denominator is 6 + 6, or 12.
- 2 <division> sums the frequency counts for all occurrences of Division within the same value of Type. Thus, for Type=1, the denominator is 6 + 3 + 8 + 5, or 22.
- 3 The third use of PCTN has no denominator definition. Omitting a denominator definition is the same as including all class variables in the denominator

definition. Thus, for all cells, the denominator is $6 + 3 + 8 + 5 + 6 + 3 + 8 + 5$, or 44.

Specifying a Denominator for the PCTSUM Statistic

The following PROC TABULATE step sums expenditures for each combination of Type and Division and calculates three different versions of PCTSUM.

```
proc tabulate data=energy format=8.2;
  class division type;
  var expenditures;
  table division*
    (sum='Expenditures'*f=dollar10.2
     pctsum<type>='% of row' 1
     pctsum<division>='% of column' 2
     pctsum='% of all customers'), 3
    type*expenditures/rtts=40;
  title 'Expenditures in Each Division';
run;
```

The TABLE statement creates a row for each value of Division and a column for each value of Type. Because Type is crossed with Expenditures, the value in each cell is the sum of the values of Expenditures for all observations that contribute to the cell. Within each row, the TABLE statement nests four statistics: SUM and three different calculations of PCTSUM. (See the following figure.) Each occurrence of PCTSUM uses a different denominator definition.

Figure 61.5 Three Different Uses of the PCTSUM Statistic with Sums Highlighted

Expenditures in Each Division

		Type	
		1	2
		Expenditures	Expenditures
Division			
1	Expenditures	\$7,477.00	\$5,129.00
	% of row 1	59.31	40.69
	% of column 2	16.15	13.66
	% of all customers 3	8.92	6.12
2	Expenditures	\$19,379.00	\$15,078.00
	% of row	56.24	43.76
	% of column	41.86	40.15
	% of all customers	23.11	17.98
3	Expenditures	\$5,476.00	\$4,729.00
	% of row	53.66	46.34
	% of column	11.83	12.59
	% of all customers	6.53	5.64
4	Expenditures	\$13,959.00	\$12,619.00
	% of row	52.52	47.48
	% of column	30.15	33.60
	% of all customers	16.65	15.05

- 1 <type> sums the values of Expenditures for all occurrences of Type within the same value of Division. Thus, for Division=1, the denominator is \$7,477 + \$5,129.
- 2 <division> sums the frequency counts for all occurrences of Division within the same value of Type. Thus, for Type=1, the denominator is \$7,477 + \$19,379 + \$5,476 + \$13,959.
- 3 The third use of PCTN has no denominator definition. Omitting a denominator definition is the same as including all class variables in the denominator definition. Thus, for all cells, the denominator is \$7,477 + \$19,379 + \$5,476 + \$13,959 + \$5,129 + \$15,078 + \$4,729 + \$12,619.

Using ODS Styles with PROC TABULATE

Using Styles with Base SAS Procedures

Most Base SAS procedures that support ODS use one or more table templates to produce output objects. These table templates include templates for table

elements: columns, headers, and footers. Each table element can specify the use of one or more style elements for various parts of the output. These style elements cannot be specified within the syntax of the procedure, but you can use customized styles for the ODS destinations that you use. For more information about customizing tables and styles, see [“TEMPLATE Procedure: Creating a Style Template” in SAS Output Delivery System: Procedures Guide](#).

The Base SAS reporting procedures, PROC PRINT, PROC REPORT, and PROC TABULATE, enable you to quickly analyze your data and organize it into easy-to-read tables. You can use the STYLE= option with these procedure statements to modify the appearance of your report. The STYLE= option enables you to make changes in sections of output without changing the default style for all of the output. You can customize specific sections of procedure output by specifying the STYLE= option in specific statements within the procedure.

The following program uses the STYLE= option to create the colors in the PROC TABULATE output below:

```
ods _all_ close;
ods html5 file="file-path/customStyle.html";
proc sort data=sashelp.prdsale out=prdsale;
    by Country;
run;

proc tabulate data=prdsale;
    class region division prodtype / style=[background=lightgreen];
    classlev region division prodtype / style=[background=yellow];
    var actual / style=[background=tan];
    keyword all sum / style=[background=linen color=blue];
    keylabel all='Total';
    table (region all)*(division all),
          (prodtype all)*(actual*f=dollar10.) /
          box=[label='Region by Division and Type'
style=[backgroundcolor=orange]];

title 'Actual Product Sales';
title2 '(millions of dollars)';
run;
ods _all_ close;
```

Output 61.4 Enhanced PROC TABULATE Output

Actual Product Sales (millions of dollars)				
Region by Division and Type		Product type		Total
		FURNITURE	OFFICE	
		Actual Sales	Actual Sales	Actual Sales
		Sum	Sum	Sum
Region	Division			
EAST	CONSUMER	\$72,570	\$108,686	\$181,256
	EDUCATION	\$73,901	\$115,104	\$189,005
	Total	\$146,471	\$223,790	\$370,261
WEST	Division			
	CONSUMER	\$76,209	\$105,020	\$181,229
	EDUCATION	\$67,945	\$110,902	\$178,847
	Total	\$144,154	\$215,922	\$360,076
Total	Division			
	CONSUMER	\$148,779	\$213,706	\$362,485
	EDUCATION	\$141,846	\$226,006	\$367,852
	Total	\$290,625	\$439,712	\$730,337

Styles, Style Elements, and Style Attributes

Understanding Styles, Style Elements, and Style Attributes

The appearance of SAS output is controlled by ODS style templates (ODS styles). ODS styles are produced from compiled STYLE templates written in PROC TEMPLATE style syntax. An ODS style template is a collection of style elements that provides specific visual attributes for your SAS output.

- A style element is a named collection of style attributes that apply to a particular part of the output. Each area of ODS output has a style element name that is associated with it. The style element name specifies where the style attributes are applied. For example, a style element might contain instructions for the presentation of column headings or for the presentation of the data inside the cells. Style elements might also specify default colors and fonts for output that uses the style.
- A style attribute is a visual property, such as color, font properties, and line characteristics, that is defined in ODS with a reserved name and value. Style attributes are collectively referenced by a style element within a style template.

Each *style attribute* specifies a value for one aspect of the presentation. For example, the BACKGROUND_COLOR= attribute specifies the color for the background of an HTML table or for a colored table in printed output. The FONTSTYLE= attribute specifies whether to use a Roman font or an italic font.

Note: Because styles control the presentation of the data, they have no effect on output objects that go to the DOCUMENT or OUTPUT destination.

Available styles are in the SASHELP.TMPLMST item store. You can display the list of available style templates by using the [LIST](#) statement in PROC TEMPLATE:

```
proc template;
  list styles / store=sashelp.tmplmst;
run;
```

For complete information about viewing ODS styles, see “[Viewing ODS Styles Supplied by SAS](#)” in *SAS Output Delivery System: Procedures Guide*.

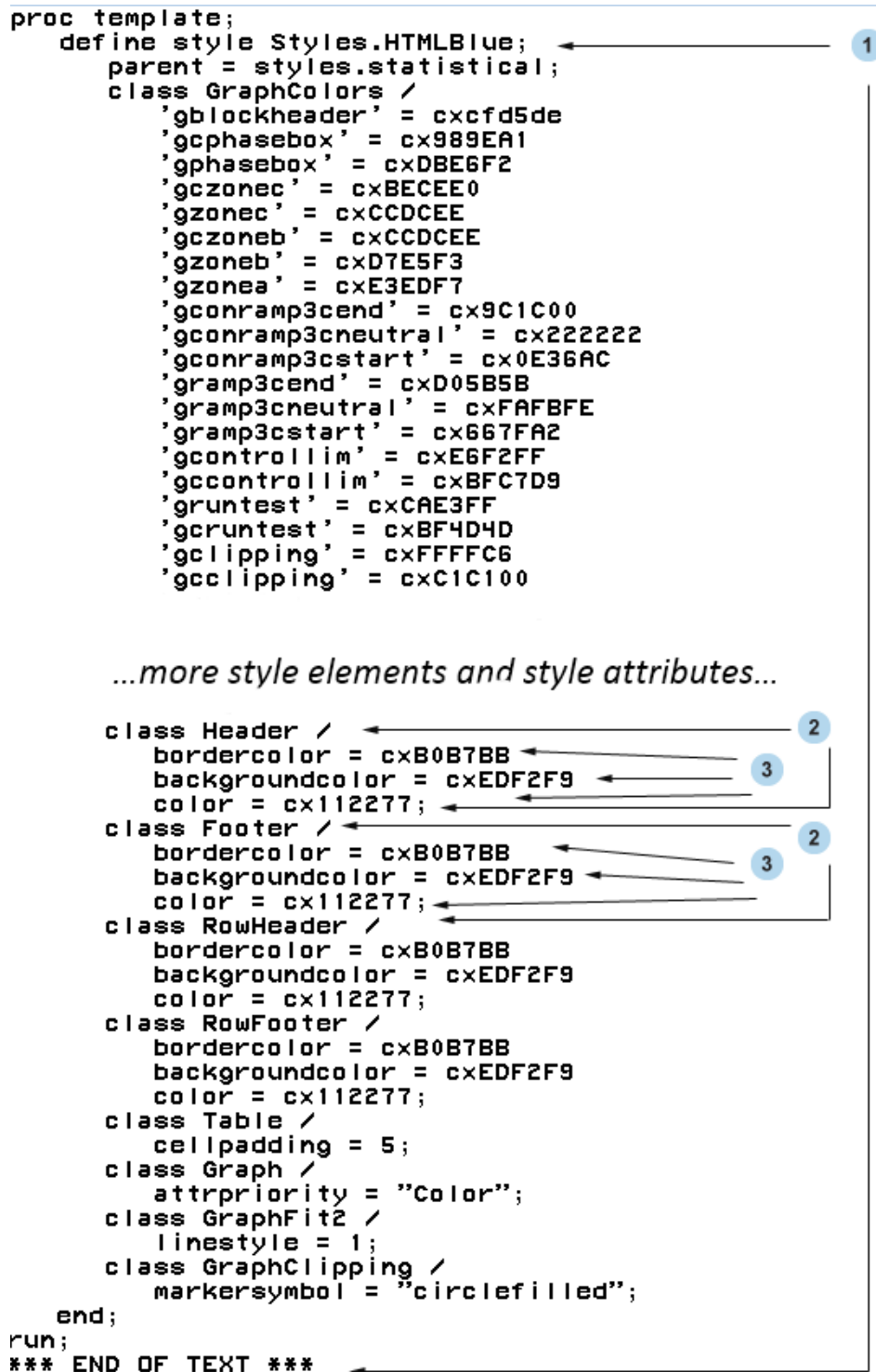
HTML5 output uses the HTML_Enclsr style template. To help you become familiar with styles, style elements, and style attributes, look at the relationship between them.

You can use the [SOURCE](#) statement in PROC TEMPLATE to display the structure of a style template. The following code prints the structure of the HTMLBlue style template to the SAS log:

```
proc template;
  source styles.HTMLBlue;
run;
```

The following figure illustrates the structure of a style. The figure shows the relationship between the style, the style elements, and the style attributes.

Figure 61.6 Diagram of the HtmlBlue Style



The following list corresponds to the numbered items in the preceding figure:

- 1 Styles.HtmlBlue is the *style*. Styles describe how to display presentation aspects (color, font, font size, and so on) of the SAS output. A style determines the overall appearance of the ODS documents that use it. The default style for HTML output is HtmlBlue. Each style consists of style elements.

You can create new styles with the “[DEFINE STYLE Statement](#)” in *SAS Output Delivery System: Procedures Guide*. New styles can be created independently or from an existing style. You can use “[PARENT= Statement](#)” in *SAS Output Delivery System: Procedures Guide* to create a new style from an existing style. For complete documentation about ODS styles, see “[Style Templates](#)” in *SAS Output Delivery System: Procedures Guide*.

- 2 Header and Footer are examples of *style elements*. A style element is a collection of style attributes that apply to a particular part of the output for a SAS program. For example, a style element might contain instructions for the presentation of column headings or for the presentation of the data inside table cells. Style elements might also specify default colors and fonts for output that uses the style. Style elements exist inside styles and consist of one or more style attributes. Style elements can be user-defined or supplied by SAS. User-defined style elements can be created by the “[STYLE Statement](#)” in *SAS Output Delivery System: Procedures Guide*.

Note: For a list of the default style elements used for HTML and markup languages and their inheritance, see “[Style Elements](#)” in *SAS Output Delivery System: Procedures Guide*.

- 3 BORDERCOLOR=, BACKGROUNDColor=, and COLOR= are examples of *style attributes*. Style attributes specify a value for one aspect of the area of the output that its style element applies to. For example, the COLOR= attribute specifies the value cxx112277 for the font color. For a list of style attributes supplied by SAS, see “[Style Attributes](#)” in *SAS Output Delivery System: Procedures Guide*.

Style attributes can be referenced with style references. See “[style-reference](#)” in *SAS Output Delivery System: Procedures Guide* for more information about style references.

The following table shows commonly used style attributes that you can set with the STYLE= option in PROC PRINT, PROC TABULATE, and PROC REPORT. Most of these attributes apply to parts of the table other than cells (for example, table borders and the lines between columns and rows). Note that not all attributes are valid in all destinations. For more information about these style attributes, their valid values, and their applicable destinations, see “[Style Attributes Tables](#)” in *SAS Output Delivery System: Procedures Guide*.

Table 61.5 Style Attributes for PROC REPORT, PROC TABULATE, and PROC PRINT

Attribute	PROC REPORT STATEMENT REPORT Area	PROC REPORT Areas: CALLDEF, COLUMN, HEADER, LINES, SUMMARY	PROC TABULATE STATEMENT TABLE	PROC TABULATE STATEMENTS VAR, CLASS, BOX, CLASSLEV, KEYWORD	PROC PRINT TABLE location	PROC PRINT: all locations other than TABLE
ASIS=	X	X		X		X

Attribute	PROC REPORT STATEMENT REPORT Area	PROC REPORT Areas: CALLDEF, COLUMN, HEADER, LINES, SUMMARY	PROC TABULATE STATEMENT TABLE	PROC TABULATE STATEMENTS VAR, CLASS, BOX, CLASSLEV, KEYWORD	PROC PRINT TABLE location	PROC PRINT: all locations other than TABLE
BACKGROUNDCOLOR=	X	X	X	X	X	X
BACKGROUNDIMAGE=	X	X	X	X	X	X
BORDERBOTTOMCOLOR=	X	X		X		
BORDERBOTTOMSTYLE=	X	X	X	X		
BORDERBOTTOMWIDTH=	X	X	X	X		
BORDERLEFTCOLOR=	X	X		X		
BORDERLEFTSTYLE=	X	X	X	X		
BORDERLEFTWIDTH=	X	X	X	X		
BORDERCOLOR=	X	X		X	X	X
BORDERCOLORDARK=	X	X	X	X	X	X
BORDERCOLORLIGHT=	X	X	X	X	X	X
BODERRIGHTCOLOR=	X	X		X		
BODERRIGHTSTYLE=	X	X	X	X		
BODERRIGHTWIDTH=	X	X	X	X		
BODERTOPCOLOR=	X	X		X		

Attribute	PROC REPORT STATEMENT REPORT Area	PROC REPORT Areas: CALLDEF, COLUMN, HEADER, LINES, SUMMARY	PROC TABULATE STATEMENT TABLE	PROC TABULATE STATEMENTS VAR, CLASS, BOX, CLASSLEV, KEYWORD	PROC PRINT TABLE location	PROC PRINT: all locations other than TABLE
BORDERTOPSTYLE=	X	X	X	X		
BORDERTOPWIDTH =	X	X	X	X		
BORDERWIDTH=	X	X	X	X	X	X
CELLPADDING=	X		X		X	
CELLSPACING=	X		X		X	
CELLWIDTH=	X	X	X	X		X
CLASS=	X	X	X	X	X	X
COLOR=	X	X	X			
FLYOVER=	X	X		X		X
FONT=	X	X	X	X	X	X
FONTFAMILY=	X	X	X	X	X	X
FONTSIZE=	X	X	X	X	X	X
FONTSTYLE=	X	X	X	X	X	X
FONTWEIGHT=	X	X	X	X	X	X
FONTWIDTH=	X	X	X	X		X
FRAME=	X		X		X	
HEIGHT=	X	X		X	X	X
HREFTARGET=		X		X		X
HTMLSTYLE=	X	X	X	X	X	
NOBREAKSPACE=2	X	X		X		X
OUTPUTWIDTH=	X	X	X	X	X	
POSTHTML=1	X	X	X	X	X	X

Attribute	PROC REPORT STATEMENT REPORT Area	PROC REPORT Areas: CALLDEF, COLUMN, HEADER, LINES, SUMMARY	PROC TABULATE STATEMENT TABLE	PROC TABULATE STATEMENTS VAR, CLASS, BOX, CLASSLEV, KEYWORD	PROC PRINT TABLE location	PROC PRINT: all locations other than TABLE
POSTIMAGE=	X	X	X	X	X	X
POSTTEXT= ¹	X	X	X	X	X	X
PREHTML= ¹	X	X	X	X	X	X
PREIMAGE=	X	X	X	X	X	X
PRETEXT= ¹	X	X	X	X	X	X
PROTECTSPECIALC HARS=		X		X	X	X
RULES=	X		X		X	
TAGATTR=	X	X	X	X	X	X
TEXTALIGN=	X	X	X	X	X	X
URL=		X		X		X
VERTICALALIGN=		X		X		X
WIDTH=	X	X	X	X	X	

- ¹ When you use these attributes in this location, they affect only the text that is specified with the PRETEXT=, POSTTEXT=, PREHTML=, and POSTHTML= attributes. To alter the foreground color or the font for the text that appears in the table, you must set the corresponding attribute in a location that affects the cells rather than the table. For complete documentation about style attributes and their values, see ["Style Attributes" in SAS Output Delivery System: Procedures Guide](#).
- ² To help prevent unexpected wrapping of long text strings when using PROC REPORT with the ODS RTF destination, set NOBREAKSPACE=OFF in a location that affects the LINE statement. The NOBREAKSPACE=OFF attribute must be set in the PROC REPORT code either on the LINE statement or on the PROC REPORT statement where style(lines) is specified.

Style Elements and Style Attributes for Table Regions

The following table lists the default style elements and style attributes for various regions of a PROC TABULATE table.

The table lists defaults for the most commonly used ODS destinations: HTML5, PDF, and RTF. Each destination has a default style template that is applied to all output that is written to the destination. For complete documentation about the

ODS destinations and their default styles, see [“Style Templates” in SAS Output Delivery System: Procedures Guide](#).

Note: The default style for ODS output in SAS Data and AI Studio is set by the theme that the session is actively using. Depending on the theme, the default is either Illuminate, Ignite, or HighContrast. For more information, see [“Customizing SAS Data and AI Studio” in SAS Data and AI Studio: User’s Guide](#).

- The default style for HTML5 output is HTMLEncore.
- The default style for PDF output is Pearl.
- The default style for RTF output is RTF.

Table 61.6 Default Style Elements and Style Attributes for Table Regions

Region	Style Element	HTML5 Style Attributes	PDF Style Attributes	RTF Style Attributes
Column headings and Box	Header	FONTFAMILY = "Arial, 'Albany AMT', Helvetica, Helv" FONTSIZE = 2 FONTWEIGHT = bold FONTSTYLE = roman COLOR = cx112277 BACKGROUNDCOLOR = cxd2f2f9	FONTFAMILY = "Arial, 'Albany AMT'" FONTSIZE = 8pt FONTWEIGHT = bold FONTSTYLE = roman COLOR = cx000000 BACKGROUNDCOLOR = cxffffff BORDERWIDTH = NaN	FONTFAMILY = ""Times New Roman', 'Times Roman'" FONTSIZE = 11pt FONTWEIGHT = bold FONTSTYLE = roman COLOR = cx000000 BACKGROUNDCOLOR = cxbbbbbb
Page dimension text	Beforecaption	FONTFAMILY = "Arial, 'Albany AMT', Helvetica, Helv" FONTSIZE = 2 FONTWEIGHT = bold FONTSTYLE = roman COLOR = cx112277 BACKGROUNDCOLOR = cxfafbfe	FONTFAMILY = "Arial, 'Albany AMT'" FONTSIZE = 8pt FONTWEIGHT = bold FONTSTYLE = roman COLOR = cx000000 BACKGROUNDCOLOR = cxffffff	FONTFAMILY = ""Times New Roman', 'Times Roman'" FONTSIZE = 11pt FONTWEIGHT = bold FONTSTYLE = roman COLOR = cx000000
Row headings	Rowheader	FONTFAMILY = "Arial, 'Albany AMT', Helvetica, Helv" FONTSIZE = 2 FONTWEIGHT = bold FONTSTYLE = roman	FONTFAMILY = "Arial, 'Albany AMT'" FONTSIZE = 8pt FONTWEIGHT = bold FONTSTYLE = roman	FONTFAMILY = ""Times New Roman', 'Times Roman'" FONTSIZE = 11pt FONTWEIGHT = bold

Region	Style Element	HTML5 Style Attributes	PDF Style Attributes	RTF Style Attributes
		COLOR = cx112277 BACKGROUNDCOLOR = cxd2f2f9	COLOR = cx000000 BACKGROUNDCOLOR = cxffffff BORDERWIDTH = NaN	FONTSTYLE = roman COLOR = cx000000 BACKGROUNDCOLOR = cxbbbbbb
Data cells	Data	FONTFAMILY = "Arial, 'Albany AMT', Helvetica, Helv" FONTSIZE = 2 FONTWEIGHT = medium FONTSTYLE = roman BACKGROUNDCOLOR = cxffffff	FONTFAMILY = "'Albany AMT', Albany" FONTSIZE = 8pt FONTWEIGHT = medium FONTSTYLE = roman COLOR = cx000000 BORDERWIDTH = NaN COLOR = cx000000	FONTFAMILY = "'Times New Roman', 'Times Roman'" FONTSIZE = 10pt FONTWEIGHT = medium FONTSTYLE = roman COLOR = cx000000

Using the STYLE= Option with PROC TABULATE Statements

PROC TABULATE style overrides are applied to a table based on the statements that create the table. With the STYLE= option, you can make changes to either the style element or style attributes that control the appearance of an area of the output by specifying the STYLE= option in the statement that controls the area that you want to modify. The following table shows the areas of a table that can be modified and their corresponding statements.

Specifications in the TABLE statement override the same specification in the PROC TABULATE, CLASS, CLASSLEV, VAR, and KEYWORD statements. This enables you to have different style behavior with multiple TABLE statements. However, any style attributes that you specify in the PROC TABULATE statement and that you do not override in the TABLE statement are inherited. For example, if you specify a blue background and a white foreground for all data cells in the PROC TABULATE statement, and you specify a gray background for the data cells of a particular crossing in the TABLE statement, then the background for those data cells is gray, and the foreground is white (as specified in the PROC TABULATE statement).

Detailed information about the STYLE= option is provided in the documentation for individual statements.

Table 61.7 Using the STYLE= Option in PROC TABULATE

Area To Be Modified	STYLE Option To Use
Data cells	PROC TABULATE statement or dimension expression(s) (p. 2216)
Page dimension text and class variable name headings	“CLASS Statement” (p. 2234)
Class level value headings	“CLASSLEV Statement” (p. 2241)
Keyword headings	“KEYWORD Statement” (p. 2246)
Table borders, rules, and other parts that are not specified elsewhere	“TABLE Statement” (p. 2250)
Box text	BOX= option of the TABLE statement (p. 2250)
Missing values	MISSTEXT= option of the TABLE statement (p. 2250)
Analysis variable name headings	“VAR Statement” (p. 2264)

Figure 61.7 PROC TABULATE Areas and Corresponding Statements

table			
box		var	var
		keyword	keyword
class	class		
classlev		proc / x	proc / x
classlev	classlev	proc / x	proc / x
	keyword	all / x	all / x

Applying Style Attributes to Table Cells

PROC TABULATE determines the style attributes to use for a particular cell from the following default order of precedence for styles:

- 1 If no other style attributes are specified, then PROC TABULATE uses the default style attributes from the default style (Data).
- 2 The STYLE= option in the PROC TABULATE statement changes the default style attributes. If no other STYLE= option specifications affect a cell, then PROC TABULATE uses these style attributes for that cell.
- 3 A STYLE= option that is specified in the page dimension applies to all the table cells on the logical page unless you specify another STYLE= option for a cell in the row or column dimension.
- 4 A STYLE= option that is specified in the row dimension applies to all the table cells in the row unless you specify another STYLE= option for a cell in the column dimension.
- 5 A STYLE= option that is specified in the column dimension applies to all the table cells in the column.

You can change this order of precedence by using the STYLE_PRECEDENCE= option in the [TABLE statement on page 2260](#). For example, if you specify STYLE_PRECEDENCE=ROW and specify a STYLE= option in the row dimension, then those style attribute values override all others that are specified for the table cells.

Using a Format to Assign a Style Attribute

You can use a format to assign a style attribute value to any cell whose content is determined by values of a class or analysis variable. For example, the following code assigns a red background to cells whose values are less than 10,000, a yellow background to cells whose values are at least 10,000 but less than 20,000, and a green background to cells whose values are at least 20,000:

```
proc format;
    value expfmt low-<10000='red'
                    10000-<20000='yellow'
                    20000-high='green';
run;

ods html5 file='external-HTML-file';
proc tabulate data=energy style=[backgroundcolor=expfmt.];
    class region division type;
    var expenditures;
    table (region all)*(division all),
           type*expenditures;
run;
ods html5 close;
```


Specifying Style Attributes and Style Elements in Dimension Expressions

You can specify one or more style elements or style attributes in a dimension expression to control the appearance of non-LISTING output. You can modify the appearance of the following areas:

- analysis variable name headings
- class variable name headings
- class variable level value headings
- data cells
- keyword headings
- page dimension text

Specifying a style attribute or style element in a dimension expression is useful when you want to override a style attribute or style element that you have specified in another statement, such as the PROC TABULATE, CLASS, CLASSLEV, KEYWORD, TABLE, or VAR statements.

The syntax for specifying style elements and style attributes in a dimension expression is

[STYLE<(CLASSLEV)>=<style-element-name | PARENT>

[style-attribute-name-1=style-attribute-value-1< style-attribute-name-2=style-attribute-value-2 ...>]]

These are some examples of style attributes in dimension expressions:

- ```
dept={label='Department'
 style=[color=red]}, N
```
- dept\*[style=MyDataStyle], N
- dept\*[format=12.2 style=MyDataStyle], N

**Note:** When used in a dimension expression, the STYLE= option must be enclosed within square brackets ([ and ]) or braces ({ and }).

### (CLASSLEV)

assigns a style element to a class variable level value heading. For example, the following TABLE statement specifies that the level value heading for the class variable, DEPT, has a foreground color of yellow:

```
table dept=[style(classlev)=
 [color=yellow]]*sales;
```

**Note:** The CLASSLEV option is used only in dimension expressions.

For an example that shows how to specify style elements within dimension expressions, see [“Example 13: Specifying Style Overrides for ODS Output” on page 2357](#). For information about using styles with PROC TABULATE, see [“Using ODS Styles with PROC TABULATE” on page 2278](#).

## SAS Cloud Analytic Services Processing for PROC TABULATE

If your input data set originates from SAS Cloud Analytic Services (CAS), some of the PROC TABULATE analysis can be performed by the CAS server. Reports that require significant summarization of data can benefit from CAS processing. Running PROC TABULATE with CAS actions has several advantages over processing within SAS. These advantages include reduced network traffic, and the potential for faster processing. Faster processing is possible because in-memory tables are manipulated locally on the server instead of being transferred across a relatively slow network connection. CAS is used because it might have more processing resources at its disposal.

When the DATA= input data set references an in-memory table or view in CAS, the TABULATE procedure can use CAS actions to perform a significant portion of its work within the server. To reference an in-memory table or view, you must specify the CAS engine LIBNAME statement and use the CAS engine libref with the input table name.

By default, PROC TABULATE uses CAS processing whenever a CAS engine libref is specified on the input table name.

In the following example, the LIBNAME statement assigns a CAS engine libref named mycas that you use to connect to the CAS session casauto.

```
option casport=5570 cashost="cloud.example.com";
cas casauto ;
libname mycas cas;
data mycas.class;
 set sashelp.class;
run;

proc tabulate data=mycas.class;
 class sex age;
 var height weight;
 table sex,age*(height weight)*(sum mean);
run;
```

The following statistics are supported for CAS processing:

|      |        |
|------|--------|
| CSS  | RANGE  |
| CV   | STDERR |
| LCLM | SUM    |
| MAX  | SUMWGT |
| MEAN | STD    |
| MIN  | UCLM   |

|       |     |
|-------|-----|
| N     | USS |
| NMISS | VAR |

When SAS format definitions reside in CAS, formatting of class variables occurs in CAS. If the SAS format definitions do not reside on the CAS server, the CAS aggregation occurs on the raw values, and the relevant formats are applied by SAS as the results set is merged into the PROC TABULATE internal structure. User-defined formats that are created in SAS must be copied into CAS for them to work as expected. It is a best practice to keep formats consistent between SAS and CAS. For complete documentation about using user-defined formats with CAS, see [SAS Cloud Analytic Services: User-Defined Formats](#).

For information about how to use the CAS LIBNAME statement, see “[LIBNAME Statement: CAS Engine](#)” in [SAS Cloud Analytic Services: User's Guide](#). For more information about how procedures work with CAS processing, see [Chapter 4, “CAS Processing of Base Procedures,”](#) on page 75.

---

## In-Database Processing for PROC TABULATE

In-database processing has several advantages over processing within SAS. These advantages include increased security, reduced network traffic, and the potential for faster processing. Increased security is possible because sensitive data does not have to be extracted from the database management system (DBMS). Faster processing is possible because data is manipulated locally, on the DBMS, using high-speed secondary storage devices instead of being transported across a relatively slow network connection, because the DBMS might have more processing resources at its disposal, and because the DBMS might be capable of optimizing a query for execution in a highly parallel and scalable fashion.

When the DATA= input data set is stored as a table or view in a DBMS, the PROC TABULATE procedure can use in-database processing to perform most of its work within the database. In-database processing can provide the advantages of faster processing and reduced data transfer between the database and SAS software.

PROC TABULATE performs in-database processing by using SQL implicit pass-through. The procedure generates SQL queries that are based on the classifications and the statistics that you specify in the TABLE statement. The database executes these SQL queries to construct initial summary tables, which are then transmitted to PROC TABULATE.

If class variables are specified, the procedure creates an SQL GROUP BY clause that represents the n-way type. Only the n-way class tree is generated on the DBMS. The result set that is created when the aggregation query executes in the database is read by SAS into the internal PROC TABULATE data structure.

When SAS format definitions have been deployed in the database, formatting of class variables occurs in the database. If the SAS format definitions have not been deployed in the database, the in-database aggregation occurs on the raw values, and the relevant formats are applied by SAS as the results' set is merged into the PROC TABULATE internal structures. Multi-label formatting is always done by SAS using the initially aggregated result set that is returned by the database.

The following statistics are supported for in-database processing:

|       |        |
|-------|--------|
| CSS   | RANGE  |
| CV    | STDERR |
| LCLM  | SUM    |
| MAX   | SUMWGT |
| MEAN  | STD    |
| MIN   | UCLM   |
| N     | USS    |
| NMISS | VAR    |

The SQLGENERATION system option or LIBNAME statement option controls whether and how in-database procedures are run inside the database. By default, the in-database procedures are run inside the database when possible. There are many data set options that will prevent in-database processing. For a complete listing, see “In-Database Procedures” in *SAS/ACCESS for Relational Databases: Reference*.

PROC TABULATE is supported by the following databases:

- Amazon Redshift
- DB2
- Greenplum
- Hadoop
- IMPALA
- Microsoft SQL Server
- MySQL
- Netezza
- Oracle
- PostgreSQL
- SAP HANA
- Snowflake
- Spark
- Teradata
- Vertica

---

## Substituting BY Line Values in a Text String

#BYLINE, #BYVAR, and #BYVAL substitutions are available in the following options:

- The CONTENTS= option in the TABLE statement and the PROC TABULATE statement.

- The CAPTION= option in the TABLE statement

To use the #BYVAR and #BYVAL substitutions, insert the item in the text string at the position where you want the substitution text to appear. Both #BYVAR and #BYVAL specifications must be followed by a delimiting character. The character can be either a space or other non-alphanumeric character, such as a quotation mark. If no delimiting character is provided, then the specification is ignored and its text remains intact and is displayed with the rest of the string. To allow a #BYVAR or #BYVAL substitution to be followed immediately by other text, with no delimiter, use a trailing dot (as with macro variables). The trailing dot is not displayed in the resolved text. If you want a period to be displayed as the last character in the resolved text, use two dots after the #BYVAR or #BYVAL substitution.

The substitution for #BYVAR or #BYVAL does not occur in the following cases:

- if you use a #BYVAR or #BYVAL specification for a variable that is not named in the BY statement. For example, you might use #BYVAL2 when there is only one BY-variable or #BYVAL(ABC) when ABC is non-existent or is not a BY-variable.
- if there is no BY statement

---

## Results: TABULATE Procedure

---



---

### Missing Values

---

#### How PROC TABULATE Treats Missing Values

How a missing value for a variable in the input data set affects your output depends on how you use the variable in the PROC TABULATE step. The following table summarizes how the procedure treats missing values.

**Table 61.8** Summary of How PROC TABULATE Treats Missing Values

| Condition                                                        | PROC TABULATE Default                                                                                          | To Override Default |
|------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------|---------------------|
| An observation contains a missing value for an analysis variable | Excludes that observation from the calculation of statistics (except N and NMISS) for that particular variable | No alternative      |

---

| Condition                                                                                            | PROC TABULATE Default                                                         | To Override Default                                                                                                                                                                                                              |
|------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| An observation contains a missing value for a class variable                                         | Excludes that observation from the table <sup>1</sup>                         | Use MISSING in the PROC TABULATE statement, or MISSING in the CLASS statement                                                                                                                                                    |
| There are no data for a category                                                                     | Does not show the category in the table                                       | Use PRINTMISS in the TABLE statement, or use CLASSDATA= in the PROC TABULATE statement                                                                                                                                           |
| Every observation that contributes to a table cell contains a missing value for an analysis variable | Displays a missing value for any statistics (except N and NMISS) in that cell | Use MISSTEXT= in the TABLE statement                                                                                                                                                                                             |
| There are no data for a formatted value                                                              | Does not display that formatted value in the table                            | Use PRELOADFMT in the CLASS statement with PRINTMISS in the TABLE statement, or use CLASSDATA= in the PROC TABULATE statement, or add dummy observations to the input data set so that it contains data for each formatted value |
| A FREQ variable value is missing or is less than 1                                                   | Does not use that observation to calculate statistics                         | No alternative                                                                                                                                                                                                                   |
| A WEIGHT variable value is missing or 0                                                              | Uses a value of 0                                                             | No alternative                                                                                                                                                                                                                   |

This section presents a series of PROC TABULATE steps that illustrate how PROC TABULATE treats missing values. The following program creates the data set and formats that are used in this section and prints the data set. The data set COMPREV contains no missing values. (See the output below.)

```
proc format;
 value centryfmt 1='United States'
 2='Japan';
 value compfmt 1='Supercomputer'
 2='Mainframe'
 3='Midrange'
 4='Workstation'
 5='Personal Computer'
 6='Laptop';
run;
```

1. The CLASS statement applies to all TABLE statements in a PROC TABULATE step. Therefore, if you define a variable as a class variable, PROC TABULATE omits observations that have missing values for that variable even if you do not use the variable in a TABLE statement.

```

data comprev;
 input Country Computer Rev90 Rev91 Rev92;
 datalines;
1 1 788.8 877.6 944.9
1 2 12538.1 9855.6 8527.9
1 3 9815.8 6340.3 8680.3
1 4 3147.2 3474.1 3722.4
1 5 18660.9 18428.0 23531.1
2 1 469.9 495.6 448.4
2 2 5697.6 6242.4 5382.3
2 3 5392.1 5668.3 4845.9
2 4 1511.6 1875.5 1924.5
2 5 4746.0 4600.8 4363.7
;

proc print data=comprev noobs;
 format country centryfmt. computer compfmt.;
 title 'The Data Set COMPREV';
run;

```

**Output 61.5** *The Data Set COMPREV*

| The Data Set COMPREV |                   |         |         |         |
|----------------------|-------------------|---------|---------|---------|
| Country              | Computer          | Rev90   | Rev91   | Rev92   |
| United States        | Supercomputer     | 788.8   | 877.6   | 944.9   |
| United States        | Mainframe         | 12538.1 | 9855.6  | 8527.9  |
| United States        | Midrange          | 9815.8  | 6340.3  | 8680.3  |
| United States        | Workstation       | 3147.2  | 3474.1  | 3722.4  |
| United States        | Personal Computer | 18660.9 | 18428.0 | 23531.1 |
| Japan                | Supercomputer     | 469.9   | 495.6   | 448.4   |
| Japan                | Mainframe         | 5697.6  | 6242.4  | 5382.3  |
| Japan                | Midrange          | 5392.1  | 5668.3  | 4845.9  |
| Japan                | Workstation       | 1511.6  | 1875.5  | 1924.5  |
| Japan                | Personal Computer | 4746.0  | 4600.8  | 4363.7  |

## No Missing Values

The following PROC TABULATE step produces the following output:

```

proc tabulate data=comprev;
 class country computer;
 var rev90 rev91 rev92;
 table computer*country, rev90 rev91 rev92 /
 rts=32;
 format country centryfmt. computer compfmt.;
 title 'Revenues from Computer Sales';
 title2 'for 1990 to 1992';
run;

```

Because the data set contains no missing values, the table includes all observations. All headings and cells contain nonmissing values.

**Output 61.6** Computer Sales Data: No Missing Values

| Revenues from Computer Sales<br>for 1990 to 1992 |               |          |          |          |
|--------------------------------------------------|---------------|----------|----------|----------|
|                                                  |               | Rev90    | Rev91    | Rev92    |
|                                                  |               | Sum      | Sum      | Sum      |
| Computer                                         | Country       |          |          |          |
| Supercomputer                                    | United States | 788.80   | 877.60   | 944.90   |
|                                                  | Japan         | 469.90   | 495.60   | 448.40   |
| Mainframe                                        | United States | 12538.10 | 9855.60  | 8527.90  |
|                                                  | Japan         | 5697.60  | 6242.40  | 5382.30  |
| Midrange                                         | United States | 9815.80  | 6340.30  | 8680.30  |
|                                                  | Japan         | 5392.10  | 5668.30  | 4845.90  |
| Workstation                                      | United States | 3147.20  | 3474.10  | 3722.40  |
|                                                  | Japan         | 1511.60  | 1875.50  | 1924.50  |
| Personal Computer                                | United States | 18660.90 | 18428.00 | 23531.10 |
|                                                  | Japan         | 4746.00  | 4600.80  | 4363.70  |

## A Missing Class Variable

The next program copies COMPREV and alters the data so that the eighth observation has a missing value for Computer. Except for specifying this new data set, the program that produces the output “Computer Sales Data: Midrange, Japan, Deleted” below, is the same as the program that produces the output “Computer Sales Data: No Missing Values”, above. PROC TABULATE ignores observations with missing values for a class variable.

```
data compmiss;
 set comprev;
 if _n_=8 then computer=.;
run;

proc tabulate data=compmiss;
 class country computer;
 var rev90 rev91 rev92;
 table computer*country, rev90 rev91 rev92 /
 rts=32;
 format country cnyfmt. computer compfmt.;
 title 'Revenues from Computer Sales';
 title2 'for 1990 to 1992';
run;
```

The observation with a missing value for Computer was the category **Midrange, Japan**. This category no longer exists. By default, PROC TABULATE ignores



observations with missing values for a class variable, so this table contains one less row than the output "Computer Sales Data: No Missing Values".

**Output 61.7** Computer Sales Data: Midrange, Japan, Deleted

| Revenues from Computer Sales<br>for 1990 to 1992 |               |          |          |          |
|--------------------------------------------------|---------------|----------|----------|----------|
|                                                  |               | Rev90    | Rev91    | Rev92    |
|                                                  |               | Sum      | Sum      | Sum      |
| Computer                                         | Country       |          |          |          |
| Supercomputer                                    | United States | 788.80   | 877.60   | 944.90   |
|                                                  | Japan         | 469.90   | 495.60   | 448.40   |
| Mainframe                                        | United States | 12538.10 | 9855.60  | 8527.90  |
|                                                  | Japan         | 5697.60  | 6242.40  | 5382.30  |
| Midrange                                         | United States | 9815.80  | 6340.30  | 8680.30  |
| Workstation                                      | United States | 3147.20  | 3474.10  | 3722.40  |
|                                                  | Japan         | 1511.60  | 1875.50  | 1924.50  |
| Personal Computer                                | United States | 18660.90 | 18428.00 | 23531.10 |
|                                                  | Japan         | 4746.00  | 4600.80  | 4363.70  |

## Including Observations with Missing Class Variables

This program adds the MISSING option to the previous program. MISSING is available either in the PROC TABULATE statement or in the CLASS statement. If you want MISSING to apply only to selected class variables, but not to others, then specify MISSING in a separate CLASS statement with the selected variables. The MISSING option includes observations with missing values of a class variable in the report. (See the following output.)

```
proc tabulate data=compmiss missing;
 class country computer;
 var rev90 rev91 rev92;
 table computer*country, rev90 rev91 rev92 /
 rts=32;
 format country centryfmt. computer compfmt.;
 title 'Revenues from Computer Sales';
 title2 'for 1990 to 1992';
run;
```

This table includes a category with missing values of Computer. This category makes up the first row of data in the table.

**Output 61.8** Computer Sales Data: Missing Values for Computer

| Revenues from Computer Sales<br>for 1990 to 1992 |               |          |          |          |
|--------------------------------------------------|---------------|----------|----------|----------|
|                                                  |               | Rev90    | Rev91    | Rev92    |
|                                                  |               | Sum      | Sum      | Sum      |
| Computer                                         | Country       |          |          |          |
| .                                                | Japan         | 5392.10  | 5668.30  | 4845.90  |
| Supercomputer                                    | United States | 788.80   | 877.60   | 944.90   |
|                                                  | Japan         | 469.90   | 495.60   | 448.40   |
| Mainframe                                        | United States | 12538.10 | 9855.60  | 8527.90  |
|                                                  | Japan         | 5697.60  | 6242.40  | 5382.30  |
| Midrange                                         | United States | 9815.80  | 6340.30  | 8680.30  |
| Workstation                                      | United States | 3147.20  | 3474.10  | 3722.40  |
|                                                  | Japan         | 1511.60  | 1875.50  | 1924.50  |
| Personal Computer                                | United States | 18660.90 | 18428.00 | 23531.10 |
|                                                  | Japan         | 4746.00  | 4600.80  | 4363.70  |

## Formatting Headings for Observations with Missing Class Variables

By default, as shown in the output “Computer Sales Data: Missing Values for Computer”, PROC TABULATE displays missing values of a class variable as one of the standard SAS characters for missing values (a period, a blank, an underscore, or one of the letters A through Z). If you want to display something else instead, then you must assign a format to the class variable that has missing values, as shown in the following program. (See the following output.)

```
proc format;
 value misscomp 1='Supercomputer'
 2='Mainframe'
 3='Midrange'
 4='Workstation'
 5='Personal Computer'
 6='Laptop'
 .='No type given';
run;

proc tabulate data=compmiss missing;
 class country computer;
 var rev90 rev91 rev92;
 table computer*country, rev90 rev91 rev92 /
 rts=32;
 format country centryfmt. computer misscomp.;
 title 'Revenues for Computer Sales';
 title2 'for 1990 to 1992';
run;
```

In this table, the missing value appears as the text that the MISSCOMP. format specifies.

**Output 61.9** Computer Sales Data: Text Supplied for Missing Computer Value

| Revenues for Computer Sales<br>for 1990 to 1992 |               |          |          |          |
|-------------------------------------------------|---------------|----------|----------|----------|
|                                                 |               | Rev90    | Rev91    | Rev92    |
|                                                 |               | Sum      | Sum      | Sum      |
| Computer                                        | Country       |          |          |          |
| No type given                                   | Japan         | 5392.10  | 5668.30  | 4845.90  |
| Supercomputer                                   | United States | 788.80   | 877.60   | 944.90   |
|                                                 | Japan         | 469.90   | 495.60   | 448.40   |
| Mainframe                                       | United States | 12538.10 | 9855.60  | 8527.90  |
|                                                 | Japan         | 5697.60  | 6242.40  | 5382.30  |
| Midrange                                        | United States | 9815.80  | 6340.30  | 8680.30  |
| Workstation                                     | United States | 3147.20  | 3474.10  | 3722.40  |
|                                                 | Japan         | 1511.60  | 1875.50  | 1924.50  |
| Personal Computer                               | United States | 18660.90 | 18428.00 | 23531.10 |
|                                                 | Japan         | 4746.00  | 4600.80  | 4363.70  |

## Providing Headings for All Categories

By default, PROC TABULATE evaluates each page that it prints and omits columns and rows for categories that do not exist. For example, “Computer Sales Data: Text Supplied for Missing Computer Value ” does not include a row for **No type given** and for **United States** or for **Midrange** and for **Japan** because there are no data in these categories. If you want the table to represent all possible categories, then use the PRINTMISS option in the TABLE statement, as shown in the following program. (See the following output.)

```
proc tabulate data=compmiss missing;
 class country computer;
 var rev90 rev91 rev92;
 table computer*country, rev90 rev91 rev92 /
 rts=32 printmiss;
 format country ctryfmt. computer misscomp.;
 title 'Revenues for Computer Sales';
 title2 'for 1990 to 1992';
run;
```

This table contains a row for the category **No type given**, **United States** and the category **Midrange**, **Japan**. Because there are no data in these categories, the values for the statistics are all missing.

**Output 61.10** Computer Sales Data: Missing Statistics Values

| Revenues for Computer Sales<br>for 1990 to 1992 |               |          |          |          |
|-------------------------------------------------|---------------|----------|----------|----------|
|                                                 |               | Rev90    | Rev91    | Rev92    |
|                                                 |               | Sum      | Sum      | Sum      |
| Computer                                        | Country       |          |          |          |
| No type given                                   | United States | .        | .        | .        |
|                                                 | Japan         | 5392.10  | 5668.30  | 4845.90  |
| Supercomputer                                   | United States | 788.80   | 877.60   | 944.90   |
|                                                 | Japan         | 469.90   | 495.60   | 448.40   |
| Mainframe                                       | United States | 12538.10 | 9855.60  | 8527.90  |
|                                                 | Japan         | 5697.60  | 6242.40  | 5382.30  |
| Midrange                                        | United States | 9815.80  | 6340.30  | 8680.30  |
|                                                 | Japan         | .        | .        | .        |
| Workstation                                     | United States | 3147.20  | 3474.10  | 3722.40  |
|                                                 | Japan         | 1511.60  | 1875.50  | 1924.50  |
| Personal Computer                               | United States | 18660.90 | 18428.00 | 23531.10 |
|                                                 | Japan         | 4746.00  | 4600.80  | 4363.70  |

## Providing Text for Cells That Contain Missing Values

If some observations in a category contain missing values for analysis variables, then PROC TABULATE does not use those observations to calculate statistics (except N and NMISS). However, if each observation in a category contains a missing value, then PROC TABULATE displays a missing value for the value of the statistic. To replace missing values for analysis variables with text, use the MISSTEXT= option in the TABLE statement to specify the text to use, as shown in the following program. (See the following output.)

```
proc tabulate data=compmiss missing;
 class country computer;
 var rev90 rev91 rev92;
 table computer*country, rev90 rev91 rev92 /
 rts=32 printmiss misstext='NO DATA!';
 format country cnyrfmt. computer misscomp.;
 title 'Revenues for Computer Sales';
 title2 'for 1990 to 1992';
run;
```

This table replaces the period normally used to display missing values with the text of the MISSTEXT= option.

**Output 61.11** Computer Sales Data: Text Supplied for Missing Statistics Values

| Revenues for Computer Sales<br>for 1990 to 1992 |               |          |          |          |
|-------------------------------------------------|---------------|----------|----------|----------|
|                                                 |               | Rev90    | Rev91    | Rev92    |
|                                                 |               | Sum      | Sum      | Sum      |
| Computer                                        | Country       |          |          |          |
| No type given                                   | United States | NO DATA! | NO DATA! | NO DATA! |
|                                                 | Japan         | 5392.10  | 5668.30  | 4845.90  |
| Supercomputer                                   | United States | 788.80   | 877.60   | 944.90   |
|                                                 | Japan         | 469.90   | 495.60   | 448.40   |
| Mainframe                                       | United States | 12538.10 | 9855.60  | 8527.90  |
|                                                 | Japan         | 5697.60  | 6242.40  | 5382.30  |
| Midrange                                        | United States | 9815.80  | 6340.30  | 8680.30  |
|                                                 | Japan         | NO DATA! | NO DATA! | NO DATA! |
| Workstation                                     | United States | 3147.20  | 3474.10  | 3722.40  |
|                                                 | Japan         | 1511.60  | 1875.50  | 1924.50  |
| Personal Computer                               | United States | 18660.90 | 18428.00 | 23531.10 |
|                                                 | Japan         | 4746.00  | 4600.80  | 4363.70  |

## Providing Headings for All Values of a Format

PROC TABULATE prints headings only for values that appear in the input data set. For example, the format COMPFMT. provides for six possible values of Computer. Only five of these values occur in the data set COMPREV. The data set contains no data for laptop computers.

If you want to include headings for all possible values of Computer (perhaps to make it easier to compare the output with tables that are created later when you do have data for laptops), then you have three different ways to create such a table:

- Use the PRELOADFMT option in the CLASS statement with the PRINTMISS option in the TABLE statement. See [“Example 3: Using Preloaded Formats with Class Variables” on page 2311](#) for another example that uses PRELOADFMT.
- Use the CLASSDATA= option in the PROC TABULATE statement. See [“Example 2: Specifying Class Variable Combinations to Appear in a Table” on page 2308](#) for an example that uses the CLASSDATA= option.
- Add dummy values to the input data set so that each value that the format handles appears at least once in the data set.

The following program adds the PRELOADFMT option to a CLASS statement that contains the relevant variable.

The results are shown in the following output.

```
proc tabulate data=compmiss missing;
 class country;
```

```

class computer / preloadfmt;
var rev90 rev91 rev92;
table computer*country,rev90 rev91 rev92 /
 rts=32 printmiss misstext='NO DATA!';
format country centryfmt. computer compfmt.;
title 'Revenues for Computer Sales';
title2 'for 1990 to 1992';
run;

```

This table contains a heading for each possible value of Computer.

**Output 61.12** *Computer Sales Data: All Possible Computer Values Included*

| Revenues for Computer Sales<br>for 1990 to 1992 |               |          |          |          |
|-------------------------------------------------|---------------|----------|----------|----------|
|                                                 |               | Rev90    | Rev91    | Rev92    |
|                                                 |               | Sum      | Sum      | Sum      |
| Computer                                        | Country       |          |          |          |
| .                                               | United States | NO DATA! | NO DATA! | NO DATA! |
|                                                 | Japan         | 5392.10  | 5668.30  | 4845.90  |
| Supercomputer                                   | United States | 788.80   | 877.60   | 944.90   |
|                                                 | Japan         | 469.90   | 495.60   | 448.40   |
| Mainframe                                       | United States | 12538.10 | 9855.60  | 8527.90  |
|                                                 | Japan         | 5697.60  | 6242.40  | 5382.30  |
| Midrange                                        | United States | 9815.80  | 6340.30  | 8680.30  |
|                                                 | Japan         | NO DATA! | NO DATA! | NO DATA! |
| Workstation                                     | United States | 3147.20  | 3474.10  | 3722.40  |
|                                                 | Japan         | 1511.60  | 1875.50  | 1924.50  |
| Personal Computer                               | United States | 18660.90 | 18428.00 | 23531.10 |
|                                                 | Japan         | 4746.00  | 4600.80  | 4363.70  |
| Laptop                                          | United States | NO DATA! | NO DATA! | NO DATA! |
|                                                 | Japan         | NO DATA! | NO DATA! | NO DATA! |

## Understanding the Order of Headings with ORDER=DATA

The ORDER= option applies to all class variables. Occasionally, you want to order the headings for different variables differently. One method for reordering the headings is to group the data as you want them to appear and to specify ORDER=DATA.

For this technique to work, the first value of the first class variable must occur in the data with all possible values of all the other class variables. If this criterion is not met, then the order of the headings might surprise you.

The following program creates a simple data set in which the observations are ordered first by the values of Animal, then by the values of Food. The ORDER= option in the PROC TABULATE statement orders the heading for the class variables by the order of their appearance in the data set. (See the following output.) Although bones is the first value for Food in the group of observations where Animal=dog, all other values for Food appear before bones in the data set because bones never appears when Animal=cat. Therefore, the heading for bones in the table in the following output is not in alphabetical order.

In other words, PROC TABULATE maintains for subsequent categories the order that was established by earlier categories. If you want to re-establish the order of Food for each value of Animal, then use BY-group processing. PROC TABULATE creates a separate table for each BY group, so that the ordering can differ from one BY group to the next.

```

title "Animal Food Preference";
data foodpref;
 input Animal $ Food $;
 datalines;
cat fish
cat meat
cat milk
dog bones
dog fish
dog meat
;

proc tabulate data=foodpref format=9.
 order=data;
 class animal food;
 table animal*food;
run;

```

**Output 61.13** Ordering the Headings of Class Variables

| Animal Food Preference |      |      |      |      |       |
|------------------------|------|------|------|------|-------|
| Animal                 |      |      |      |      |       |
| cat                    |      |      | dog  |      |       |
| Food                   |      |      | Food |      |       |
| fish                   | meat | milk | fish | meat | bones |
| N                      | N    | N    | N    | N    | N     |
| 1                      | 1    | 1    | 1    | 1    | 1     |

---

# Examples: TABULATE Procedure

---

---

## Example 1: Creating a Basic Two-Dimensional Table

---

|           |                                                                                                                                                                                                                             |
|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Features: | CLASS statement<br>PROC TABULATE statement options<br>DATA=<br>FORMAT=<br>TABLE statement options<br>crossing (*) operator<br>RTS=<br>VAR statement<br>DATA step<br>FORMAT procedure<br>FORMAT statement<br>TITLE statement |
| Data set: | ENERGY                                                                                                                                                                                                                      |

---

---

## Details

The following example program does the following:

- creates a category for each type of user (residential or business) in each division of each region
- applies the same format to all cells in the table
- applies a format to each class variable
- extends the space for row headings

---

## Program

```
proc format;
 value regfmt 1='Northeast'
 2='South'
 3='Midwest'
```



```

 4='West';
value divfmt 1='New England'
 2='Middle Atlantic'
 3='Mountain'
 4='Pacific';
value usetype 1='Residential Customers'
 2='Business Customers';
run;

proc tabulate data=energy format=dollar12.;
 class region division type;

 var expenditures;

 table region*division,
 type*expenditures
 / rts=25;

 format region regfmt. division divfmt. type usetype.;

 title 'Energy Expenditures for Each Region';
 title2 '(millions of dollars)';
run;

```

---

## Program Description

**Create the REGFMT., DIVFMT., and USETYPE. formats.** PROC FORMAT creates formats for Region, Division, and Type.

```

proc format;
 value regfmt 1='Northeast'
 2='South'
 3='Midwest'
 4='West';
 value divfmt 1='New England'
 2='Middle Atlantic'
 3='Mountain'
 4='Pacific';
 value usetype 1='Residential Customers'
 2='Business Customers';
run;

```

**Specify the table options.** The FORMAT= option specifies DOLLAR12. as the default format for the value in each table cell.

```
proc tabulate data=energy format=dollar12.;
```

**Specify subgroups for the analysis.** The CLASS statement separates the analysis by values of Region, Division, and Type.

```
class region division type;
```

**Specify the analysis variable.** The VAR statement specifies that PROC TABULATE calculate statistics on the Expenditures variable.

```
var expenditures;
```

**Define the table rows and columns.** The TABLE statement creates a row for each formatted value of Region. Nested within each row are rows for each formatted value of Division. The TABLE statement also creates a column for each formatted value of Type. Each cell that is created by these rows and columns contains the sum of the analysis variable Expenditures for all observations that contribute to that cell.

```
table region*division,
 type*expenditures
```

**Specify the row title space.** RTS= provides 25 characters per line for row headings.

```
/ rts=25;
```

**Format the output.** The FORMAT statement assigns formats to the variables Region, Division, and Type.

```
format region regfmt. division divfmt. type usetype.;
```

**Specify the titles.**

```
title 'Energy Expenditures for Each Region';
title2 '(millions of dollars)';
run;
```

## Output

**Output 61.14** Basic Two-Dimensional Table

| Energy Expenditures for Each Region<br>(millions of dollars) |                 |                       |                    |
|--------------------------------------------------------------|-----------------|-----------------------|--------------------|
|                                                              |                 | Type                  |                    |
|                                                              |                 | Residential Customers | Business Customers |
|                                                              |                 | Expenditures          | Expenditures       |
|                                                              |                 | Sum                   | Sum                |
| Region                                                       | Division        |                       |                    |
| Northeast                                                    | New England     | \$7,477               | \$5,129            |
|                                                              | Middle Atlantic | \$19,379              | \$15,078           |
| West                                                         | Mountain        | \$5,476               | \$4,729            |
|                                                              | Pacific         | \$13,959              | \$12,619           |

## Example 2: Specifying Class Variable Combinations to Appear in a Table

Features: CLASS statement  
PROC TABULATE Statement options  
DATA=

CLASSDATA=  
 EXCLUSIVE  
 FORMAT=  
 TABLE statement options  
 crossing (\*) operator  
 RTS=  
 VAR statement  
 DATA step  
 FORMAT statement  
 TITLE statement

Data set:

ENERGY

## Details

This example does the following:

- uses the CLASSDATA= option to specify combinations of class variables to appear in a table.
- uses the EXCLUSIVE option to restrict the output to only the combinations specified in the CLASSDATA= data set. Without the EXCLUSIVE option, the output would be the same as in [“Example 1: Creating a Basic Two-Dimensional Table” on page 2306](#).

## Program

```

data classes;
 input region division type;
 datalines;
1 1 1
1 1 2
4 4 1
4 4 2
;

proc tabulate data=energy format=dollar12.
 classdata=classes exclusive;

 class region division type;

 var expenditures;

 table region*division,
 type*expenditures
 / rts=25;

 format region regfmt. division divfmt. type usetype.;

 title 'Energy Expenditures for Each Region';

```

```

 title2 '(millions of dollars)';
run;

```

---

## Program Description

**Create the CLASSES data set.** CLASSES contains the combinations of class variable values that PROC TABULATE uses to create the table.

```

data classes;
 input region division type;
 datalines;
1 1 1
1 1 2
4 4 1
4 4 2
;

```

**Specify the table options.** CLASSDATA= and EXCLUSIVE restrict the class level combinations to those that are specified in the CLASSES data set.

```

proc tabulate data=energy format=dollar12.
 classdata=classes exclusive;

```

**Specify subgroups for the analysis.** The CLASS statement separates the analysis by values of Region, Division, and Type.

```

class region division type;

```

**Specify the analysis variable.** The VAR statement specifies that PROC TABULATE calculate statistics on the Expenditures variable.

```

var expenditures;

```

**Define the table rows and columns.** The TABLE statement creates a row for each formatted value of Region. Nested within each row are rows for each formatted value of Division. The TABLE statement also creates a column for each formatted value of Type. Each cell that is created by these rows and columns contains the sum of the analysis variable Expenditures for all observations that contribute to that cell.

```

table region*division,
 type*expenditures

```

**Specify the row title space.** RTS= provides 25 characters per line for row headings.

```

/ rts=25;

```

**Format the output.** The FORMAT statement assigns formats to the variables Region, Division, and Type.

```

format region regfmt. division divfmt. type usetype.;

```

**Specify the titles.**

```

title 'Energy Expenditures for Each Region';
title2 '(millions of dollars)';
run;

```

## Output

**Output 61.15** *Energy Expenditures for Each Region*

| Energy Expenditures for Each Region<br>(millions of dollars) |             |                       |                    |
|--------------------------------------------------------------|-------------|-----------------------|--------------------|
|                                                              |             | Type                  |                    |
|                                                              |             | Residential Customers | Business Customers |
|                                                              |             | Expenditures          | Expenditures       |
|                                                              |             | Sum                   | Sum                |
| Region                                                       | Division    |                       |                    |
| Northeast                                                    | New England | \$7,477               | \$5,129            |
| West                                                         | Pacific     | \$13,959              | \$12,619           |

## Example 3: Using Preloaded Formats with Class Variables

Features:

- CLASS statement options
  - EXCLUSIVE
  - PRELOADFMT
- PROC TABULATE statement options
  - DATA=
  - FORMAT=
  - OUT=
- TABLE statement options
  - crossing (\*) operator
  - PRINTMISS
  - RTS
- VAR statement
- FORMAT statement
- PRINT procedure
- TITLE statement

Data set: [ENERGY](#)

## Details

This example does the following:

- creates a table that includes all possible combinations of formatted class variable values (PRELOADFMT with PRINTMISS), even if those combinations have a zero frequency and even if they do not make sense
- uses only the preloaded range of user-defined formats as the levels of class variables (PRELOADFMT with EXCLUSIVE)
- writes the output to an output data set, and prints that data set

---

## Program

```
proc tabulate data=energy format=dollar12.;
 class region division type / preloadfmt;
 var expenditures;
 table region*division,
 type*expenditures / rts=25 printmiss;
 format region regfmt. division divfmt. type usetype.;
 title 'Energy Expenditures for Each Region';
 title2 '(millions of dollars)';
run;

proc tabulate data=energy format=dollar12. out=tabdata;
 class region division type / preloadfmt exclusive;
 var expenditures;
 table region*division,
 type*expenditures / rts=25;
 format region regfmt. division divfmt. type usetype.;
 title 'Energy Expenditures for Each Region';
 title2 '(millions of dollars)';
run;

proc print data=tabdata;
run;
```

---

## Program Description

**Specify the table options.** The FORMAT= option specifies DOLLAR12. as the default format for the value in each table cell.

```
proc tabulate data=energy format=dollar12.;
```

**Specify subgroups for the analysis.** The CLASS statement separates the analysis by values of Region, Division, and Type. PRELOADFMT specifies that PROC TABULATE use the preloaded values of the user-defined formats for the class variables.

```
class region division type / preloadfmt;
```

**Specify the analysis variable.** The VAR statement specifies that PROC TABULATE calculate statistics on the Expenditures variable.

```
var expenditures;
```

**Define the table rows and columns, and specify row and column options.**

PRINTMISS specifies that all possible combinations of user-defined formats be used as the levels of the class variables.

```
table region*division,
 type*expenditures / rts=25 printmiss;
```

**Format the output.** The FORMAT statement assigns formats to the variables Region, Division, and Type.

```
format region regfmt. division divfmt. type usetype.;
```

**Specify the titles.**

```
title 'Energy Expenditures for Each Region';
title2 '(millions of dollars)';
run;
```

**Specify the table options and the output data set.** The OUT= option specifies the name of the output data set to which PROC TABULATE writes the data.

```
proc tabulate data=energy format=dollar12. out=tabdata;
```

**Specify subgroups for the analysis.** The EXCLUSIVE option, when used with PRELOADFMT, uses only the preloaded range of user-defined formats as the levels of class variables.

```
class region division type / preloadfmt exclusive;
```

**Specify the analysis variable.** The VAR statement specifies that PROC TABULATE calculate statistics on the Expenditures variable.

```
var expenditures;
```

**Define the table rows and columns, and specify row and column options.** The PRINTMISS option is not specified in this case. If it were, then it would override the EXCLUSIVE option in the CLASS statement.

```
table region*division,
 type*expenditures / rts=25;
```

**Format the output.** The FORMAT statement assigns formats to the variables Region, Division, and Type.

```
format region regfmt. division divfmt. type usetype.;
```

**Specify the titles.**

```
title 'Energy Expenditures for Each Region';
title2 '(millions of dollars)';
run;
```

**Print the output data set WORK.TABDATA.**

```
proc print data=tabdata;
run;
```

## Output

This output, created with the PRELOADFMT and PRINTMISS options, contains all possible combinations of preloaded user-defined formats for the class variable values. It includes combinations with zero frequencies, and combinations that make no sense, such as **Northeast** and **Pacific**.

**Output 61.16** *Energy Expenditures for Each Region*

| Energy Expenditures for Each Region<br>(millions of dollars) |                        |                       |                    |
|--------------------------------------------------------------|------------------------|-----------------------|--------------------|
|                                                              |                        | Type                  |                    |
|                                                              |                        | Residential Customers | Business Customers |
|                                                              |                        | Expenditures          | Expenditures       |
|                                                              |                        | Sum                   | Sum                |
| Region                                                       | Division               |                       |                    |
| <b>Northeast</b>                                             | <b>New England</b>     | \$7,477               | \$5,129            |
|                                                              | <b>Middle Atlantic</b> | \$19,379              | \$15,078           |
|                                                              | <b>Mountain</b>        | .                     | .                  |
|                                                              | <b>Pacific</b>         | .                     | .                  |
| <b>South</b>                                                 | <b>New England</b>     | .                     | .                  |
|                                                              | <b>Middle Atlantic</b> | .                     | .                  |
|                                                              | <b>Mountain</b>        | .                     | .                  |
|                                                              | <b>Pacific</b>         | .                     | .                  |
| <b>Midwest</b>                                               | <b>New England</b>     | .                     | .                  |
|                                                              | <b>Middle Atlantic</b> | .                     | .                  |
|                                                              | <b>Mountain</b>        | .                     | .                  |
|                                                              | <b>Pacific</b>         | .                     | .                  |
| <b>West</b>                                                  | <b>New England</b>     | .                     | .                  |
|                                                              | <b>Middle Atlantic</b> | .                     | .                  |
|                                                              | <b>Mountain</b>        | \$5,476               | \$4,729            |
|                                                              | <b>Pacific</b>         | \$13,959              | \$12,619           |

This output, created with the PRELOADFMT and EXCLUSIVE options, contains only those combinations of preloaded user-defined formats for the class variable values that appear in the input data set. This output is identical to the output from [“Example 1: Creating a Basic Two-Dimensional Table” on page 2306](#).



**Output 61.17** Energy Expenditures for Each Region

| Energy Expenditures for Each Region<br>(millions of dollars) |                 |                       |                    |
|--------------------------------------------------------------|-----------------|-----------------------|--------------------|
|                                                              |                 | Type                  |                    |
|                                                              |                 | Residential Customers | Business Customers |
|                                                              |                 | Expenditures          | Expenditures       |
|                                                              |                 | Sum                   | Sum                |
| Region                                                       | Division        |                       |                    |
| Northeast                                                    | New England     | \$7,477               | \$5,129            |
|                                                              | Middle Atlantic | \$19,379              | \$15,078           |
| West                                                         | Mountain        | \$5,476               | \$4,729            |
|                                                              | Pacific         | \$13,959              | \$12,619           |

This output shows the output data set TABDATA, which was created by the OUT= option in the PROC TABULATE statement. TABDATA contains the data that is created by having the PRELOADFMT and EXCLUSIVE options specified.

**Output 61.18** Energy Expenditures for Each Region

| Energy Expenditures for Each Region<br>(millions of dollars) |           |                 |                       |        |        |         |
|--------------------------------------------------------------|-----------|-----------------|-----------------------|--------|--------|---------|
| Obs                                                          | Region    | Division        | Type                  | _TYPE_ | _PAGE_ | _TABLE_ |
| 1                                                            | Northeast | New England     | Residential Customers | 111    | 1      | 1       |
| 2                                                            | Northeast | New England     | Business Customers    | 111    | 1      | 1       |
| 3                                                            | Northeast | Middle Atlantic | Residential Customers | 111    | 1      | 1       |
| 4                                                            | Northeast | Middle Atlantic | Business Customers    | 111    | 1      | 1       |
| 5                                                            | West      | Mountain        | Residential Customers | 111    | 1      | 1       |
| 6                                                            | West      | Mountain        | Business Customers    | 111    | 1      | 1       |
| 7                                                            | West      | Pacific         | Residential Customers | 111    | 1      | 1       |
| 8                                                            | West      | Pacific         | Business Customers    | 111    | 1      | 1       |

## Example 4: Using Multilabel Formats

Features:

- CLASS statement options
  - MLF
- PROC TABULATE statement options
  - DATA=
  - FORMAT=
- TABLE statement
  - ALL class variable
  - concatenation (blank) operator

crossing (\*) operator  
 grouping elements (parentheses) operator  
 label  
 variable list  
 VAR statement  
 DATA step  
 FORMAT procedure  
 FORMAT statement  
 TITLE statement

Data set:

CARSURVEY

## Details

This example does the following:

- shows how to specify a multilabel format in the VALUE statement of PROC FORMAT
- shows how to activate multilabel format processing using the MLF option with the CLASS statement
- demonstrates the behavior of the N statistic when multilabel format processing is activated

## Program

```

proc format;
 value agefmt (multilabel notsorted)
 15 - 29 = 'Below 30 years'
 30 - 50 = 'Between 30 and 50'
 51 - high = 'Over 50 years'
 15 - 19 = '15 to 19'
 20 - 25 = '20 to 25'
 25 - 39 = '25 to 39'
 40 - 55 = '40 to 55'
 56 - high = '56 and above';
run;

proc tabulate data=carsurvey format=10.;
 class age / mlf;
 var progressa remark jupiter dynamo;
 table age all, n all='Potential Car Names'*(progressa remark
 jupiter dynamo)*mean;

 title1 "Rating Four Potential Car Names";
 title2 "Rating Scale 0-100 (100 is the highest rating)";

```

```
format age agefmt.;
run;
```

## Program Description

**Create the AGEFMT. format.** The FORMAT procedure creates a multilabel format for ages by using the “[MULTILABEL](#)” on [page 913](#). A multilabel format is one in which multiple labels can be assigned to the same value, in this case because of overlapping ranges. Each value is represented in the table for each range in which it occurs. The NOTSORTED option stores the ranges in the order in which they are defined.

```
proc format;
 value agefmt (multilabel notsorted)
 15 - 29 = 'Below 30 years'
 30 - 50 = 'Between 30 and 50'
 51 - high = 'Over 50 years'
 15 - 19 = '15 to 19'
 20 - 25 = '20 to 25'
 25 - 39 = '25 to 39'
 40 - 55 = '40 to 55'
 56 - high = '56 and above';
run;
```

**Specify the table options.** The FORMAT= option specifies up to 10 digits as the default format for the value in each table cell.

```
proc tabulate data=carsurvey format=10.;
```

**Specify subgroups for the analysis.** The CLASS statement identifies Age as the class variable and uses the MLF option to activate multilabel format processing.

```
class age / mlf;
```

**Specify the analysis variables.** The VAR statement specifies that PROC TABULATE calculate statistics on the Progressa, Remark, Jupiter, and Dynamo variables.

```
var progressa remark jupiter dynamo;
```

**Define the table rows and columns.** The row dimension of the TABLE statement creates a row for each formatted value of Age. Multilabel formatting allows an observation to be included in multiple rows or age categories. The row dimension uses the ALL class variable to summarize information for all rows. The column dimension uses the N statistic to calculate the number of observations for each age group. Notice that the result of the N statistic crossed with the ALL class variable in the row dimension is the total number of observations instead of the sum of the N statistics for the rows. The column dimension uses the ALL class variable at the beginning of a crossing to assign a label, Potential Car Names. The four nested columns calculate the mean ratings of the car names for each age group.

```
table age all, n all='Potential Car Names'*(progressa remark
 jupiter dynamo)*mean;
```

**Specify the titles.**

```

title1 "Rating Four Potential Car Names";
title2 "Rating Scale 0-100 (100 is the highest rating)";

```

**Format the output.** The FORMAT statement assigns the user-defined format AGEFMT. to Age for this analysis.

```

format age agefmt.;
run;

```

## Output

*Output 61.19 Rating Four Potential Car Names*

| Rating Four Potential Car Names<br>Rating Scale 0-100 (100 is the highest rating) |    |                     |        |         |        |
|-----------------------------------------------------------------------------------|----|---------------------|--------|---------|--------|
|                                                                                   | N  | Potential Car Names |        |         |        |
|                                                                                   |    | Progressa           | Remark | Jupiter | Dynamo |
|                                                                                   |    | Mean                | Mean   | Mean    | Mean   |
| <b>Age</b>                                                                        |    |                     |        |         |        |
| 15 to 19                                                                          | 14 | 75                  | 78     | 81      | 73     |
| 20 to 25                                                                          | 11 | 89                  | 88     | 84      | 89     |
| 25 to 39                                                                          | 26 | 84                  | 90     | 82      | 72     |
| 40 to 55                                                                          | 14 | 85                  | 87     | 80      | 68     |
| 56 and above                                                                      | 15 | 84                  | 82     | 81      | 75     |
| Below 30 years                                                                    | 36 | 82                  | 84     | 82      | 75     |
| Between 30 and 50                                                                 | 25 | 86                  | 89     | 81      | 73     |
| Over 50 years                                                                     | 19 | 82                  | 84     | 80      | 76     |
| All                                                                               | 80 | 83                  | 86     | 81      | 74     |

## Example 5: Customizing Row and Column Headings

Features:

- CLASS statement
- PROC TABULATE statement options
  - DATA=
  - FORMAT=
- TABLE statement options
  - crossing (\*) operator
  - labels
  - RTS=
- VAR statement
- FORMAT statement
- TITLE statement

Data set: ENERGY

Format: REGFMT.

Format: DIVFMT.

Format: USETYPE.

---

## Details

This example shows how to customize row and column headings. A label specifies text for a heading. A blank label creates a blank heading. PROC TABULATE removes the space for blank column headings from the table.

## Program

```
proc tabulate data=energy format=dollar12.;
 class region division type;
 var expenditures;
 table region*division,
 type='Customer Base'*expenditures=' '*sum=' '
 / rts=25;
 format region regfmt. division divfmt. type usetype.;
 title 'Energy Expenditures for Each Region';
 title2 '(millions of dollars)';
run;
```

## Program Description

**Specify the table options.** The FORMAT= option specifies DOLLAR12. as the default format for the value in each table cell.

```
proc tabulate data=energy format=dollar12.;
```

**Specify subgroups for the analysis.** The CLASS statement identifies Region, Division, and Type as class variables.

```
class region division type;
```

**Specify the analysis variable.** The VAR statement specifies that PROC TABULATE calculate statistics on the Expenditures variable.

```
var expenditures;
```

**Define the table rows and columns.** The TABLE statement creates a row for each formatted value of Region. Nested within each row are rows for each formatted value of Division. The TABLE statement also creates a column for each formatted value of Type. Each cell that is created by these rows and columns contains the

sum of the analysis variable Expenditures for all observations that contribute to that cell. Text in quotation marks specifies headings for the corresponding variable or statistic. Although Sum is the default statistic, it is specified here so that you can specify a blank for its heading.

```
table region*division,
 type='Customer Base'*expenditures=' '*sum=' '
```

**Specify the row title space.** RTS= provides 25 characters per line for row headings.

```
/ rts=25;
```

**Format the output.** The FORMAT statement assigns formats to Region, Division, and Type.

```
format region regfmt. division divfmt. type usetype.;
```

**Specify the titles.**

```
title 'Energy Expenditures for Each Region';
title2 '(millions of dollars)';
run;
```

## Output

The heading for Type contains text that is specified in the TABLE statement. The TABLE statement eliminated the headings for Expenditures and Sum.

**Output 61.20** *Energy Expenditures for Each Region*

| Energy Expenditures for Each Region<br>(millions of dollars) |                 |                       |                    |
|--------------------------------------------------------------|-----------------|-----------------------|--------------------|
|                                                              |                 | Customer Base         |                    |
|                                                              |                 | Residential Customers | Business Customers |
| Region                                                       | Division        |                       |                    |
| Northeast                                                    | New England     | \$7,477               | \$5,129            |
|                                                              | Middle Atlantic | \$19,379              | \$15,078           |
| West                                                         | Mountain        | \$5,476               | \$4,729            |
|                                                              | Pacific         | \$13,959              | \$12,619           |

## Example 6: Summarizing Information with the Universal Class Variable ALL

Features:

CLASS statement  
PROC TABULATE statement options  
DATA=  
FORMAT=

TABLE statement  
 ALL class variable  
 concatenation (blank operator)  
 format modifiers  
 grouping elements (parentheses operator)  
 RTS=

VAR statement

FORMAT statement

TITLE statement

Data set: ENERGY

Format: REGFMT.

Format: DIVFMT.

Format: USETYPE.

## Details

This example shows how to use the universal class variable ALL to summarize information from multiple categories.

## Program

```
proc tabulate data=energy format=comma12.;
 class region division type;
 var expenditures;
 table region*(division all='Subtotal')
 all='Total for All Regions'*f=dollar12.,
 type='Customer Base'*expenditures=' '*sum=' '
 all='All Customers'*expenditures=' '*sum=' '
 / rts=25;
 format region regfmt. division divfmt. type usetype.;
 title 'Energy Expenditures for Each Region';
 title2 '(millions of dollars)';
run;
```

## Program Description

**Specify the table options.** The FORMAT= option specifies COMMA12. as the default format for the value in each table cell.

```
proc tabulate data=energy format=comma12.;
```

**Specify subgroups for the analysis.** The CLASS statement identifies Region, Division, and Type as class variables.

```
class region division type;
```

**Specify the analysis variable.** The VAR statement specifies that PROC TABULATE calculate statistics on the Expenditures variable.

```
var expenditures;

table region*(division all='Subtotal')
 all='Total for All Regions'*f=dollar12.,
 type='Customer Base'*expenditures=' '*sum=' '
 all='All Customers'*expenditures=' '*sum=' ';
```

**Specify the row title space.** RTS= provides 25 characters per line for row headings.

```
/ rts=25;
```

**Format the output.** The FORMAT statement assigns formats to the variables Region, Division, and Type.

```
format region regfmt. division divfmt. type usetype.;
```

**Specify the titles.**

```
title 'Energy Expenditures for Each Region';
title2 '(millions of dollars)';
run;
```

## Output

The universal class variable ALL provides subtotals and totals in this table.

**Output 61.21** Energy Expenditures for Each Region

| Energy Expenditures for Each Region<br>(millions of dollars) |                 |                       |                    |               |
|--------------------------------------------------------------|-----------------|-----------------------|--------------------|---------------|
|                                                              |                 | Customer Base         |                    | All Customers |
|                                                              |                 | Residential Customers | Business Customers |               |
| Region                                                       | Division        |                       |                    |               |
| Northeast                                                    | New England     | 7,477                 | 5,129              | 12,606        |
|                                                              | Middle Atlantic | 19,379                | 15,078             | 34,457        |
|                                                              | Subtotal        | 26,856                | 20,207             | 47,063        |
| West                                                         | Division        |                       |                    |               |
|                                                              | Mountain        | 5,476                 | 4,729              | 10,205        |
|                                                              | Pacific         | 13,959                | 12,619             | 26,578        |
|                                                              | Subtotal        | 19,435                | 17,348             | 36,783        |
| Total for All Regions                                        |                 | \$46,291              | \$37,555           | \$83,846      |



## Example 7: Eliminating Row Headings

|           |                                                                                                                                                                                                                   |
|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Features: | CLASS statement<br>PROC TABULATE statement options<br>DATA=<br>FORMAT=<br>TABLE statement options<br>crossing (*) operator<br>labels<br>ROW=FLOAT<br>RTS=<br>VAR statement<br>FORMAT statement<br>TITLE statement |
| Data set: | ENERGY                                                                                                                                                                                                            |
| Format:   | REGFMT.                                                                                                                                                                                                           |
| Format:   | DIVFMT.                                                                                                                                                                                                           |
| Format:   | USETYPE.                                                                                                                                                                                                          |

## Details

This example shows how to eliminate blank row headings from a table. To do so, you must both provide blank labels for the row headings and specify ROW=FLOAT in the TABLE statement.

## Program

```
proc tabulate data=energy format=dollar12.;
 class region division type;
 var expenditures;
 table region*division*expenditures=' '*sum=' ',
 type='Customer Base'
 / rts=25 row=float;
 format region regfmt. division divfmt. type usetype.;
 title 'Energy Expenditures for Each Region';
 title2 '(millions of dollars)';
run;
```

## Program Description

**Specify the table options.** The FORMAT= option specifies DOLLAR12. as the default format for the value in each table cell.

```
proc tabulate data=energy format=dollar12.;
```

**Specify subgroups for the analysis.** The CLASS statement identifies Region, Division, and Type as class variables.

```
class region division type;
```

**Specify the analysis variable.** The VAR statement specifies that PROC TABULATE calculate statistics on the Expenditures variable.

```
var expenditures;
```

**Define the table rows.** The row dimension of the TABLE statement creates a row for each formatted value of Region. Nested within these rows is a row for each formatted value of Division. The analysis variable Expenditures and the Sum statistic are also included in the row dimension, so PROC TABULATE creates row headings for them as well. The text in quotation marks specifies the headings for the corresponding variable or statistic. Although Sum is the default statistic, it is specified here so that you can specify a blank for its heading.

```
table region*division*expenditures=' '*sum=' ',
```

**Define the table columns.** The column dimension of the TABLE statement creates a column for each formatted value of Type.

```
type='Customer Base'
```

**Specify the row title space and eliminate blank row headings.** RTS= provides 25 characters per line for row headings. ROW=FLOAT eliminates blank row headings.

```
/ rts=25 row=float;
```

**Format the output.** The FORMAT statement assigns formats to the variables Region, Division, and Type.

```
format region regfmt. division divfmt. type usetype.;
```

**Specify the titles.**

```
title 'Energy Expenditures for Each Region';
title2 '(millions of dollars)';
run;
```

## Output

Compare this table with the output in [“Example 5: Customizing Row and Column Headings” on page 2318](#). The two tables are identical, but the program that creates this table uses Expenditures and Sum in the row dimension. PROC TABULATE automatically eliminates blank headings from the column dimension, whereas you must specify ROW=FLOAT to eliminate blank headings from the row dimension.

**Output 61.22** Energy Expenditures for Each Region

| Energy Expenditures for Each Region<br>(millions of dollars) |                 |                       |                    |
|--------------------------------------------------------------|-----------------|-----------------------|--------------------|
|                                                              |                 | Customer Base         |                    |
|                                                              |                 | Residential Customers | Business Customers |
| Region                                                       | Division        |                       |                    |
| Northeast                                                    | New England     | \$7,477               | \$5,129            |
|                                                              | Middle Atlantic | \$19,379              | \$15,078           |
| West                                                         | Mountain        | \$5,476               | \$4,729            |
|                                                              | Pacific         | \$13,959              | \$12,619           |

## Example 8: Creating Multipage Tables

|           |                                 |
|-----------|---------------------------------|
| Features: | CLASS statement                 |
|           | PROC TABULATE statement options |
|           | DATA=                           |
|           | FORMAT=                         |
|           | TABLE statement options         |
|           | ALL class variable              |
|           | BOX=                            |
|           | CONDENSE                        |
|           | INDENT=                         |
|           | page expression                 |
|           | RTS=                            |
| Data set: | VAR statement                   |
|           | FORMAT statement                |
|           | TITLE statement                 |
|           |                                 |
| Format:   | ENERGY                          |
| Format:   | REGFMT.                         |
| Format:   | DIVFMT.                         |
| Format:   | USETYPE.                        |

## Details

This example creates a separate table for each region and one table for all regions. By default, PROC TABULATE creates each table on a separate page, but the CONDENSE option places them all on the same page.

## Program

```
proc tabulate data=energy format=dollar12.;
 class region division type;
 var expenditures;
 table region='Region: ' all='All Regions',
 division all='All Divisions',
 type='Customer Base'*expenditures=' '*sum=' '
 / rts=25 box=_page_ condense indent=1;
 format region regfmt. division divfmt. type usetype.;
 title 'Energy Expenditures for Each Region and All Regions';
 title2 '(millions of dollars)';
run;
```

## Program Description

**Specify the table options.** The FORMAT= option specifies DOLLAR12. as the default format for the value in each table cell.

```
proc tabulate data=energy format=dollar12.;
```

**Specify subgroups for the analysis.** The CLASS statement identifies Region, Division, and Type as class variables.

```
class region division type;
```

**Specify the analysis variable.** The VAR statement specifies that PROC TABULATE calculate statistics on the Expenditures variable.

```
var expenditures;
```

**Define the table pages.** The page dimension of the TABLE statement creates one table for each formatted value of Region and one table for all regions. Text in quotation marks provides the heading for each page.

```
table region='Region: ' all='All Regions',
```

**Define the table rows.** The row dimension creates a row for each formatted value of Division and a row for all divisions. Text in quotation marks provides the row headings.

```
division all='All Divisions',
```

**Define the table columns.** The column dimension of the TABLE statement creates a column for each formatted value of Type. Each cell that is created by these pages, rows, and columns contains the sum of the analysis variable Expenditures for all observations that contribute to that cell. Text in quotation marks specifies headings for the corresponding variable or statistic. Although Sum is the default statistic, it is specified here so that you can specify a blank for its heading.

```
type='Customer Base'*expenditures=' '*sum=' ';
```

**Specify additional table options.** RTS= provides 25 characters per line for row headings. BOX= places the page heading inside the box above the row headings. CONDENSE places as many tables as possible on one physical page. INDENT= eliminates the row heading for Division. (Because there is no nesting in the row dimension, there is nothing to indent.)

```
/ rts=25 box=_page_ condense indent=1;
```

**Format the output.** The FORMAT statement assigns formats to the variables Region, Division, and Type.

```
format region regfmt. division divfmt. type usetype.;
```

**Specify the titles.**

```
title 'Energy Expenditures for Each Region and All Regions';
title2 '(millions of dollars)';
run;
```

## Output

**Output 61.23** *Energy Expenditures for Each Region and All Regions*

| Energy Expenditures for Each Region and All Regions<br>(millions of dollars) |                       |                    |
|------------------------------------------------------------------------------|-----------------------|--------------------|
| Region: Northeast                                                            | Customer Base         |                    |
|                                                                              | Residential Customers | Business Customers |
| New England                                                                  | \$7,477               | \$5,129            |
| Middle Atlantic                                                              | \$19,379              | \$15,078           |
| All Divisions                                                                | \$26,856              | \$20,207           |
|                                                                              |                       |                    |
| Region: West                                                                 | Customer Base         |                    |
|                                                                              | Residential Customers | Business Customers |
| Mountain                                                                     | \$5,476               | \$4,729            |
| Pacific                                                                      | \$13,959              | \$12,619           |
| All Divisions                                                                | \$19,435              | \$17,348           |
|                                                                              |                       |                    |
| All Regions                                                                  | Customer Base         |                    |
|                                                                              | Residential Customers | Business Customers |
| New England                                                                  | \$7,477               | \$5,129            |
| Middle Atlantic                                                              | \$19,379              | \$15,078           |
| Mountain                                                                     | \$5,476               | \$4,729            |
| Pacific                                                                      | \$13,959              | \$12,619           |
| All Divisions                                                                | \$46,291              | \$37,555           |

---

## Example 9: Reporting on Multiple-Response Survey Data

|           |                                                                                                                                                                                                                                                                                                                                      |
|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Features: | PROC TABULATE statement options<br>DATA=<br>TABLE statement<br>denominator definition (angle bracket operators)<br>N statistic<br>PCTN statistic<br>variable list<br>VAR statement<br>DATA step<br>FORMAT procedure<br>FOOTNOTE statement<br>OPTIONS statement options<br>FORMDLIM=<br>NONUMBER<br>SYMPUT routine<br>TITLE statement |
| Data set: | <a href="#">CUSTOMER_RESPONSE</a>                                                                                                                                                                                                                                                                                                    |

---

---

## Details

The two tables in this example show the following:

- which factors most influenced customers' decisions to buy products
- where customers heard of the company

The reports appear on one physical page with only one page number. By default, they would appear on separate pages.

In addition to showing how to create these tables, this example shows how to do the following:

- use a DATA step to count the number of observations in a data set
- store that value in a macro variable
- access that value later in the SAS session

The following figure shows the survey form that is used to collect data.

**Figure 61.8** Completed Survey Form

| Customer Questionnaire                              |                                                                                                                |
|-----------------------------------------------------|----------------------------------------------------------------------------------------------------------------|
| ID#                                                 | _____                                                                                                          |
| Please place a check beside all answers that apply. |                                                                                                                |
| Why do you buy our products?                        |                                                                                                                |
| <input type="checkbox"/> Cost                       | <input type="checkbox"/> Performance <input type="checkbox"/> Reliability <input type="checkbox"/> Sales staff |
| How did you find out about our company?             |                                                                                                                |
| <input type="checkbox"/> T.V./Radio                 | <input type="checkbox"/> Newspaper/Magazine <input type="checkbox"/> Word of mouth                             |
| What makes a sale person effective?                 |                                                                                                                |
| <input type="checkbox"/> Product knowledge          | <input type="checkbox"/> Personality <input type="checkbox"/> Appearance                                       |

---

## Program

```

data _null_;
 if 0 then set customer_response nobs=count;
 call symput('num',left(put(count,4.)));
 stop;
run;

proc format;
 picture pctfmt low-high='009.9 %';
run;

proc tabulate data=customer_response;
 var factor1-factor4 customer;

 table factor1='Cost'
 factor2='Performance'
 factor3='Reliability'
 factor4='Sales Staff',
 (n='Count'*f=7. ptn<customer>='Percent'*f=pctfmt9.) ;

 title 'Customer Survey Results: Spring 1996';
 title3 'Factors Influencing the Decision to Buy';
run;

proc tabulate
data=customer_response;
 var source1-source3 customer;

 table source1='TV/Radio'
 source2='Newspaper'
 source3='Word of Mouth',
 (n='Count'*f=7. ptn<customer>='Percent'*f=pctfmt9.) ;

 title 'Source of Company Name';
 footnote "Number of Respondents: &num";
run;

```

```
options formdlm='' number;
```

## Program Description

**Store the number of observations in a macro variable.** The SET statement reads the descriptor portion of CUSTOMER\_RESPONSE at compile time and stores the number of observations (the number of respondents) in COUNT. The SYMPUT routine stores the value of COUNT in the macro variable NUM. This variable is available for use by other procedures and DATA steps for the remainder of the SAS session. The IF 0 condition, which is always false, ensures that the SET statement, which reads the observations, never executes. (Reading observations is unnecessary.) The STOP statement ensures that the DATA step executes only once.

```
data _null_;
 if 0 then set customer_response nobs=count;
 call symput('num',left(put(count,4.)));
 stop;
run;
```

**Create the PCTFMT. format.** The FORMAT procedure creates a format for percentages. The PCTFMT. format writes all values with at least one digit to the left of the decimal point and with one digit to the right of the decimal point. A blank and a percent sign follow the digits.

```
proc format;
 picture pctfmt low-high='009.9 %';
run;
```

**Create the report and use the default table options.**

```
proc tabulate data=customer_response;
```

**Specify the analysis variables.** The VAR statement specifies that PROC TABULATE calculate statistics on the Factor1, Factor2, Factor3, Factor4, and Customer variables. The variable Customer must be listed because it is used to calculate the Percent column that is defined in the TABLE statement.

```
var factor1-factor4 customer;
```

**Define the table rows and columns.** The TABLE statement creates a row for each factor, a column for frequency counts, and a column for the percentages. Text in quotation marks supplies headings for the corresponding row or column. The format modifiers F=7. and F=PCTFMT9. provide formats for values in the associated cells and extend the column widths to accommodate the column headings.

```
table factor1='Cost'
 factor2='Performance'
 factor3='Reliability'
 factor4='Sales Staff',
 (n='Count'*f=7. pctl<customer>='Percent'*f=pctfmt9.) ;
```

**Specify the titles.**

```
title 'Customer Survey Results: Spring 1996';
title3 'Factors Influencing the Decision to Buy';
```



```
run;
```

**Create the report and use the default table options.**

```
proc tabulate
 data=customer_response;
```

**Specify the analysis variables.** The VAR statement specifies that PROC TABULATE calculate statistics on the Source1, Source2, Source3, and Customer variables. The variable Customer must be in the variable list because it appears in the denominator definition.

```
var source1-source3 customer;
```

**Define the table rows and columns.** The TABLE statement creates a row for each source of the company name, a column for frequency counts, and a column for the percentages. Text in quotation marks supplies a heading for the corresponding row or column.

```
table source1='TV/Radio'
 source2='Newspaper'
 source3='Word of Mouth',
 (n='Count'*f=7. ptn<customer>='Percent'*f=pctfmt9.) ;
```

**Specify the title and footnote.** The macro variable NUM resolves to the number of respondents. The FOOTNOTE statement uses double rather than single quotation marks so that the macro variable will resolve.

```
title 'Source of Company Name';
footnote "Number of Respondents: &num";
run;
```

**Reset the SAS system options.** The FORMDLIM= option resets the page delimiter to a page eject. The NUMBER option resumes the display of page numbers on subsequent pages.

```
options formdlim='' number;
```

## Output

**Output 61.24** Customer Survey Results: Spring 1996

| Customer Survey Results: Spring 1996    |       |         |
|-----------------------------------------|-------|---------|
| Factors Influencing the Decision to Buy |       |         |
|                                         | Count | Percent |
| Cost                                    | 87    | 72.5 %  |
| Performance                             | 62    | 51.6 %  |
| Reliability                             | 30    | 25.0 %  |
| Sales Staff                             | 120   | 100.0 % |

| Source of Company Name |       |         |
|------------------------|-------|---------|
|                        | Count | Percent |
| TV/Radio               | 92    | 76.6 %  |
| Newspaper              | 69    | 57.5 %  |
| Word of Mouth          | 26    | 21.6 %  |

Number of Respondents: 120

## Example 10: Reporting on Multiple-Choice Survey Data

Features:

- CLASS statement
- PROC TABULATE statement options
  - DATA=
  - FORMAT=
- TABLE statement
  - N statistic
- DATA step
- FORMAT procedure
- FORMAT statement
- TRANSPOSE procedure
- Data set options
  - RENAME=
- TITLE statement

Data set: **RADIO**

## Details

This report of listener preferences shows how many listeners select each type of programming during each of seven time periods on a typical weekday. The data was collected by a survey, and the results were stored in a SAS data set. Although this data set contains all the information needed for this report, the information is not arranged in a way that PROC TABULATE can use.

To make this crosstabulation of time of day and choice of radio programming, you must have a data set that contains a variable for time of day and a variable for programming preference. PROC TRANSPOSE reshapes the data into a new data set that contains these variables. Once the data are in the appropriate form, PROC TABULATE creates the report.

The following figure shows the survey form that is used to collect data.

**Figure 61.9** Completed Survey Form

| LISTENER SURVEY                                                               |                                 | phone. --                                                                 |
|-------------------------------------------------------------------------------|---------------------------------|---------------------------------------------------------------------------|
| 1. ____What is your age?                                                      |                                 |                                                                           |
| 2. ____What is your gender?                                                   |                                 |                                                                           |
| 3. ____On the average WEEKDAY, how many hours do you listen to the radio?     |                                 |                                                                           |
| 4. ____On the average WEEKEND-DAY, how many hours do you listen to the radio? |                                 |                                                                           |
| Use codes 1-8 for questions 5. Use codes 0-8 for 6-19.                        |                                 |                                                                           |
| 0 Do not listen at that time                                                  |                                 |                                                                           |
| 1 Rock                                                                        | 5 Classical                     |                                                                           |
| 2 Top 40                                                                      | 6 Easy Listening                |                                                                           |
| 3 Country                                                                     | 7 News/Information/Talk         |                                                                           |
| 4 Jazz                                                                        | 8 Other                         |                                                                           |
| 5. ____What style of music or radio programming do you most often listen to?  |                                 |                                                                           |
| On a typical WEEKDAY, what kind of radio programming do you listen to         |                                 | On a typical WEEKEND-DAY, what kind of radio programming do you listen to |
| 6. ____from 6-9 a.m.?                                                         | 13. ____from 6-9 a.m.?          |                                                                           |
| 7. ____from 9 a.m. to noon?                                                   | 14. ____from 9 a.m. to noon?    |                                                                           |
| 8. ____from noon to 1 p.m.?                                                   | 15. ____from noon to 1 p.m.?    |                                                                           |
| 9. ____from 1-4 p.m.?                                                         | 16. ____from 1-4 p.m.?          |                                                                           |
| 10. ____from 4-6 p.m.?                                                        | 17. ____from 4-6 p.m.?          |                                                                           |
| 11. ____from 6-10 p.m.?                                                       | 18. ____from 6-10 p.m.?         |                                                                           |
| 12. ____from 10 p.m. to 2 a.m.?                                               | 19. ____from 10 p.m. to 2 a.m.? |                                                                           |

An [external file on page 2535](#) contains the raw data for the survey. Several lines from that file appear here.

```

967 32 f 5 3 5
7 5 5 5 7 0 0 0 8 7 0 0 8 0
781 30 f 2 3 5
5 0 0 0 5 0 0 0 4 7 5 0 0 0
859 39 f 1 0 5
1 0 0 0 1 0 0 0 0 0 0 0 0 0

... more data lines ...

859 32 m .25 .25 1
1 0 0 0 0 0 0 0 1 0 0 0 0 0

```

## Program

```

data radio;
 infile 'input-file' missover;

 input /(Time1-Time7) ($1. +1);
 listener=_n_;
run;

proc format;
 value $timefmt 'Time1'='6-9 a.m.'
 'Time2'='9 a.m. to noon'
 'Time3'='noon to 1 p.m.'
 'Time4'='1-4 p.m.'
 'Time5'='4-6 p.m.'
 'Time6'='6-10 p.m.'
 'Time7'='10 p.m. to 2 a.m.'
 other='*** Data Entry Error ***';
 value $pgmfmt '0'='Don't Listen'
 '1','2'='Rock and Top 40'
 '3'='Country'
 '4','5','6'='Jazz, Classical, and Easy Listening'
 '7'='News/ Information /Talk'
 '8'='Other'
 other='*** Data Entry Error ***';
run;

proc transpose data=radio
 out=radio_transposed(rename=(coll=Choice))
 name=Timespan;
 by listener;
 var time1-time7;
run;

proc tabulate data=radio_transposed format=12.;
format timespan $timefmt. choice $pgmfmt.;

class timespan choice;

table timespan='Time of Day',
 choice='Choice of Radio Program'*n='Number of Listeners';

```

```

 title 'Listening Preferences on Weekdays';
run;

```

## Program Description

**Create the RADIO data set and specify the input file.** RADIO contains data from a survey of 336 listeners. The data set contains information about listeners and their preferences in radio programming. The INFILE statement specifies the external file that contains the data. MISSOVER prevents the input pointer from going to the next record if it fails to find values in the current line for all variables that are listed in the INPUT statement.

```

data radio;
 infile 'input-file' missover;

 input /(Time1-Time7) ($1. +1);
 listener=_n_;
run;

```

**Create the \$TIMEFMT. and \$PGMFMT. formats.** PROC FORMAT creates formats for the time of day and the choice of programming.

```

proc format;
 value $timefmt 'Time1'='6-9 a.m.'
 'Time2'='9 a.m. to noon'
 'Time3'='noon to 1 p.m.'
 'Time4'='1-4 p.m.'
 'Time5'='4-6 p.m.'
 'Time6'='6-10 p.m.'
 'Time7'='10 p.m. to 2 a.m.'
 other='*** Data Entry Error ***';

 value $pgmfmt '0'='Don't Listen'
 '1','2'='Rock and Top 40'
 '3'='Country'
 '4','5','6'='Jazz, Classical, and Easy Listening'
 '7'='News/ Information /Talk'
 '8'='Other'
 other='*** Data Entry Error ***';

run;

```

**Reshape the data by transposing the RADIO data set.** PROC TRANSPOSE creates RADIO\_TRANSPOSED. This data set contains the variable Listener from the original data set. It also contains two transposed variables: Timespan and Choice. Timespan contains the names of the variables (Time1-Time7) from the input data set that are transposed to form observations in the output data set. Choice contains the values of these variables. (See [“Details” on page 2337](#) for a complete explanation of the PROC TRANSPOSE step.)

```

proc transpose data=radio
 out=radio_transposed(rename=(coll=Choice))
 name=Timespan;
 by listener;
 var time1-time7;
run;

```

**Create the report and specify the table options.** The FORMAT= option specifies the default format for the values in each table cell.

```
proc tabulate data=radio_transposed format=12.;
```

**Format the transposed variables.** The FORMAT statement permanently associates these formats with the variables in the output data set.

```
format timespan $timefmt. choice $pgmfmt.;
```

**Specify subgroups for the analysis.** The CLASS statement identifies Timespan and Choice as class variables.

```
class timespan choice;
```

**Define the table rows and columns.** The TABLE statement creates a row for each formatted value of Timespan and a column for each formatted value of Choice. In each column are values for the N statistic. Text in quotation marks supplies headings for the corresponding rows or columns.

```
table timespan='Time of Day',
 choice='Choice of Radio Program'*n='Number of Listeners';
```

**Specify the title.**

```
title 'Listening Preferences on Weekdays';
run;
```

## Output

**Output 61.25** *Listening Preferences on Weekdays*

| Listening Preferences on Weekdays |                         |                     |                     |                                        |                            |                     |
|-----------------------------------|-------------------------|---------------------|---------------------|----------------------------------------|----------------------------|---------------------|
|                                   | Choice of Radio Program |                     |                     |                                        |                            |                     |
|                                   | Don't Listen            | Rock and Top 40     | Country             | Jazz, Classical,<br>and Easy Listening | News/ Information<br>/Talk | Other               |
|                                   | Number of Listeners     | Number of Listeners | Number of Listeners | Number of Listeners                    | Number of Listeners        | Number of Listeners |
| Time of Day                       |                         |                     |                     |                                        |                            |                     |
| 6-9 a.m.                          | 34                      | 143                 | 7                   | 39                                     | 96                         | 17                  |
| 9 a.m. to noon                    | 214                     | 59                  | 5                   | 51                                     | 3                          | 4                   |
| noon to 1 p.m.                    | 238                     | 55                  | 3                   | 27                                     | 9                          | 4                   |
| 1-4 p.m.                          | 216                     | 60                  | 5                   | 50                                     | 2                          | 3                   |
| 4-6 p.m.                          | 56                      | 130                 | 6                   | 57                                     | 69                         | 18                  |
| 6-10 p.m.                         | 202                     | 54                  | 9                   | 44                                     | 20                         | 7                   |
| 10 p.m. to 2 a.m.                 | 264                     | 29                  | 3                   | 36                                     | 2                          | 2                   |

---

## Details

---

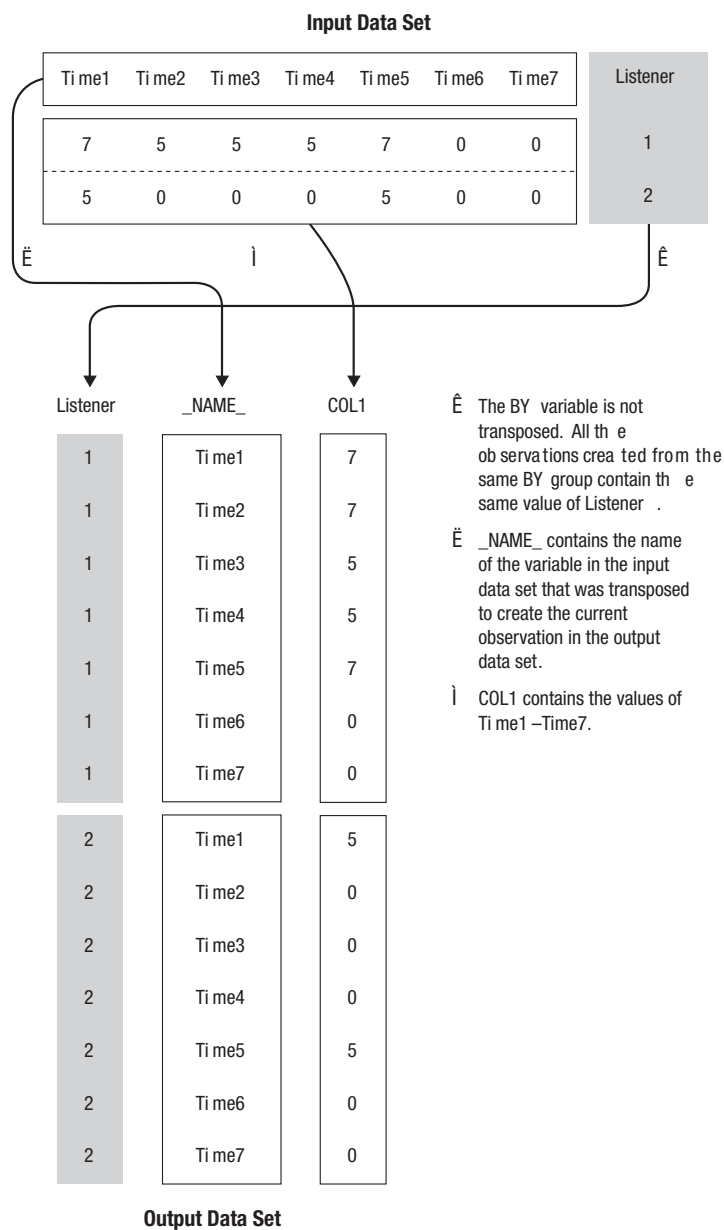
### Reshape the Data

The original input data set has all the information that you need to make the crosstabular report, but PROC TABULATE cannot use the information in that form. PROC TRANSPOSE rearranges the data so that each observation in the new data set contains the variable Listener, a variable for time of day, and a variable for programming preference. The following figure illustrates the transposition. PROC TABULATE uses this new data set to create the crosstabular report.

PROC TRANSPOSE restructures data so that values that were stored in one observation are written to one variable. You can specify which variables you want to transpose.

When you transpose with BY processing, as this example does, you create from each BY group one observation for each variable that you transpose. In this example, Listener is the BY variable. Each observation in the input data set is a BY group because the value of Listener is unique for each observation.

This example transposes seven variables, Time1 through Time7. Therefore, the output data set has seven observations from each BY group (each observation) in the input data set.

**Figure 61.10** Transposing Two Observations

## Understanding the PROC TRANSPOSE Step

Here is a detailed explanation of the PROC TRANSPOSE step that reshapes the data:

```

proc transpose data=radio 1
 out=radio_transposed(rename=(col1=Choice)) 2
 name=Timespan; 3
 by listener; 4
 var time1-time7; 5
 format timespan $timefmt. choice $pgmfmt.; 6

```



```
run;
```

- 1 The DATA= option specifies the input data set.
- 2 The OUT= option specifies the output data set. The RENAME= data set option renames the transposed variable from COL1 (the default name) to Choice.
- 3 The NAME= option specifies the name for the variable in the output data set that contains the name of the variable that is being transposed to create the current observation. By default, the name of this variable is \_NAME\_.
- 4 The BY statement identifies Listener as the BY variable.
- 5 The VAR statement identifies Time1 through Time7 as the variables to transpose.
- 6 The FORMAT statement assigns formats to Timespan and Choice. The PROC TABULATE step that creates the report does not need to format Timespan and Choice because the formats are stored with these variables.

---

## Example 11: Calculating Various Percentage Statistics

Features:

CLASS statement  
 PROC TABULATE statement options  
   FORMAT=  
 TABLE statement options  
   ALL class variable  
   COLPCTSUM statistic  
   concatenation (blank) operator  
   crossing (\*) operator  
   format modifiers  
   grouping elements (parentheses) operator  
   labels  
   REPPCTSUM statistic  
   ROWPCTSUM statistic  
   variable list  
   ROW=FLOAT  
   RTS=  
 VAR statement  
 DATA step  
 FORMAT procedure  
 TITLE statement

---

## Details

This example shows how to use three percentage sum statistics: COLPCTSUM, REPPCTSUM, and ROWPCTSUM.

### Program

```
data fundrais;
 length name $ 8 classrm $ 1;
 input @1 team $ @8 classrm $ @10 name $
 @19 pencils @23 tablets;
 sales=pencils + tablets;
 datalines;
BLUE A ANN 4 8
RED A MARY 5 10
GREEN A JOHN 6 4
RED A BOB 2 3
BLUE B FRED 6 8
GREEN B LOUISE 12 2
BLUE B ANNETTE . 9
RED B HENRY 8 10
GREEN A ANDREW 3 5
RED A SAMUEL 12 10
BLUE A LINDA 7 12
GREEN A SARA 4 .
BLUE B MARTIN 9 13
RED B MATTHEW 7 6
GREEN B BETH 15 10
RED B LAURA 4 3
;

proc format;
 picture pctfmt low-high='009 %';
run;

title "Fundraiser Sales";

proc tabulate format=7.;
 class team classrm;
 var sales;
 table (team all),
 classrm='Classroom'*sales=' '* (sum
 colpctsum*f=pctfmt9.
 rowpctsum*f=pctfmt9.
 reppctsum*f=pctfmt9.)
 all*sales*sum=' '
 /rts=20;
run;
```

## Program Description

**Create the FUNDRAIS data set.** FUNDRAIS contains data on student sales during a school fund-raiser. A DATA step creates the data set.

```
data fundrais;
 length name $ 8 classrm $ 1;
 input @1 team $ @8 classrm $ @10 name $
 @19 pencils @23 tablets;
 sales=pencils + tablets;
 datalines;
BLUE A ANN 4 8
RED A MARY 5 10
GREEN A JOHN 6 4
RED A BOB 2 3
BLUE B FRED 6 8
GREEN B LOUISE 12 2
BLUE B ANNETTE . 9
RED B HENRY 8 10
GREEN A ANDREW 3 5
RED A SAMUEL 12 10
BLUE A LINDA 7 12
GREEN A SARA 4 .
BLUE B MARTIN 9 13
RED B MATTHEW 7 6
GREEN B BETH 15 10
RED B LAURA 4 3
;
```

**Create the PCTFMT. format.** The FORMAT procedure creates a format for percentages. The PCTFMT. format writes all values with at least one digit, a blank, and a percent sign.

```
proc format;
 picture pctfmt low-high='009 %';
run;
```

**Specify the title.**

```
title "Fundraiser Sales";
```

**Create the report and specify the table options.** The FORMAT= option specifies up to seven digits as the default format for the value in each table cell.

```
proc tabulate format=7.;
```

**Specify subgroups for the analysis.** The CLASS statement identifies Team and Classrm as class variables.

```
class team classrm;
```

**Specify the analysis variable.** The VAR statement specifies that PROC TABULATE calculate statistics on the Sales variable.

```
var sales;
```

**Define the table rows.** The row dimension of the TABLE statement creates a row for each formatted value of Team. The last row of the report summarizes sales for all teams.

```
table (team all),
```

**Define the table columns.** The column dimension of the TABLE statement creates a column for each formatted value of `Classrm`. Crossed within each value of `Classrm` is the analysis variable (`sales`) with a blank label. Nested within each column are columns that summarize sales for the class. The first nested column, labeled `sum`, is the sum of sales for the row for the classroom. The second nested column, labeled `ColPctSum`, is the percentage of the sum of sales for the row for the classroom in relation to the sum of sales for all teams in the classroom. The third nested column, labeled `RowPctSum`, is the percentage of the sum of sales for the row for the classroom in relation to the sum of sales for the row for all classrooms. The fourth nested column, labeled `RepPctSum`, is the percentage of the sum of sales for the row for the classroom in relation to the sum of sales for all teams for all classrooms. The last column of the report summarizes sales for the row for all classrooms.

```
classrm='Classroom'*sales=' '(sum
colpctsum*f=pctfmt9.
rowpctsum*f=pctfmt9.
reppctsum*f=pctfmt9.)
all*sales*sum=' '
```

**Specify the row title space and eliminate blank row headings.** `RTS=` provides 20 characters per line for row headings.

```
/rts=20;
run;
```

## Output

**Output 61.26** Fundraiser Sales

| Fundraiser Sales |           |           |           |           |     |           |           |           |       |
|------------------|-----------|-----------|-----------|-----------|-----|-----------|-----------|-----------|-------|
| team             | Classroom |           |           |           |     |           |           |           | All   |
|                  | A         |           |           |           | B   |           |           |           | sales |
|                  | Sum       | ColPctSum | RowPctSum | RepPctSum | Sum | ColPctSum | RowPctSum | RepPctSum |       |
| BLUE             | 31        | 34 %      | 46 %      | 15 %      | 36  | 31 %      | 53 %      | 17 %      | 67    |
| GREEN            | 18        | 19 %      | 31 %      | 8 %       | 39  | 34 %      | 68 %      | 19 %      | 57    |
| RED              | 42        | 46 %      | 52 %      | 20 %      | 38  | 33 %      | 47 %      | 18 %      | 80    |
| All              | 91        | 100 %     | 44 %      | 44 %      | 113 | 100 %     | 55 %      | 55 %      | 204   |

## Details

Here are the percentage sum statistic calculations used to produce the output for the Blue Team in Classroom A:

- $\text{COLPCTSUM} = 31/91 * 100 = 34\%$
- $\text{ROWPCTSUM} = 31/67 * 100 = 46\%$
- $\text{REPPCTSUM} = 31/204 * 100 = 15\%$

Similar calculations were used to produce the output for the remaining teams and classrooms.

---

## Example 12: Using Denominator Definitions to Display Basic Frequency Counts and Percentages

Features:

- CLASS statement
- PROC TABULATE statement options
  - DATA=
  - FORMAT=
- TABLE statement options
  - ALL class variable
  - denominator definitions (angle bracket operators)
  - N statistic
  - PCTN statistic
  - RTS=
- DATA step
- FORMAT procedure
- FORMAT statement
- TITLE statement

---

---

## Details

Crosstabulation tables (also called contingency tables or stub-and-banner reports) show combined frequency distributions for two or more variables. This table shows frequency counts for females and males within each of four job classes. The table also shows the percentage that each frequency count represents the following:

- the total women and men in that job class (row percentage)
- the total for that gender in all job classes (column percentage)
- the total for all employees

---

## Program

```
data jobclass;
 input Gender Occupation @@;
```

```

 datalines;
1 1 1 1 1 1 1 1 1 1
1 2 1 2 1 2 1 2 1 2
1 3 1 3 1 3 1 3 1 3
1 1 1 1 1 1 1 2 1 2
1 2 1 2 1 3 1 3 1 4
1 4 1 4 1 4 1 1 1 1
1 1 1 2 1 2 1 2 1 2
1 2 1 3 1 3 1 3 1 4
1 4 1 4 1 4 1 1 1 3
2 1 2 1 2 1 2 1 2 2
2 2 2 2 2 2 2 3 2 3
2 4 2 4 2 4 2 4 2 1
2 3 2 3 2 3 2 3 2 4
2 4 2 4 2 1 2 1 2 1
2 2 2 2 2 2 2 2 2 2
2 3 2 3 2 4 2 4 2 1
2 1 2 1 2 1 2 2 2 2
2 3 2 3 2 3 2 4
;

proc format;
 value gendfmt 1='Female'
 2='Male'
 other='*** Data Entry Error ***';
 value occupfmt 1='Technical'
 2='Manager/Supervisor'
 3='Clerical'
 4='Administrative'
 other='*** Data Entry Error ***';
run;

proc tabulate data=jobclass format=8.2;
 class gender occupation;
 table (occupation='Job Class' all='All Jobs')
 *(n='Number of employees'*f=9.
 pctn<gender all>='Percent of row total'
 pctn<occupation all>='Percent of column total'
 pctn='Percent of total'),
 gender='Gender' all='All Employees'/ rts=50;
 format gender gendfmt. occupation occupfmt.;
 title 'Gender Distribution';
 title2 'within Job Classes';
run;

```

---

## Program Description

**Create the JOBCLASS data set.** JOBCLASS contains encoded information about the gender and job class of employees at a fictitious company.

```

data jobclass;
 input Gender Occupation @@;
 datalines;

```

```

1 1 1 1 1 1 1 1 1 1
1 2 1 2 1 2 1 2 1 2
1 3 1 3 1 3 1 3 1 3
1 1 1 1 1 1 1 2 1 2
1 2 1 2 1 3 1 3 1 4 1 4
1 4 1 4 1 4 1 1 1 1 1 1
1 1 1 2 1 2 1 2 1 2 1 2
1 2 1 3 1 3 1 3 1 3 1 4 1 4
1 4 1 4 1 4 1 1 1 3 2 1 2 1
2 1 2 1 2 1 2 1 2 2 2 2
2 2 2 2 2 2 2 3 2 3 2 3 2 4
2 4 2 4 2 4 2 4 2 4 2 1 2 3
2 3 2 3 2 3 2 3 2 4 2 4 2 4
2 4 2 4 2 1 2 1 2 1 2 1 2 1
2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 3 2 3 2 4 2 4 2 4 2 1 2 1
2 1 2 1 2 1 2 2 2 2 2 2 2 3
2 3 2 3 2 3 2 4
;

```

**Create the GENDFMT. and OCCUPFMT. formats.** PROC FORMAT creates formats for the variables Gender and Occupation.

```

proc format;
 value gendfmt 1='Female'
 2='Male'
 other='*** Data Entry Error ***';
 value occupfmt 1='Technical'
 2='Manager/Supervisor'
 3='Clerical'
 4='Administrative'
 other='*** Data Entry Error ***';
run;

```

**Create the report and specify the table options.** The FORMAT= option specifies the 8.2 format as the default format for the value in each table cell.

```
proc tabulate data=jobclass format=8.2;
```

**Specify subgroups for the analysis.** The CLASS statement identifies Gender and Occupation as class variables.

```

class gender occupation;

table (occupation='Job Class' all='All Jobs')
 *(n='Number of employees'*f=9.
 pctn<gender all>='Percent of row total'
 pctn<occupation all>='Percent of column total'
 pctn='Percent of total'),

```

**Define the table columns and specify the amount of space for row headings.** The column dimension creates a column for each formatted value of Gender and for all employees. Text in quotation marks supplies the heading for the corresponding column. The RTS= option provides 50 characters per line for row headings.

```
gender='Gender' all='All Employees'/ rts=50;
```

**Format the output.** The FORMAT statement assigns formats to the variables Gender and Occupation.

```
format gender gendfmt. occupation occupfmt.;
```

**Specify the titles.**

```

title 'Gender Distribution';
title2 'within Job Classes';
run;

```

## Output

**Output 61.27** *Gender Distribution within Job Classes*

| Gender Distribution<br>within Job Classes |                         |        |        |               |
|-------------------------------------------|-------------------------|--------|--------|---------------|
|                                           |                         | Gender |        | All Employees |
|                                           |                         | Female | Male   |               |
| Job Class                                 |                         |        |        |               |
| Technical                                 | Number of employees     | 16     | 18     | 34            |
|                                           | Percent of row total    | 47.06  | 52.94  | 100.00        |
|                                           | Percent of column total | 26.23  | 29.03  | 27.64         |
|                                           | Percent of total        | 13.01  | 14.63  | 27.64         |
| Manager/Supervisor                        | Number of employees     | 20     | 15     | 35            |
|                                           | Percent of row total    | 57.14  | 42.86  | 100.00        |
|                                           | Percent of column total | 32.79  | 24.19  | 28.46         |
|                                           | Percent of total        | 16.26  | 12.20  | 28.46         |
| Clerical                                  | Number of employees     | 14     | 14     | 28            |
|                                           | Percent of row total    | 50.00  | 50.00  | 100.00        |
|                                           | Percent of column total | 22.95  | 22.58  | 22.76         |
|                                           | Percent of total        | 11.38  | 11.38  | 22.76         |
| Administrative                            | Number of employees     | 11     | 15     | 26            |
|                                           | Percent of row total    | 42.31  | 57.69  | 100.00        |
|                                           | Percent of column total | 18.03  | 24.19  | 21.14         |
|                                           | Percent of total        | 8.94   | 12.20  | 21.14         |
| All Jobs                                  | Number of employees     | 61     | 62     | 123           |
|                                           | Percent of row total    | 49.59  | 50.41  | 100.00        |
|                                           | Percent of column total | 100.00 | 100.00 | 100.00        |
|                                           | Percent of total        | 49.59  | 50.41  | 100.00        |



# Details

## Overview

The part of the TABLE statement that defines the rows of the table uses the PCTN statistic to calculate three different percentages.

In all calculations of PCTN, the numerator is N, the frequency count for one cell of the table. The denominator for each occurrence of PCTN is determined by the denominator definition. The denominator definition appears in angle brackets after the keyword PCTN. It is a list of one or more expressions. The list tells PROC TABULATE which frequency counts to sum for the denominator.

## Analyzing the Structure of the Table

Taking a close look at the structure of the table helps you understand how PROC TABULATE uses the denominator definitions. The following simplified version of the TABLE statement clarifies the basic structure of the table:

```
table occupation='Job Class' all='All Jobs',
 gender='Gender' all='All Employees';
```

The table is a concatenation of four subtables. In this report, each subtable is a crossing of one class variable in the row dimension and one class variable in the column dimension. Each crossing establishes one or more categories. A category is a combination of unique values of class variables, such as female, technical or all, clerical

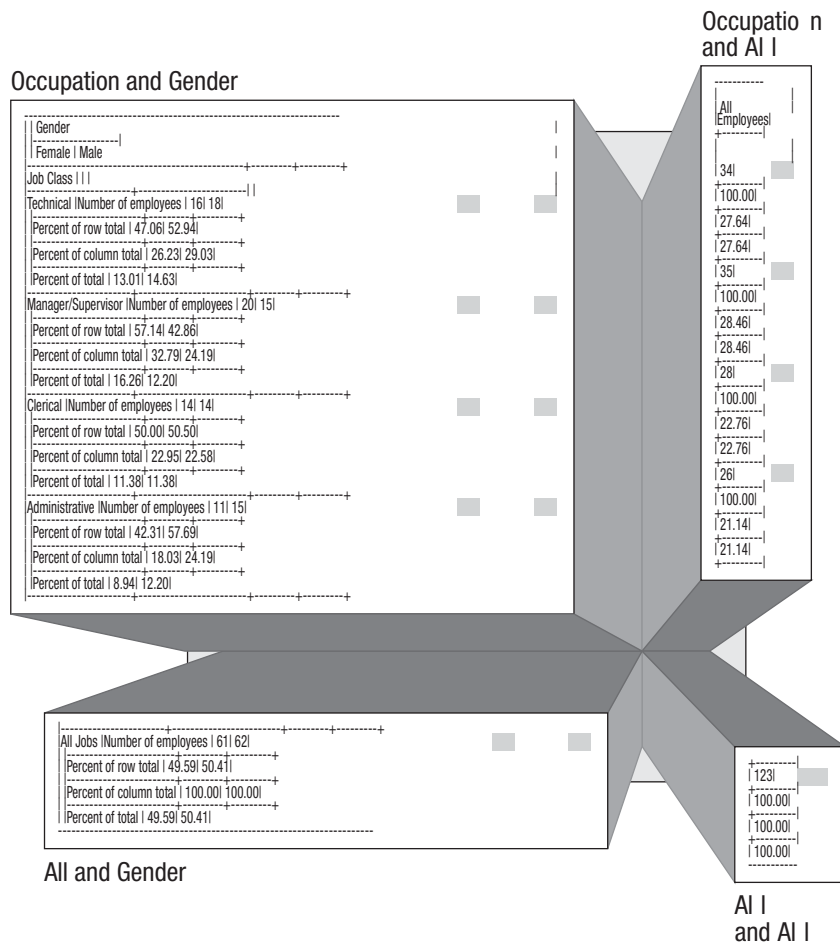
The following table describes each subtable:

**Table 61.9** Contents of Subtables

| Class Variables Contributing to the Subtable | Description of Frequency Counts                              | Number of Categories |
|----------------------------------------------|--------------------------------------------------------------|----------------------|
| Occupation and Gender                        | Number of females in each job or number of males in each job | 8                    |
| All and Gender                               | Number of females or number of males                         | 2                    |
| Occupation and All                           | Number of people in each job                                 | 4                    |
| All and All                                  | Number of people in all jobs                                 | 1                    |

The following figure highlights these subtables and the frequency counts for each category.

**Figure 61.11** Illustration of the Four Subtables



## Interpreting Denominator Definitions

The following fragment of the TABLE statement defines the denominator definitions for this report. The PCTN keyword and the denominator definitions are highlighted.

```
table (occupation='Job Class' all='All Jobs')
 *(n='Number of employees'*f=5.
 pctn<gender all>='Row percent'
 pctn<occupation all>='Column percent'
 pctn='Percent of total'),
```

Each use of PCTN nests a row of statistics within each value of Occupation and All. Each denominator definition tells PROC TABULATE which frequency counts to sum for the denominators in that row. This section explains how PROC TABULATE interprets these denominator definitions.

## Row Percentages

The part of the TABLE statement that calculates the row percentages and that labels the row is

```
pctn<gender all>='Row percent '
```

Consider how PROC TABULATE interprets this denominator definition for each subtable.

**Output 61.28** Subtable 1: Occupation and Gender

| Gender Distribution<br>within Job Classes |                         |        |        |               |
|-------------------------------------------|-------------------------|--------|--------|---------------|
| Job Class                                 |                         | Gender |        | All Employees |
|                                           |                         | Female | Male   |               |
| Technical                                 | Number of employees     | 16     | 18     | 34            |
|                                           | Percent of row total    | 47.06  | 52.94  | 100.00        |
|                                           | Percent of column total | 26.23  | 29.03  | 27.64         |
|                                           | Percent of total        | 13.01  | 14.63  | 27.64         |
| Manager/Supervisor                        | Number of employees     | 20     | 15     | 35            |
|                                           | Percent of row total    | 57.14  | 42.86  | 100.00        |
|                                           | Percent of column total | 32.79  | 24.19  | 28.46         |
|                                           | Percent of total        | 16.26  | 12.20  | 28.46         |
| Clerical                                  | Number of employees     | 14     | 14     | 28            |
|                                           | Percent of row total    | 50.00  | 50.00  | 100.00        |
|                                           | Percent of column total | 22.95  | 22.58  | 22.76         |
|                                           | Percent of total        | 11.38  | 11.38  | 22.76         |
| Administrative                            | Number of employees     | 11     | 15     | 26            |
|                                           | Percent of row total    | 42.31  | 57.69  | 100.00        |
|                                           | Percent of column total | 18.03  | 24.19  | 21.14         |
|                                           | Percent of total        | 8.94   | 12.20  | 21.14         |
| All Jobs                                  | Number of employees     | 61     | 62     | 123           |
|                                           | Percent of row total    | 49.59  | 50.41  | 100.00        |
|                                           | Percent of column total | 100.00 | 100.00 | 100.00        |
|                                           | Percent of total        | 49.59  | 50.41  | 100.00        |

PROC TABULATE looks at the first element in the denominator definition, Gender, and asks whether Gender contributes to the subtable. Because Gender does contribute to the subtable, PROC TABULATE uses it as the denominator definition. This denominator definition tells PROC TABULATE to sum the frequency counts for all occurrences of Gender within the same value of Occupation.

For example, the denominator for the category `female, technical` is the sum of all frequency counts for all categories in this subtable for which the value of `Occupation` is `technical`. There are two such categories: `female, technical` and `male, technical`. The corresponding frequency counts are 16 and 18. Therefore, the denominator for this category is 16+18, or 34.

**Output 61.29** Subtable 2: All and Gender

| Gender Distribution<br>within Job Classes |                         |        |        |               |
|-------------------------------------------|-------------------------|--------|--------|---------------|
|                                           |                         | Gender |        | All Employees |
|                                           |                         | Female | Male   |               |
| Job Class                                 |                         |        |        |               |
| Technical                                 | Number of employees     | 16     | 18     | 34            |
|                                           | Percent of row total    | 47.06  | 52.94  | 100.00        |
|                                           | Percent of column total | 26.23  | 29.03  | 27.64         |
|                                           | Percent of total        | 13.01  | 14.63  | 27.64         |
| Manager/Supervisor                        | Number of employees     | 20     | 15     | 35            |
|                                           | Percent of row total    | 57.14  | 42.86  | 100.00        |
|                                           | Percent of column total | 32.79  | 24.19  | 28.46         |
|                                           | Percent of total        | 16.26  | 12.20  | 28.46         |
| Clerical                                  | Number of employees     | 14     | 14     | 28            |
|                                           | Percent of row total    | 50.00  | 50.00  | 100.00        |
|                                           | Percent of column total | 22.95  | 22.58  | 22.76         |
|                                           | Percent of total        | 11.38  | 11.38  | 22.76         |
| Administrative                            | Number of employees     | 11     | 15     | 26            |
|                                           | Percent of row total    | 42.31  | 57.69  | 100.00        |
|                                           | Percent of column total | 18.03  | 24.19  | 21.14         |
|                                           | Percent of total        | 8.94   | 12.20  | 21.14         |
| All Jobs                                  | Number of employees     | 61     | 62     | 123           |
|                                           | Percent of row total    | 49.59  | 50.41  | 100.00        |
|                                           | Percent of column total | 100.00 | 100.00 | 100.00        |
|                                           | Percent of total        | 49.59  | 50.41  | 100.00        |

PROC TABULATE looks at the first element in the denominator definition, `Gender`, and asks whether `Gender` contributes to the subtable. Because `Gender` does contribute to the subtable, PROC TABULATE uses it as the denominator definition. This denominator definition tells PROC TABULATE to sum the frequency counts for all occurrences of `Gender` in the subtable.

For example, the denominator for the category `all, female` is the sum of the frequency counts for `all, female` and `all, male`. The corresponding frequency counts are 61 and 62. Therefore, the denominator for cells in this subtable is 61+62, or 123.

**Output 61.30** Subtable 3: Occupation and All

|                    |                         | Gender |        | All Employees |
|--------------------|-------------------------|--------|--------|---------------|
|                    |                         | Female | Male   |               |
| Job Class          |                         |        |        |               |
| Technical          | Number of employees     | 16     | 18     | 34            |
|                    | Percent of row total    | 47.06  | 52.94  | 100.00        |
|                    | Percent of column total | 26.23  | 29.03  | 27.64         |
|                    | Percent of total        | 13.01  | 14.63  | 27.64         |
| Manager/Supervisor | Number of employees     | 20     | 15     | 35            |
|                    | Percent of row total    | 57.14  | 42.86  | 100.00        |
|                    | Percent of column total | 32.79  | 24.19  | 28.46         |
|                    | Percent of total        | 16.26  | 12.20  | 28.46         |
| Clerical           | Number of employees     | 14     | 14     | 28            |
|                    | Percent of row total    | 50.00  | 50.00  | 100.00        |
|                    | Percent of column total | 22.95  | 22.58  | 22.76         |
|                    | Percent of total        | 11.38  | 11.38  | 22.76         |
| Administrative     | Number of employees     | 11     | 15     | 26            |
|                    | Percent of row total    | 42.31  | 57.69  | 100.00        |
|                    | Percent of column total | 18.03  | 24.19  | 21.14         |
|                    | Percent of total        | 8.94   | 12.20  | 21.14         |
| All Jobs           | Number of employees     | 61     | 62     | 123           |
|                    | Percent of row total    | 49.59  | 50.41  | 100.00        |
|                    | Percent of column total | 100.00 | 100.00 | 100.00        |
|                    | Percent of total        | 49.59  | 50.41  | 100.00        |

PROC TABULATE looks at the first element in the denominator definition, Gender, and asks whether Gender contributes to the subtable. Because Gender does not contribute to the subtable, PROC TABULATE looks at the next element in the denominator definition, which is All. The variable All does contribute to this subtable, so PROC TABULATE uses it as the denominator definition. All is a reserved class variable with only one category. Therefore, this denominator definition tells PROC TABULATE to use the frequency count of All as the denominator.

For example, the denominator for the category `clerical`, `all` is the frequency count for that category, 28.

**Note:** In these table cells, because the numerator and the denominator are the same, the row percentages in this subtable are all 100.

**Output 61.31** Subtable 4: All and All

| Gender Distribution<br>within Job Classes |                         |        |        |               |
|-------------------------------------------|-------------------------|--------|--------|---------------|
|                                           |                         | Gender |        | All Employees |
|                                           |                         | Female | Male   |               |
| Job Class                                 |                         |        |        |               |
| Technical                                 | Number of employees     | 16     | 18     | 34            |
|                                           | Percent of row total    | 47.06  | 52.94  | 100.00        |
|                                           | Percent of column total | 26.23  | 29.03  | 27.64         |
|                                           | Percent of total        | 13.01  | 14.63  | 27.64         |
| Manager/Supervisor                        | Number of employees     | 20     | 15     | 35            |
|                                           | Percent of row total    | 57.14  | 42.86  | 100.00        |
|                                           | Percent of column total | 32.79  | 24.19  | 28.46         |
|                                           | Percent of total        | 16.26  | 12.20  | 28.46         |
| Clerical                                  | Number of employees     | 14     | 14     | 28            |
|                                           | Percent of row total    | 50.00  | 50.00  | 100.00        |
|                                           | Percent of column total | 22.95  | 22.58  | 22.76         |
|                                           | Percent of total        | 11.38  | 11.38  | 22.76         |
| Administrative                            | Number of employees     | 11     | 15     | 26            |
|                                           | Percent of row total    | 42.31  | 57.69  | 100.00        |
|                                           | Percent of column total | 18.03  | 24.19  | 21.14         |
|                                           | Percent of total        | 8.94   | 12.20  | 21.14         |
| All Jobs                                  | Number of employees     | 61     | 62     | 123           |
|                                           | Percent of row total    | 49.59  | 50.41  | 100.00        |
|                                           | Percent of column total | 100.00 | 100.00 | 100.00        |
|                                           | Percent of total        | 49.59  | 50.41  | 100.00        |

PROC TABULATE looks at the first element in the denominator definition, Gender, and asks whether Gender contributes to the subtable. Because Gender does not contribute to the subtable, PROC TABULATE looks at the next element in the denominator definition, which is All. The variable All does contribute to this subtable, so PROC TABULATE uses it as the denominator definition. All is a reserved class variable with only one category. Therefore, this denominator definition tells PROC TABULATE to use the frequency count of All as the denominator.

There is only one category in this subtable: all, all. The denominator for this category is 123.

**Note:** In this table cell, because the numerator and denominator are the same, the row percentage in this subtable is 100.

## Column Percentages

The part of the TABLE statement that calculates the column percentages and labels the row is

```
pctn<occupation all>='Column percent'
```

Consider how PROC TABULATE interprets this denominator definition for each subtable.

**Output 61.32** Subtable 1: Occupation and Gender

| Gender Distribution<br>within Job Classes |                         |        |        |               |
|-------------------------------------------|-------------------------|--------|--------|---------------|
| Job Class                                 |                         | Gender |        | All Employees |
|                                           |                         | Female | Male   |               |
| Technical                                 | Number of employees     | 16     | 18     | 34            |
|                                           | Percent of row total    | 47.06  | 52.94  | 100.00        |
|                                           | Percent of column total | 26.23  | 29.03  | 27.64         |
|                                           | Percent of total        | 13.01  | 14.63  | 27.64         |
| Manager/Supervisor                        | Number of employees     | 20     | 15     | 35            |
|                                           | Percent of row total    | 57.14  | 42.86  | 100.00        |
|                                           | Percent of column total | 32.79  | 24.19  | 28.46         |
|                                           | Percent of total        | 16.26  | 12.20  | 28.46         |
| Clerical                                  | Number of employees     | 14     | 14     | 28            |
|                                           | Percent of row total    | 50.00  | 50.00  | 100.00        |
|                                           | Percent of column total | 22.95  | 22.58  | 22.76         |
|                                           | Percent of total        | 11.38  | 11.38  | 22.76         |
| Administrative                            | Number of employees     | 11     | 15     | 26            |
|                                           | Percent of row total    | 42.31  | 57.69  | 100.00        |
|                                           | Percent of column total | 18.03  | 24.19  | 21.14         |
|                                           | Percent of total        | 8.94   | 12.20  | 21.14         |
| All Jobs                                  | Number of employees     | 61     | 62     | 123           |
|                                           | Percent of row total    | 49.59  | 50.41  | 100.00        |
|                                           | Percent of column total | 100.00 | 100.00 | 100.00        |
|                                           | Percent of total        | 49.59  | 50.41  | 100.00        |

PROC TABULATE looks at the first element in the denominator definition, Occupation, and asks whether Occupation contributes to the subtable. Because Occupation does contribute to the subtable, PROC TABULATE uses it as the denominator definition. This denominator definition tells PROC TABULATE to sum the frequency counts for all occurrences of Occupation within the same value of Gender.

For example, the denominator for the category `manager/supervisor, male` is the sum of all frequency counts for all categories in this subtable for which the value of `Gender` is `male`. There are four such categories: `technical, male`; `manager/supervisor, male`; `clerical, male`; and `administrative, male`. The corresponding frequency counts are 18, 15, 14, and 15. Therefore, the denominator for this category is  $18+15+14+15$ , or 62.

**Output 61.33** Subtable 2: All and Gender

| Gender Distribution<br>within Job Classes |                         |        |        |               |
|-------------------------------------------|-------------------------|--------|--------|---------------|
|                                           |                         | Gender |        | All Employees |
|                                           |                         | Female | Male   |               |
| Job Class                                 |                         |        |        |               |
| Technical                                 | Number of employees     | 16     | 18     | 34            |
|                                           | Percent of row total    | 47.06  | 52.94  | 100.00        |
|                                           | Percent of column total | 26.23  | 29.03  | 27.64         |
|                                           | Percent of total        | 13.01  | 14.63  | 27.64         |
| Manager/Supervisor                        | Number of employees     | 20     | 15     | 35            |
|                                           | Percent of row total    | 57.14  | 42.86  | 100.00        |
|                                           | Percent of column total | 32.79  | 24.19  | 28.46         |
|                                           | Percent of total        | 16.26  | 12.20  | 28.46         |
| Clerical                                  | Number of employees     | 14     | 14     | 28            |
|                                           | Percent of row total    | 50.00  | 50.00  | 100.00        |
|                                           | Percent of column total | 22.95  | 22.58  | 22.76         |
|                                           | Percent of total        | 11.38  | 11.38  | 22.76         |
| Administrative                            | Number of employees     | 11     | 15     | 26            |
|                                           | Percent of row total    | 42.31  | 57.69  | 100.00        |
|                                           | Percent of column total | 18.03  | 24.19  | 21.14         |
|                                           | Percent of total        | 8.94   | 12.20  | 21.14         |
| All Jobs                                  | Number of employees     | 61     | 62     | 123           |
|                                           | Percent of row total    | 49.59  | 50.41  | 100.00        |
|                                           | Percent of column total | 100.00 | 100.00 | 100.00        |
|                                           | Percent of total        | 49.59  | 50.41  | 100.00        |

PROC TABULATE looks at the first element in the denominator definition, `Occupation`, and asks whether `Occupation` contributes to the subtable. Because `Occupation` does not contribute to the subtable, PROC TABULATE looks at the next element in the denominator definition, which is `All`. Because the variable `All` does contribute to this subtable, PROC TABULATE uses it as the denominator definition. `All` is a reserved class variable with only one category. Therefore, this denominator definition tells PROC TABULATE to use the frequency count for `All` as the denominator.

For example, the denominator for the category `all, female` is the frequency count for that category, 61.



**Note:** In these table cells, because the numerator and denominator are the same, the column percentages in this subtable are all 100.

**Output 61.34** Subtable 3: Occupation and All

|                    |                         | Gender |        | All Employees |
|--------------------|-------------------------|--------|--------|---------------|
|                    |                         | Female | Male   |               |
| Job Class          |                         |        |        |               |
| Technical          | Number of employees     | 16     | 18     | 34            |
|                    | Percent of row total    | 47.06  | 52.94  | 100.00        |
|                    | Percent of column total | 26.23  | 29.03  | 27.64         |
|                    | Percent of total        | 13.01  | 14.63  | 27.64         |
| Manager/Supervisor | Number of employees     | 20     | 15     | 35            |
|                    | Percent of row total    | 57.14  | 42.86  | 100.00        |
|                    | Percent of column total | 32.79  | 24.19  | 28.46         |
|                    | Percent of total        | 16.26  | 12.20  | 28.46         |
| Clerical           | Number of employees     | 14     | 14     | 28            |
|                    | Percent of row total    | 50.00  | 50.00  | 100.00        |
|                    | Percent of column total | 22.95  | 22.58  | 22.76         |
|                    | Percent of total        | 11.38  | 11.38  | 22.76         |
| Administrative     | Number of employees     | 11     | 15     | 26            |
|                    | Percent of row total    | 42.31  | 57.69  | 100.00        |
|                    | Percent of column total | 18.03  | 24.19  | 21.14         |
|                    | Percent of total        | 8.94   | 12.20  | 21.14         |
| All Jobs           | Number of employees     | 61     | 62     | 123           |
|                    | Percent of row total    | 49.59  | 50.41  | 100.00        |
|                    | Percent of column total | 100.00 | 100.00 | 100.00        |
|                    | Percent of total        | 49.59  | 50.41  | 100.00        |

PROC TABULATE looks at the first element in the denominator definition, Occupation, and asks whether Occupation contributes to the subtable. Because Occupation does contribute to the subtable, PROC TABULATE uses it as the denominator definition. This denominator definition tells PROC TABULATE to sum the frequency counts for all occurrences of Occupation in the subtable.

For example, the denominator for the category `technical, all` is the sum of the frequency counts for `technical, all`; `manager/supervisor, all`; `clerical, all`; and `administrative, all`. The corresponding frequency counts are 34, 35, 28, and 26. Therefore, the denominator for this category is 34+35+28+26, or 123.

**Output 61.35** Subtable 4: All and All

|                    |                         | Gender |        | All Employees |
|--------------------|-------------------------|--------|--------|---------------|
|                    |                         | Female | Male   |               |
| Job Class          |                         |        |        |               |
| Technical          | Number of employees     | 16     | 18     | 34            |
|                    | Percent of row total    | 47.06  | 52.94  | 100.00        |
|                    | Percent of column total | 26.23  | 29.03  | 27.64         |
|                    | Percent of total        | 13.01  | 14.63  | 27.64         |
| Manager/Supervisor | Number of employees     | 20     | 15     | 35            |
|                    | Percent of row total    | 57.14  | 42.86  | 100.00        |
|                    | Percent of column total | 32.79  | 24.19  | 28.46         |
|                    | Percent of total        | 16.26  | 12.20  | 28.46         |
| Clerical           | Number of employees     | 14     | 14     | 28            |
|                    | Percent of row total    | 50.00  | 50.00  | 100.00        |
|                    | Percent of column total | 22.95  | 22.58  | 22.76         |
|                    | Percent of total        | 11.38  | 11.38  | 22.76         |
| Administrative     | Number of employees     | 11     | 15     | 26            |
|                    | Percent of row total    | 42.31  | 57.69  | 100.00        |
|                    | Percent of column total | 18.03  | 24.19  | 21.14         |
|                    | Percent of total        | 8.94   | 12.20  | 21.14         |
| All Jobs           | Number of employees     | 61     | 62     | 123           |
|                    | Percent of row total    | 49.59  | 50.41  | 100.00        |
|                    | Percent of column total | 100.00 | 100.00 | 100.00        |
|                    | Percent of total        | 49.59  | 50.41  | 100.00        |

PROC TABULATE looks at the first element in the denominator definition, Occupation, and asks whether Occupation contributes to the subtable. Because Occupation does not contribute to the subtable, PROC TABULATE looks at the next element in the denominator definition, which is All. Because the variable All does contribute to this subtable, PROC TABULATE uses it as the denominator definition. All is a reserved class variable with only one category. Therefore, this denominator definition tells PROC TABULATE to use the frequency count of All as the denominator.

There is only one category in this subtable: all, all. The frequency count for this category is 123.

**Note:** In this calculation, because the numerator and denominator are the same, the column percentage in this subtable is 100.

## Total Percentages

The part of the TABLE statement that calculates the total percentages and labels the row is:

```
pctn='Total percent'
```

If you do not specify a denominator definition, then PROC TABULATE obtains the denominator for a cell by totaling all the frequency counts in the subtable. The following table summarizes the process for all subtables in this example.

**Table 61.10** Denominators for Total Percentages

| Class Variables<br>Contributing to the<br>Subtable | Frequency Counts              | Total |
|----------------------------------------------------|-------------------------------|-------|
| Occupant and Gender                                | 16, 18, 20, 15 14, 14, 11, 15 | 123   |
| Occupant and All                                   | 34, 35, 28, 26                | 123   |
| Gender and All                                     | 61, 62                        | 123   |
| All and All                                        | 123                           | 123   |

Consequently, the denominator for total percentages is always 123.

## Example 13: Specifying Style Overrides for ODS Output

Features:

- CLASS statement option  
STYLE=
- CLASSLEV statement option  
STYLE=
- KEYLABEL statement
- KEYWORD statement option  
STYLE=
- PROC TABULATE statement options  
DATA=  
STYLE=
- TABLE statement options  
STYLE=  
MISSTEXT=  
BOX=

ODS HTML5 statement  
 ODS PDF statement  
 ODS WORD statement  
 OPTIONS statement  
 TITLE statement

Data set: ENERGY  
 Format: REGFMT.  
 Format: DIVFMT.  
 Format: USETYPE.

---

## Details

This example creates HTML5, Word, and PDF files and specifies style overrides for various table regions.

### Program

```
options nodate pageno=1;

proc sort data=energy;
 by region;
run;

ods _all_ close;
ods html5 path='file-path' file='CustomEnergy.htm';
ods pdf file='file-path/CustomEnergy.pdf' contents=yes;
ods word file='file-path/CustomEnergy.docx';

proc tabulate data=energy style=[fontweight=bold];
 by region;

 class region division type / style=[textalign=center];
 classlev region division type / style=[textalign=left];
 var expenditures / style=[fontsize=3];
 keyword all sum / style=[fontwidth=wide];
 keylabel all="Total";

 table (region all)*(division all*[style=[backgroundcolor=yellow]]),
 (type all)*(expenditures*f=dollar10.) /
 style=[bordercolor=blue]

 misstext=[label="Missing" style=[fontweight=light]]

 box=[label="Region by Division by Type"
 style=[fontstyle=italic]];

 format region regfmt. division divfmt. type usetype.;
```

```

title 'Energy Expenditures';
title2 '(millions of dollars)';
run;
ods _all_ close;

```

## Program Description

**Set the SAS system options.** The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number.

```
options nodate pageno=1;
```

**Sort the data set.**

```

proc sort data=energy;
 by region;
run;

```

**Specify the ODS output filenames.** By opening multiple ODS destinations, you can produce multiple output files in a single execution. The ODS HTML5 statement produces output that is written in HTML 5.0. The ODS PDF statement produces output in Portable Document Format (PDF). The ODS WORD statement produces output for Microsoft Word. The output from PROC TABULATE goes to each of these files.

In the ODS PDF statement, the CONTENTS= option creates a table of contents.

```

ods _all_ close;
ods html5 path='file-path' file='CustomEnergy.htm';
ods pdf file='file-path/CustomEnergy.pdf' contents=yes;
ods word file='file-path/CustomEnergy.docx';

```

**Customize the data cells.** The STYLE= option in the PROC TABULATE statement specifies the style override for the data cells of the table.

```
proc tabulate data=energy style=[fontweight=bold];
```

**Specify the BY-group.** When BY statements are specified, labels for the BY group tables are displayed in the table of contents. The labels are based on the values of the BY variable.

```
by region;
```

**Customize the class variable name headings.** The STYLE= option in the CLASS statement specifies the style override for the class variable name headings.

```
class region division type / style=[textalign=center];
```

**Customize the class variable value headings.** The STYLE= option in the CLASSLEV statement specifies the style override for the class variable level value headings.

```
classlev region division type / style=[textalign=left];
```

**Customize the analysis variable name headings.** The STYLE= option in the VAR statement specifies a style element for the variable name headings.

```
var expenditures / style=[fontsize=3];
```

**Specify the style attributes for keywords, and label the “all” keyword.** The STYLE= option in the KEYWORD statement specifies a style element for keywords. The KEYLABEL statement assigns a label to the keyword.

```
keyword all sum / style=[fontwidth=wide];
keylabel all="Total";
```

**Define and customize the table rows and columns.** The STYLE= option in the dimension expression overrides any other STYLE= specifications in PROC TABULATE that specify overrides for table cells. The STYLE= option after the slash (/) specifies style overrides for parts of the table other than table cells.

```
table (region all)*(division all*[style=[backgroundcolor=yellow]]),
 (type all)*(expenditures*f=dollar10.) /
 style=[bordercolor=blue]
```

**Customize missing values.** The STYLE= option in the MISSTEXT option of the TABLE statement specifies a style element to use for the text in table cells that contain missing values.

```
misstext=[label="Missing" style=[fontweight=light]]
```

**Customize the box above the row titles.** The STYLE= option in the BOX option of the TABLE statement specifies a style override for text in the box above the row titles.

```
box=[label="Region by Division by Type"
 style=[fontstyle=italic]];
```

**Format the class variable values.**

```
format region regfmt. division divfmt. type usetype.;
```

**Specify the titles.**

```
title 'Energy Expenditures';
title2 '(millions of dollars)';
run;
```

**Close all of the ODS destinations.**

```
ods _all_ close;
```

## Output

**Output 61.36** HTML Output Showing Style Overrides

| Energy Expenditures<br>(millions of dollars) |                 |                       |                    |              |
|----------------------------------------------|-----------------|-----------------------|--------------------|--------------|
| Region=Northeast                             |                 |                       |                    |              |
| Region by Division by Type                   |                 | Type                  |                    | Total        |
|                                              |                 | Residential Customers | Business Customers | Expenditures |
|                                              |                 | Expenditures          | Expenditures       |              |
|                                              |                 | Sum                   | Sum                | Sum          |
| Region                                       | Division        |                       |                    |              |
| Northeast                                    | New England     | \$7,477               | \$5,129            | \$12,606     |
|                                              | Middle Atlantic | \$19,379              | \$15,078           | \$34,457     |
|                                              | Total           | \$26,856              | \$20,207           | \$47,063     |
| Total                                        | Division        |                       |                    |              |
|                                              | New England     | \$7,477               | \$5,129            | \$12,606     |
|                                              | Middle Atlantic | \$19,379              | \$15,078           | \$34,457     |
|                                              | Total           | \$26,856              | \$20,207           | \$47,063     |

**Output 61.37** PDF Output Showing Style Overrides

| Energy Expenditures<br>(millions of dollars) |                 |                       |                    |              |
|----------------------------------------------|-----------------|-----------------------|--------------------|--------------|
| Region=Northeast                             |                 |                       |                    |              |
| Region by Division by Type                   |                 | Type                  |                    | Total        |
|                                              |                 | Residential Customers | Business Customers | Expenditures |
|                                              |                 | Expenditures          | Expenditures       |              |
|                                              |                 | Sum                   | Sum                | Sum          |
| Region                                       | Division        |                       |                    |              |
| Northeast                                    | New England     | \$7,477               | \$5,129            | \$12,606     |
|                                              | Middle Atlantic | \$19,379              | \$15,078           | \$34,457     |
|                                              | Total           | \$26,856              | \$20,207           | \$47,063     |
| Total                                        | Division        |                       |                    |              |
|                                              | New England     | \$7,477               | \$5,129            | \$12,606     |
|                                              | Middle Atlantic | \$19,379              | \$15,078           | \$34,457     |
|                                              | Total           | \$26,856              | \$20,207           | \$47,063     |

**Output 61.38** Word Output Showing Style Overrides

| Energy Expenditures<br>(millions of dollars) |                 |                          |                       |          |
|----------------------------------------------|-----------------|--------------------------|-----------------------|----------|
| Region=Northeast                             |                 |                          |                       |          |
| <i>Region by Division by Type</i>            |                 | Type                     |                       | Total    |
|                                              |                 | Residential<br>Customers | Business<br>Customers |          |
|                                              |                 | Expenditures             | Expenditures          |          |
|                                              |                 | Sum                      | Sum                   |          |
| Region                                       | Division        |                          |                       |          |
| Northeast                                    | New England     | \$7,477                  | \$5,129               | \$12,606 |
|                                              | Middle Atlantic | \$19,379                 | \$15,078              | \$34,457 |
|                                              | Total           | \$26,856                 | \$20,207              | \$47,063 |
| Total                                        | Division        |                          |                       |          |
|                                              | New England     | \$7,477                  | \$5,129               | \$12,606 |
|                                              | Middle Atlantic | \$19,379                 | \$15,078              | \$34,457 |
|                                              | Total           | \$26,856                 | \$20,207              | \$47,063 |

## Example 14: Style Precedence

Features:

- CLASS statementO
- CLASSLEV statement option
- STYLE=
- KEYLABEL statement
- LABEL statement
- PROC TABULATE statement options
- DATA=
- FORMAT=
- TABLE statement
- crossing (\*) operator
- STYLE=
- STYLE\_PRECEDENCE= option
- VAR statement
- ODS HTML5 statement
- FORMAT procedure
- FORMAT statement
- TITLE statement

Data set: **SALES**



## Details

This example does the following:

- creates a category for each sales type, retail or wholesale, in each region
- applies the dollar format to all cells in the table
- applies an italic font style for each region and sales type
- applies a style (background = #edf8b1 (yellow-green) or orange) color based on the STYLE\_PRECEDENCE = option
- generates ODS HTML5 output

## Program

```
proc format;
 value $saletypefmt 'R'='Retail'
 'W'='WholeSale';
run;

ods _all_ close;
ods html5 file="file-path/stylePrecedence.html";

title "Style Precedence";
title2 "First Table: no precedence, Orange";
title3 "Second Table: style_precedence=page, Yellow";

proc tabulate data=sales format=dollar10.;
class product region saletype;

classlev region saletype / style={font_style=italic};

var netsales;
label netsales="Net Sales";

keylabel all="Total";

table product *{style={background=#edf8b1}},
 region*{style={background=yellow}},
 saletype*{style={background=orange}};

table product *{style={background=#edf8b1}},
 region*{style={background=yellow}},
 saletype*{style={background=orange}} / style_precedence=page;

format saletype $saletypefmt.;

run;
```

## Program Description

**Create the SALETYPEFMT. formats.** PROC FORMAT creates formats for SALETYPE.

```
proc format;
 value $saletypefmt 'R'='Retail'
 'W'='WholeSale';
run;
```

**Specify the ODS output filename.** The ODS HTML5 statement produces output that is written in HTML5.

```
ods _all_ close;
ods html5 file="file-path/stylePrecedence.html";
```

**Specify the titles of the tables to be produced.** Two tables will be generated. The First Table will show no style precedence whereas the Second Table will show that the color that takes precedence is based on what is specified by the STYLE\_PRECEDENCE option.

```
title "Style Precedence";
title2 "First Table: no precedence, Orange";
title3 "Second Table: style_precedence=page, Yellow";
```

**Specify the table options.** The FORMAT= option specifies DOLLAR10. as the default format for the value in each table cell.

```
proc tabulate data=sales format=dollar10.;
```

**Specify subgroups for the analysis.** The CLASS statement separates the analysis by values of Product, Region, and SaleType.

```
class product region saletype;
```

**Specify styles for the subgroups.** The CLASSLEV statement specifies a style for the Region and Saletype elements.

```
classlev region saletype / style={font_style=italic};
```

**Specify the analysis variable.** The VAR statement specifies that PROC TABULATE calculate statistics on the Netsales variable.

```
var netsales;
```

**Specify labels.** The LABEL statement renames the Netsales variable to Net Sales.

```
label netsales="Net Sales";
```

**Specify Keylabel.** The KEYLABEL statement labels the universal class variable ALL to Total.

```
keylabel all="Total";
```

**Define the table rows and columns.** The TABLE statement creates a table per product per page. In this example, there is one product, A100. The TABLE statement also creates a row for each formatted value of Region and creates a column for each formatted value of SaleType. Each cell that is created by these rows and columns contains the sum of the analysis variable Net Sales for all observations that contribute to that cell. The STYLE= option in the dimension expression overrides any other STYLE= specifications in PROC TABULATE that specify attributes for the table cells. In this first table, the column expression is the default and the style associated with column takes precedence. Therefore, orange will be the default color of the background.

```
table product *{style={background=#edf8b1}},
region*{style={background=yellow}},
saletype*{style={background=orange}};
```

**Define the table rows and columns using the STYLE\_PRECEDENCE option.** The TABLE statement creates a table per product per page, A100. The TABLE statement also creates a row for each formatted value of Region and creates a column for each formatted value of SaleType. Each cell that is created by these rows and columns contains the sum of the analysis variable Net Sales for all observations that contribute to that cell. The STYLE= option in the dimension expression overrides any other STYLE= specifications in PROC TABULATE that specify attributes for the table cells. In this second table, the STYLE\_PRECEDENCE option is specified on the page expression. Therefore, the style that applies to the background is #edf8b1 (yellow-green.)

```
table product *{style={background=#edf8b1}},
region*{style={background=yellow}},
saletype*{style={background=orange}} / style_precedence=page;
```

**Format the output.** The FORMAT statement assigns formats to the SaleType variable.

```
format saletype $saletypefmt.;
```

**Run the program.**

```
run;
```

## Output

**Output 61.39** Table with No Style Precedence

**Style Precedence**  
**First Table: no precedence, orange**  
**Second Table: style\_precedence=page, Yellow**

Product A100

|        | SaleType      |                  |
|--------|---------------|------------------|
|        | <i>Retail</i> | <i>WholeSale</i> |
|        | N             | N                |
| Region |               |                  |
| NC     | \$3           | \$3              |
| TX     | .             | \$2              |

**Output 61.40** Table Style Precedence

**Style Precedence**  
**First Table: no precedence, orange**  
**Second Table: style\_precedence=page, Yellow**

Product A100

|        | SaleType      |                  |
|--------|---------------|------------------|
|        | <i>Retail</i> | <i>WholeSale</i> |
|        | N             | N                |
| Region |               |                  |
| NC     | \$3           | \$3              |
| TX     | .             | \$2              |

## Example 15: Using the NOCELLMERGE Option

Features:

- CLASS statement
- PROC TABULATE statement options
  - DATA=
  - STYLE=
- CLASS statement
- TABLE statement
  - crossing (\*) operator
  - STYLE= option
  - NOCELLMERGE= option

ODS HTML5 statement

TITLE statement

---

---

## Details

This example does the following:

- creates a table with merged cells style behavior
- creates a second table without merged cells
- shows how cells styles are affected when empty data cells and the formatted data cells use different styles

---

## Program

```
ods html5 file="file-path/tabstyle.html";
proc tabulate data=sashelp.class style={background=#edf8b1};
class sex age;

table sex*{style={background=#7fcdbb}} all, age;
title 'Data Cell Styles in Merged Cells';
run;

proc tabulate data=sashelp.class style={background=#edf8b1};
class sex age;

table sex*{style={background=#7fcdbb}} all, age/nocellmerge;
title1 'Data Cell Styles with NOCELLMERGE Option';
run;

ods html5 close;
```

---

## Program Description

**Specify the ODS output filename.** The ODS HTML5 statement produces output that is written in HTML.

```
ods html5 file="file-path/tabstyle.html";
```

**Specify the PROC TABULATE options.** The STYLE= option sets the background color for the cells in the table to #edf8b1 (yellow-green.)

```
proc tabulate data=sashelp.class style={background=#edf8b1};
```

**Specify subgroups.** The CLASS statement separates the data by sex and age.

```
class sex age;
```

**Define the table rows and columns.** The TABLE statement creates a table. The STYLE= option in the dimension expression overrides the STYLE= setting from the PROC TABULATE statement for table cells attributes, changing the color to #7fcdbb (green-cyan.)

```
table sex*{style={background=#7fcdbb}} all, age;
```

**Specify the title of the table to be produced.** This table shows how changing the style color affects the merged cells.

```
title 'Data Cell Styles in Merged Cells';
```

**Run the program.**

```
run;
```

**Specify the PROC TABULATE options.** The STYLE= option sets the background color for the cells in the table to #edf8b1 (yellow-green.)

```
proc tabulate data=sashelp.class style={background=#edf8b1};
```

**Specify subgroups.** The CLASS statement separates the data by sex and age.

```
class sex age;
```

**Define the table rows and columns.** The TABLE statement creates a table. The STYLE= option in the dimension expression overrides the STYLE= setting from the PROC TABULATE statement, but only for the formatted data cells.

```
table sex*{style={background=#7fcdbb}} all, age/nocellmerge;
```

**Specify the title of the table to be produced.** This table shows how changing the style color affects the formatted cells that are not merged.

```
title1 'Data Cell Styles with NOCELLMERGE Option';
```

**Run the program.**

```
run;
```

**Close the ODS HTML5 output destination.**

```
ods html5 close;
```

## Output

**Output 61.41** Merged Cells

### Data Cell Styles in Merged Cells

|     | Age |    |    |    |    |    |
|-----|-----|----|----|----|----|----|
|     | 11  | 12 | 13 | 14 | 15 | 16 |
|     | N   | N  | N  | N  | N  | N  |
| Sex |     |    |    |    |    |    |
| F   | 1   | 2  | 2  | 2  | 2  | .  |
| M   | 1   | 3  | 1  | 2  | 2  | 1  |
| All | 2   | 5  | 3  | 4  | 4  | 1  |

**Output 61.42** Unmerged Cells

### Data Cell Styles with NOCELLMERGE Option

|     | Age |    |    |    |    |    |
|-----|-----|----|----|----|----|----|
|     | 11  | 12 | 13 | 14 | 15 | 16 |
|     | N   | N  | N  | N  | N  | N  |
| Sex |     |    |    |    |    |    |
| F   | 1   | 2  | 2  | 2  | 2  | .  |
| M   | 1   | 3  | 1  | 2  | 2  | 1  |
| All | 2   | 5  | 3  | 4  | 4  | 1  |

## References

Jain, Raj and Imrich Chlamtac. 1985. "The P<sup>2</sup> Algorithm for Dynamic Calculation of Quantiles and Histograms without Storing Observations." *Communications of the Association of Computing Machinery* 28 (10): 1076–1085.





# TRANSPOSE Procedure

---

|                                                                                     |             |
|-------------------------------------------------------------------------------------|-------------|
| <b>Overview: TRANSPOSE Procedure</b>                                                | <b>2371</b> |
| What Does the TRANSPOSE Procedure Do?                                               | 2372        |
| What Types of Transpositions Can PROC TRANSPOSE Perform?                            | 2372        |
| CAS Processing for PROC TRANSPOSE                                                   | 2375        |
| <b>Syntax: TRANSPOSE Procedure</b>                                                  | <b>2375</b> |
| PROC TRANSPOSE Statement                                                            | 2376        |
| ATTRIB Statement                                                                    | 2378        |
| BY Statement                                                                        | 2379        |
| COPY Statement                                                                      | 2384        |
| FORMAT Statement                                                                    | 2385        |
| ID Statement                                                                        | 2385        |
| IDLABEL Statement                                                                   | 2387        |
| LABEL Statement                                                                     | 2388        |
| VAR Statement                                                                       | 2390        |
| WHERE Statement                                                                     | 2390        |
| <b>Results: TRANSPOSE Procedure</b>                                                 | <b>2393</b> |
| Output Data Set                                                                     | 2393        |
| Output Data Set Variables                                                           | 2393        |
| Attributes of Transposed Variables                                                  | 2394        |
| Names of Transposed Variables                                                       | 2394        |
| <b>Examples: TRANSPOSE Procedure</b>                                                | <b>2395</b> |
| Example 1: Performing a Simple Transposition                                        | 2395        |
| Example 2: Naming Transposed Variables                                              | 2397        |
| Example 3: Labeling Transposed Variables                                            | 2398        |
| Example 4: Transposing BY Groups                                                    | 2401        |
| Example 5: Naming Transposed Variables When the ID Variable<br>Has Duplicate Values | 2403        |
| Example 6: Transposing Data for Statistical Analysis                                | 2405        |

---

# Overview: TRANSPOSE Procedure

---

---

## What Does the TRANSPOSE Procedure Do?

---

The TRANSPOSE procedure creates an output data set by restructuring the values in a SAS data set, transposing selected variables into observations. The TRANSPOSE procedure can often eliminate the need to write a lengthy DATA step to achieve the same result. Further, the output data set can be used in subsequent DATA or PROC steps for analysis, reporting, or further data manipulation.

PROC TRANSPOSE does not produce printed output. To print the output data set from the PROC TRANSPOSE step, use PROC PRINT, PROC REPORT, or another SAS reporting tool.

To create a transposed variable, the procedure transposes the values of an observation in the input data set into values of a variable in the output data set.

---

## What Types of Transpositions Can PROC TRANSPOSE Perform?

---

---

### Simple Transposition

---

The following example illustrates a simple transposition. In the input data set, each variable represents the scores from one tester. In the output data set, each observation now represents the scores from one tester. Each value of `_NAME_` is the name of a variable in the input data set that the procedure transposed. Thus, the value of `_NAME_` identifies the source of each observation in the output data set. For example, the values in the first observation in the output data set come from the values of the variable `Tester1` in the input data set. The statements that produce the output follow.

```
proc print data=proclib.product noobs;
 title 'The Input Data Set';
run;

proc transpose data=proclib.product
 out=proclib.product_transposed;
```

```
run;

proc print data=proclib.product_transposed noobs;
 title 'The Output Data Set';
run;
```

### Output 62.1 A Simple Transposition

| The Input Data Set |         |         |         |         | 1 |
|--------------------|---------|---------|---------|---------|---|
|                    | Tester1 | Tester2 | Tester3 | Tester4 |   |
|                    | 22      | 25      | 21      | 21      |   |
|                    | 15      | 19      | 18      | 17      |   |
|                    | 17      | 19      | 19      | 19      |   |
|                    | 20      | 19      | 16      | 19      |   |
|                    | 14      | 15      | 13      | 13      |   |
|                    | 15      | 17      | 18      | 19      |   |
|                    | 10      | 11      | 9       | 10      |   |
|                    | 22      | 24      | 23      | 21      |   |

| The Output Data Set |      |      |      |      |      |      |      |      | 2 |
|---------------------|------|------|------|------|------|------|------|------|---|
| _NAME_              | COL1 | COL2 | COL3 | COL4 | COL5 | COL6 | COL7 | COL8 |   |
| Tester1             | 22   | 15   | 17   | 20   | 14   | 15   | 10   | 22   |   |
| Tester2             | 25   | 19   | 19   | 19   | 15   | 17   | 11   | 24   |   |
| Tester3             | 21   | 18   | 19   | 16   | 13   | 18   | 9    | 23   |   |
| Tester4             | 21   | 17   | 19   | 19   | 13   | 19   | 10   | 21   |   |

## Complex Transposition Using BY Groups

The next example, which uses BY groups, is more complex. The input data set represents measurements of the weight and length of fish at two lakes. The statements that create the output data set do the following:

- transpose only the variables that contain the length measurements
- create six BY groups, one for each lake and date
- use a data set option to name the transposed variable

**Output 62.2** A Transposition with BY Groups

| Input Data Set |         |    |      |    |      |    |      |    |      | 1 |
|----------------|---------|----|------|----|------|----|------|----|------|---|
| Location       | Date    | L  | W    | L  | W    | L  | W    | L  | W    |   |
|                |         | e  | e    | e  | e    | e  | e    | e  | e    |   |
|                |         | n  | i    | n  | i    | n  | i    | n  | i    |   |
|                |         | g  | g    | g  | g    | g  | g    | g  | g    |   |
|                |         | t  | h    | t  | h    | t  | h    | t  | h    |   |
| o              | t       | h  | t    | h  | t    | h  | t    | t  |      |   |
| n              | e       | 1  | 1    | 2  | 2    | 3  | 3    | 4  | 4    |   |
| Cole Pond      | 02JUN95 | 31 | 0.25 | 32 | 0.30 | 32 | 0.25 | 33 | 0.30 |   |
| Cole Pond      | 03JUL95 | 33 | 0.32 | 34 | 0.41 | 37 | 0.48 | 32 | 0.28 |   |
| Cole Pond      | 04AUG95 | 29 | 0.23 | 30 | 0.25 | 34 | 0.47 | 32 | 0.30 |   |
| Eagle Lake     | 02JUN95 | 32 | 0.35 | 32 | 0.25 | 33 | 0.30 | .  | .    |   |
| Eagle Lake     | 03JUL95 | 30 | 0.20 | 36 | 0.45 | .  | .    | .  | .    |   |
| Eagle Lake     | 04AUG95 | 33 | 0.30 | 33 | 0.28 | 34 | 0.42 | .  | .    |   |
| .              |         |    |      |    |      |    |      |    |      |   |

| Fish Length Data for Each Location and Date |         |         |             |  | 2 |
|---------------------------------------------|---------|---------|-------------|--|---|
| Location                                    | Date    | _NAME_  | Measurement |  |   |
| Cole Pond                                   | 02JUN95 | Length1 | 31          |  |   |
| Cole Pond                                   | 02JUN95 | Length2 | 32          |  |   |
| Cole Pond                                   | 02JUN95 | Length3 | 32          |  |   |
| Cole Pond                                   | 02JUN95 | Length4 | 33          |  |   |
| Cole Pond                                   | 03JUL95 | Length1 | 33          |  |   |
| Cole Pond                                   | 03JUL95 | Length2 | 34          |  |   |
| Cole Pond                                   | 03JUL95 | Length3 | 37          |  |   |
| Cole Pond                                   | 03JUL95 | Length4 | 32          |  |   |
| Cole Pond                                   | 04AUG95 | Length1 | 29          |  |   |
| Cole Pond                                   | 04AUG95 | Length2 | 30          |  |   |
| Cole Pond                                   | 04AUG95 | Length3 | 34          |  |   |
| Cole Pond                                   | 04AUG95 | Length4 | 32          |  |   |
| Eagle Lake                                  | 02JUN95 | Length1 | 32          |  |   |
| Eagle Lake                                  | 02JUN95 | Length2 | 32          |  |   |
| Eagle Lake                                  | 02JUN95 | Length3 | 33          |  |   |
| Eagle Lake                                  | 02JUN95 | Length4 | .           |  |   |
| Eagle Lake                                  | 03JUL95 | Length1 | 30          |  |   |
| Eagle Lake                                  | 03JUL95 | Length2 | 36          |  |   |
| Eagle Lake                                  | 03JUL95 | Length3 | .           |  |   |
| Eagle Lake                                  | 03JUL95 | Length4 | .           |  |   |
| Eagle Lake                                  | 04AUG95 | Length1 | 33          |  |   |
| Eagle Lake                                  | 04AUG95 | Length2 | 33          |  |   |
| Eagle Lake                                  | 04AUG95 | Length3 | 34          |  |   |
| Eagle Lake                                  | 04AUG95 | Length4 | .           |  |   |

For a complete explanation of the SAS program that produces these results, see [“Example 4: Transposing BY Groups” on page 2401](#).

## CAS Processing for PROC TRANSPOSE

If your input originates from SAS Cloud Analytic Services (CAS) and your output is directed back to the CAS server, then the transposition might be performed entirely on the server. Running PROC TRANSPOSE with CAS actions has several advantages over processing within SAS. These advantages include reduced network traffic, and the potential for faster processing. Faster processing is possible because in-memory tables are manipulated locally on the server instead of being transferred across a relatively slow network connection. CAS is used because it might have more processing resources at its disposal.

See the “[Transpose Action Set](#)” in *SAS Viya Platform: System Programming Guide* for information about the Transpose action that runs in CAS.

When the DATA= input data set references an in-memory table or view in CAS, the TRANSPOSE procedure can use CAS actions to perform all of its work within the server.

The CAS LIBNAME statement option controls whether and how CAS procedures are run inside CAS. By default, the CAS procedures are run inside CAS when possible. There are many data set options that prevent CAS processing.

CAS is the analytic server and associated cloud services in the SAS Viya platform. The CAS LIBNAME engine can connect a SAS 9.4 session to an existing SAS Cloud Analytic Services session through the CAS session name or the CAS session UUID. The libref then becomes your handle to communicate from SAS with the specific session. See [The Future of Transpose](#) for more information about PROC TRANSPOSE and examples.

For conceptual information about procedures that run in CAS, see [Chapter 4, “CAS Processing of Base Procedures,”](#) on page 75.

## Syntax: TRANSPOSE Procedure

**Restriction:** If the DATA= and OUT= options point to CAS, the transpose is performed within CAS by invoking the CAS TRANSPOSE action.

**Tips:** Does not support the Output Delivery System  
You can use the ATTRIB, FORMAT, LABEL, and WHERE statements. You can also use any global statement. For a list, see “[Dictionary of SAS Global Statements](#)” in *SAS Global Statements: Reference*.

```
PROC TRANSPOSE <DATA=input-data-set> <DELIMITER=delimiter>
<LABEL=label>
<LET> <NAME=name> <OUT=output-data-set> <PREFIX=prefix> <SUFFIX=suffix>;
 ATTRIB variable-list(s) attribute-list(s);
```

```

BY <DESCENDING> variable-1
 <<DESCENDING> variable-2 ...>
 <NOTSORTED>;
COPY variable(s);
FORMAT variable-1 <...variable-n> <format> <DEFAULT=default-format>;
ID variable;
 IDLABEL variable;
LABEL variable-1=label-1<...variable-n=label-n>;
VAR variable(s);
WHERE where-expression-1
 < logical-operator where-expression-n>;

```

| Statement      | Task                                                                                                                      | Example                    |
|----------------|---------------------------------------------------------------------------------------------------------------------------|----------------------------|
| PROC TRANSPOSE | Create an output data set by restructuring the values in a SAS data set, transposing selected variables into observations | Ex. 1, Ex. 2, Ex. 3, Ex. 5 |
| BY             | Transpose each BY group                                                                                                   | Ex. 4                      |
| COPY           | Copy variables directly without transposing them                                                                          | Ex. 6                      |
| ID             | Specify a variable whose values name the transposed variables                                                             | Ex. 2                      |
| IDLABEL        | Create labels for the transposed variables                                                                                | Ex. 3                      |
| VAR            | List the variables to transpose                                                                                           | Ex. 4, Ex. 6               |

## PROC TRANSPOSE Statement

Creates an output data set by restructuring the values in a SAS data set, transposing selected variables into observations.

**Tip:** You can use data set options with the DATA= and OUT= options. You can also use any global statement. For a list, see [“Dictionary of SAS Global Statements” in SAS Global Statements: Reference](#).

**Examples:**

- “Example 1: Performing a Simple Transposition” on page 2395
- “Example 2: Naming Transposed Variables” on page 2397
- “Example 3: Labeling Transposed Variables” on page 2398
- “Example 4: Transposing BY Groups” on page 2401
- “Example 5: Naming Transposed Variables When the ID Variable Has Duplicate Values” on page 2403
- “Example 6: Transposing Data for Statistical Analysis” on page 2405

## Syntax

```
PROC TRANSPOSE <DATA=input-data-set> <DELIMITER=delimiter>
<LABEL=label> <LET> <NAME=name> <OUT=output-data-set> <PREFIX=prefix>
<SUFFIX=suffix>;
```

### Optional Arguments

#### **DATA= *input-data-set***

names the SAS data set to transpose.

Default    most recently created SAS data set

#### **DELIMITER= *delimiter***

specifies a delimiter to use in constructing names for transposed variables in the output data set. If specified, the delimiter is inserted between variable values if more than one variable has been specified in the ID statement.

Alias    DELIM=

Tip    You can use name literals (n-literals) for the value of DELIMITER. Name literals are helpful when specifying typographical or foreign characters, especially when VALIDVARNAME=ANY.

See    [“ID Statement” on page 2385](#)

#### **LABEL= *label***

specifies a name for the variable in the output data set that contains the label of the variable that is being transposed to create the current observation.

Default    \_LABEL\_

Tip    You can use name literals (n-literals) for the value of LABEL. Name literals are helpful when specifying typographical or foreign characters, especially when VALIDVARNAME=ANY.

#### **LET**

allows duplicate values of an ID variable. PROC TRANSPOSE transposes the observation that contains the last occurrence of a particular ID value within the data set or BY group.

See    [“Example 5: Naming Transposed Variables When the ID Variable Has Duplicate Values” on page 2403](#)

#### **NAME= *name***

specifies the name for the variable in the output data set that contains the name of the variable that is being transposed to create the current observation.

Default    \_NAME\_

See    [“Example 2: Naming Transposed Variables” on page 2397](#)

**OUT= *output-data-set***

names the output data set. If *output-data-set* does not exist, then PROC TRANSPOSE creates it by using the DATAn naming convention.

Default DATAn

See [“Example 1: Performing a Simple Transposition” on page 2395](#)

**PREFIX= *prefix***

specifies a prefix to use in constructing names for transposed variables in the output data set. For example, if PREFIX=VAR, then the names of the variables are VAR1, VAR2, ...,VARn.

Interaction When you use PREFIX= with an ID statement, the variable name begins with the prefix value followed by the ID value.

Tip You can use name literals (n-literals) for the value of PREFIX. Name literals are helpful when specifying typographical or foreign characters, especially when VALIDVARNAME=ANY.

See [“Example 2: Naming Transposed Variables” on page 2397](#)

**SUFFIX= *suffix***

specifies a suffix to use in constructing names for transposed variables in the output data set.

Interaction When you use SUFFIX= with an ID statement, the value is appended to the ID value.

Tip You can use name literals (n-literals) for the value of SUFFIX. Name literals are helpful when specifying typographical or foreign characters, especially when VALIDVARNAME=ANY.

---

## ATTRIB Statement

Associates a format, informat, label, and length with one or more variables.

---

### Syntax

**ATTRIB** *variable-list(s)* *attribute-list(s)*;

### Arguments

***variable-list(s)***

names the variables that you want to associate with the attributes.



**TIP** List the variables in any form that SAS allows.

### ***attribute-list(s)***

specifies one or more attributes to assign to *variable-list*. Specify one or more of these attributes in the ATTRIB statement:

#### **FORMAT=***format*

associates a format with variables in *variable-list*.

**Tip** The format can be either a standard SAS format or a format that is defined with the FORMAT procedure.

#### **INFORMAT=***informat*

associates an informat with variables in *variable-list*.

**Tip** The informat can be either a standard SAS informat or an informat that is defined with the FORMAT procedure.

#### **LABEL=***'label'*

associates a label with variables in *variable-list*.

---

## See Also

[“ATTRIB Statement” in SAS DATA Step Statements: Reference.](#)

---

# BY Statement

Orders the output according to the BY groups.

#### Restrictions:

Do not use PROC TRANSPOSE with a BY statement or an ID statement if another user is updating the data set at the same time.

Do not specify a group variable in the BY statement. The BY variable cannot be the same as the group variable. A *group variable* is a variable in the input data set that is used to categorize chart variable values into groups. For more information, see [“Group Variables” on page 1760](#).

---

## Syntax

**BY** <DESCENDING> *variable-1*

<... <DESCENDING> *variable-n*>

<NOTSORTED>;

## Required Argument

### **variable**

specifies the variable that the procedure uses to form BY groups. You can specify more than one variable. If you do not use the NOTSORTED option in the BY statement, then either the observations in the data set must be sorted by all the variables that you specify, or they must be indexed appropriately. Variables in a BY statement are called *BY variables*.

## Optional Arguments

### **DESCENDING**

specifies that the observations are sorted in descending order by the variable that immediately follows the word DESCENDING in the BY statement.

### **NOTSORTED**

specifies that observations are not necessarily sorted in alphabetic or numeric order. The observations are grouped in another way (for example, chronological order).

The requirement for ordering or indexing observations according to the values of BY variables is suspended for BY-group processing when you use the NOTSORTED option. In fact, the procedure does not use an index if you specify NOTSORTED. The procedure defines a BY group as a set of contiguous observations that have the same values for all BY variables. If observations with the same values for the BY variables are not contiguous, then the procedure treats each contiguous set as a separate BY group.

You cannot use the NOTSORTED option in a PROC SORT step.

---

**Note:** You cannot use the GROUPFORMAT option, which is available in the BY statement in a DATA step, in a BY statement in any PROC step.

---



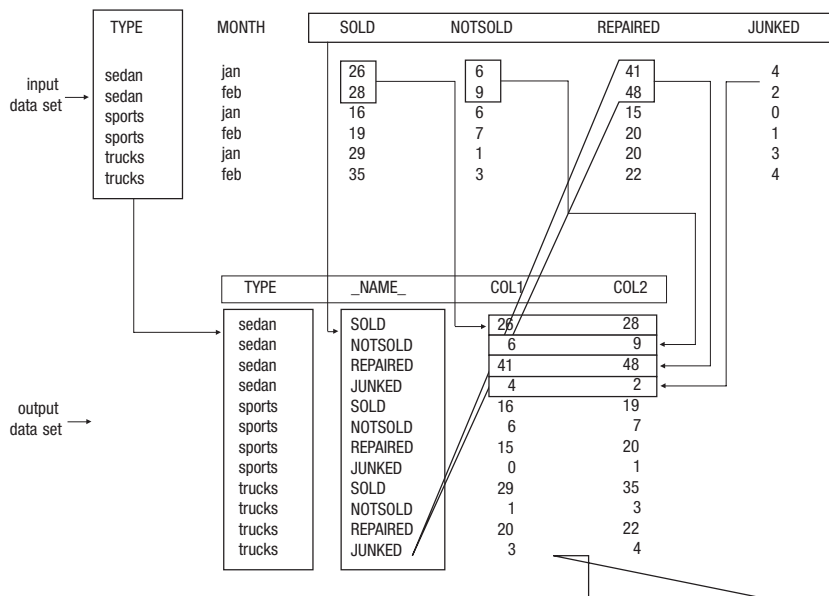
---

## Details

### BY-Group Processing

The following figure shows what happens when you transpose a data set with BY groups. TYPE is the BY variable, and SOLD, NOTSOLD, REPAIRED, and JUNKED are the variables to transpose.

Figure 62.1 Transposition with BY Groups



- The number of observations in the output data set (12) is the number of BY groups (3) multiplied by the number of variables that are transposed (4).
- The BY variable is not transposed.
- `_NAME_` contains the name of the variable in the input data set that was transposed to create the current observation in the output data set. You can use the `NAME=` option to specify another name for the `_NAME_` variable.
- The maximum number of observations in any BY group in the input data set is two. Therefore, the output data set contains two variables, `COL1` and `COL2`. `COL1` and `COL2` contain the values of `SOLD`, `NOTSOLD`, `REPAIRED`, and `JUNKED`.

**Note:** If a BY group in the input data set has more observations than other BY groups, then PROC TRANSPOSE assigns missing values in the output data set to the variables that have no corresponding input observations.

Procedures create output for each BY group. For example, the elementary statistics procedures and the scoring procedures perform separate analyses for each BY group. The reporting and ODS procedures produce a report for each BY group.

**Note:** All Base SAS procedures except PROC PRINT process BY groups independently. PROC PRINT can report the number of observations in each BY group as well as the number of observations in all BY groups. Similarly, PROC PRINT can sum numeric variables in each BY group and across all BY groups.

You can use only one BY statement in each PROC step. When you use a BY statement, the procedure expects an input data set that is sorted by the order of the BY variables or one that has an appropriate index. If your input data set does not meet these criteria, then an error occurs. Either sort it with the SORT procedure or create an appropriate index on the BY variables.

Depending on the order of your data, you might need to use the NOTSORTED or DESCENDING option in the BY statement in the PROC step.

- For more information about the BY statement, see [SAS DATA Step Statements: Reference](#).
- For more information about PROC SORT, see [Chapter 55, “SORT Procedure,” on page 2039](#).
- For more information about BY-group processing in CAS, see [“Grouping and Ordering” in SAS Cloud Analytic Services: DATA Step Programming](#).
- For more information about creating indexes, see [“INDEX CREATE Statement” on page 440](#).

## Formatting BY-Variable Values

When a procedure is submitted with a BY statement, the following actions are taken with respect to processing of BY groups:

- 1 The procedure determines whether the data is sorted by the internal (unformatted) values of the BY variable(s).
- 2 The procedure determines whether a format has been applied to the BY variable(s). If the BY variable is numeric and has no user-applied format, then the BEST12. format is applied for the purpose of BY-group processing.
- 3 The procedure continues adding observations to the current BY group until both the internal and formatted values of the BY variable or variables change.

This process can have unexpected results if, for example, nonconsecutive internal BY values share the same formatted value. In this case, the formatted value is represented in different BY groups. Alternatively, if different consecutive internal BY values share the same formatted value, then these observations are grouped into the same BY group.

## BY Variables That Have Different Lengths in Two Data Sets

When a procedure has a BY statement and two input data sets, one of the input data sets is called the primary data set and the other is called the secondary data set. The primary data set is usually, but not always, the DATA= data set. A BY statement always applies to the primary data set. The variables in the BY statement must appear in the primary data set.

Each procedure determines whether a BY statement applies to a secondary data set, and performs one of the following actions:

- The procedure might always apply the BY statement to the secondary data set. In this case, one or more variables in the BY statement must appear in the secondary data set.
- The procedure might never apply the BY statement to the secondary data set. In this case, the variables in the BY statement are not required in the secondary data set.
- The procedure might check whether the BY variables are in the secondary data set. If none of the BY variables are in the secondary data set, then BY processing

does not occur for the secondary data set. If one or more of the BY variables are in the secondary data set, and they match the BY variables in the primary data set, then BY processing is done for the secondary data set. If some but not all BY variables are in the secondary data set, then the procedure might issue an error message and quit. Or, it might take some other action described in the documentation for that particular procedure.

If the BY statement is applied to the secondary data set, then each BY variable that exists on both the data sets must have the same type, character or numeric, in both data sets. The BY variables are required to have either the same formatted value or the same unformatted value. Formatted values match only if both the formatted lengths and the formatted values are the same. Unformatted values are not required to have the same length in order to match. The unformatted character values match if the unformatted values are the same after stripping the trailing blanks. The unformatted doubles match if they have the same value.

A secondary data set does not need to have all of the BY variables that are in the primary data set. A procedure can define a subset of the BY variables for the secondary data set. For example, if the primary data set has the BY variables A,B,C,D, then the procedure can define the following BY variables on the secondary data set:

- A
- A,B
- A,B,C
- A,B,C,D

If both the primary and secondary data sets have the same number of BY variables, and all the BY variables have the same byte lengths and format lengths, then either the unformatted values or the formatted values in the BY buffer (for all of the BY variables) have to match. If they do not match, then each variable is compared. The formatted values of each variable are compared first. The formatted lengths have to match, and the formatted values have to match. If the formatted lengths and values do not match, then the unformatted values are compared even if the byte lengths are different.

If corresponding character variable lengths differ, then the longer character variable can contain only trailing blanks for the extra characters. If the lengths of the character variables are different, then the values match as long as they are the same after stripping the trailing blanks. For example, 'ABCD' in the primary data set matches 'ABCD ' in the secondary data set. If the secondary data set contained 'ABCDEF', then they would not match.

## Base SAS Procedures That Support the BY Statement

|         |                 |
|---------|-----------------|
| COMPARE | SORT (required) |
| CORR    | STANDARD        |
| FREQ    | SUMMARY         |

|                                           |            |
|-------------------------------------------|------------|
| MEANS                                     | TABULATE   |
| PRINT                                     | TRANSPOSE  |
| RANK                                      | UNIVARIATE |
| REPORT (nonwindowing<br>environment only) | ODSTEXT    |
| ODSLIST                                   | ODSTABLE   |

---

**Note:** In the SORT procedure, the BY statement specifies how to sort the data. In the other procedures, the BY statement specifies how the data is currently sorted.

---

---

## See Also

[“BY Statement” in SAS DATA Step Statements: Reference](#)

---

# COPY Statement

Copies variables directly from the input data set to the output data set without transposing them.

Example: [“Example 6: Transposing Data for Statistical Analysis” on page 2405](#)

---

## Syntax

**COPY** *variable(s)*;

### Required Argument

***variable(s)***

names one or more variables that the COPY statement copies directly from the input data set to the output data set without transposing them.

---

## Details

Because the COPY statement copies variables directly to the output data set, the number of observations in the output data set is equal to the number of observations in the input data set.

The procedure pads the output data set with missing values if the number of observations in the input data set is not equal to the number of variables that it transposes.

---

## FORMAT Statement

Associates formats with variables.

---

### Syntax

**FORMAT** *variable-1* <...*variable-n*> *format variable-1* <...*variable-n*> *format*;

### Arguments

***variable***

names one or more variables for SAS to associate with a format. You must specify at least one *variable*.

**Tip** To disassociate a format from a variable, use the variable in a FORMAT statement without specifying a format in a DATA step or in PROC DATASETS. In a DATA step, place this FORMAT statement after the SET statement. See [“Removing a Format” in SAS DATA Step Statements: Reference](#). You can also use PROC DATASETS.

***format***

specifies the format that is listed for writing the values of the variables.

**Tip** Formats that are associated with variables by using a FORMAT statement behave like formats that are used with a colon modifier in a subsequent PUT statement. For more information about using a colon modifier, see [“PUT Statement: List” in SAS DATA Step Statements: Reference](#).

See [SAS Formats and Informats: Reference](#)

---

## ID Statement

Specifies one or more variables in the input data set whose nonmissing formatted values name the transposed variables in the output data set. When a variable name is being formed in the transposed (output) data set, the formatted values of all listed ID variables are concatenated in the same order that the variables are listed in the ID statement. The PREFIX=, DELIMITER=, and SUFFIX= options can be used to modify the formed variable name. The PREFIX= option specifies a common character or character string to appear at the beginning of the formed variable names. The DELIMITER= option specifies a common character or character string to be inserted between the values of the ID

variables. The SUFFIX= option specifies a common character or character string to be appended to the end of each formed variable name.

**Restriction:** You cannot use PROC TRANSPOSE with an ID statement or a BY statement with an engine that supports concurrent access if another user is updating the data set at the same time.

**Tip:** If the value of any ID variable is missing, then PROC TRANSPOSE writes a warning message to the log. The procedure does not transpose observations that have a missing value for any ID variable.

**Example:** [“Example 2: Naming Transposed Variables” on page 2397](#)

---

## Syntax

ID *variable(s)*;

### Required Argument

***variable(s)***

names one or more variables whose formatted values are used to form the names of the variables in the output data set.

---

## Details

### Duplicate Output Data Set Variable Names

A variable name formed from the input data set ID variable values, combined with the PREFIX, DELIMITER, and SUFFIX option values, should be unique within the output data set. An output data set variable name that occurs more than once indicates that two or more observations from the input data set are transposed to a single variable in the output data set and the result is data loss. This situation occurs when, in the case of a single ID variable, duplicate formatted values occur within the input data set or, if you use a BY statement, within a BY group. Similarly, this situation occurs in the case of multiple ID variables when the combination of formatted values of the ID variables occurs more than once within the input data set or BY group. To prevent data loss, if duplicate output data set variable names are formed, PROC TRANSPOSE issues an error message about duplicate ID values and stop processing. However, if the LET option is specified in the PROC TRANSPOSE statement then the procedure issues a warning message and continue processing, transposing the observation containing the last occurrence of the duplicate formatted variable values.

---

**Note:** The character substitutions and truncation required to ensure that the variable name formed from the ID variables is a valid SAS variable name, according to the VALIDVARNAME option setting, can result in duplicate output data set variable names even though the formatted value of the ID variable or combination of ID variables is unique within the input data set.

---



## Making Variable Names Out of Numeric Values

When you use a numeric variable as an ID variable, PROC TRANSPOSE changes the formatted ID value into a valid SAS name.

SAS variable names cannot begin with a number unless you have set VALIDVARNAME=ANY. When the first character of the formatted value is numeric, the procedure prefixes an underscore to the value, this action truncates the last character of a 32-character value. Remaining invalid characters are replaced by underscores. The procedure truncates to 32 characters any ID value that is longer than 32 characters when the procedure uses that value to name a transposed variable.

If the formatted value looks like a numeric constant, then PROC TRANSPOSE changes the characters +, -, and . to P, N, and D, respectively. If the formatted value has characters that are not numeric, then PROC TRANSPOSE changes the characters +, -, and . to underscores.

---

**Note:** If the value of the VALIDVARNAME system option is V6, then PROC TRANSPOSE truncates transposed variable names to 8 characters.

---

## Making Variable Names Out of Multiple ID Variables

When you specify a single ID variable, in forming an output data set variable name, the formatted values of the variable are made to conform with the SAS variable naming conventions imposed by the VALIDVARNAME option. The name formed by combining the ID variable values with the value of the PREFIX and SUFFIX options are also made to conform with the SAS variable naming convention. For both the formatted ID variable values and their combination with the PREFIX and SUFFIX options, invalid characters are replaced with underscores or, if the name appears to be a numeric constant, an underscore is used as a prefix and the characters +, -, and . are changed to P, N, and D. The resulting name is truncated to the maximum name length allowed by the VALIDVARNAME option setting. When you specify multiple ID variables, conformance to the SAS variable naming convention is imposed on the components of the variable name, using the formatted value of each ID variable, and also on the name composed from the ID variable values and the PREFIX, DELIMITER, and SUFFIX options. The resulting name is truncated to a length appropriate for the VALIDVARNAME option setting.

---

# IDLABEL Statement

Creates labels for the transposed variables.

Restriction: Must appear after an ID statement.

Example: [“Example 3: Labeling Transposed Variables” on page 2398](#)

## Syntax

**IDLABEL** *variable*;

### Required Argument

***variable***

names the variable whose values the procedure uses to label the variables that the ID statement names. *variable* can be character or numeric.

**Note:** To see the effect of the IDLABEL statement, print the output data set with the PRINT procedure by using the LABEL option. You can also print the contents of the output data set by using the CONTENTS statement in the DATASETS procedure.

## LABEL Statement

Assigns descriptive labels to variables.

## Syntax

**LABEL** *variable-1* = '*text-string*' <...*variable-n* = '*text-string*'>;

**LABEL** *variable-1* = '' <...*variable-n* = ''>; | *variable-1* = <...*variable-n* = >;

### Arguments

***variable***

specifies the variable that you want to label. You can specify labels for multiple variables in a single LABEL statement:

```
data test;
 L = 5;
 W = 10;
 label L = 'Length' W = 'Width';
run;
```

***text-string***

specifies a label of up to 256 bytes.

```
data test;
 L = 5;
 label L = 'Length';
run;
```

**Restrictions** If the label includes a semicolon (;) or an equal sign (=), you must enclose the label in either single or double quotation marks.

```
label N = 'Number = ';
```

If the label includes single quotation marks ('), you must enclose the label in double quotation marks.

```
label Name = "Owner's Last Name";
```

There is partial support for variable name substitution with the JAVAIMG device. When the label text for the variable exceeds the space allowed, the variable name is used instead by procedures that are labeling an axis or a legend title. With the exception of the SAS/GRAPH GPLOT procedure, the variable name is not used when the JAVAIMG device is specified.

**Notes** Quotation marks are not required unless the label includes a semicolon, quotation marks, or an equal sign.

The LABEL statement expects an equal sign (=) after the first variable. If any other character is found, an error occurs. The LABEL statement considers everything after the equal sign, following the first variable, as part of the label until the statement comes to another variable followed by an equal sign, regardless of quotation marks. In this example, the label for x is **this is x y 4 = this is y** because the next variable that is followed by an equal sign is z.

```
data test;
 x = 1;
 y = 1;
 z = 1;
 label x = 'this is x' y 4 = 'this is y' z = 'This is z';
run;
```

In this LABEL statement, the x variable has the same label:

```
label x = this is x y 4 = this is y z = This is z;
```

See [“Character Constants and Character Variables” in SAS Programmer’s Guide: Essentials.](#)

”

removes a label from a variable. Enclose a single blank space in quotation marks to remove an existing label. You can remove labels from multiple variables in a single LABEL statement:

```
data test;
 L=5; W=10;
 label L = ' ' W = ' ';
run;
```

Alternatively, you can remove a label by specifying nothing after the equal sign:

```
data test;
 L=5; W=10;
```

```
label L = W = ;
run;
```

---

## VAR Statement

Lists the variables to transpose.

**Note:** Add the special SAS variable list name `_CHARACTER_` to the VAR statement to include all character variables in the transposition. See [Special SAS Name Lists](#).

**Examples:**      [“Example 4: Transposing BY Groups” on page 2401](#)  
                      [“Example 6: Transposing Data for Statistical Analysis” on page 2405](#)

---

## Syntax

```
VAR variable(s);
```

### Required Argument

***variable(s)***

names one or more variables to transpose.

---

## Details

- If you omit the VAR statement, then the TRANSPOSE procedure transposes all numeric variables in the input data set that are not listed in another statement.
- You must list character variables in a VAR statement if you want to transpose them.

---

**Note:** If the procedure is transposing any character variable, then all transposed variables are character variables.

---



---

## WHERE Statement

Subsets the input data set by specifying certain conditions that each observation must meet before it is available for processing.

---

## Syntax

**WHERE** *where-expression-1*  
< *logical-operator where-expression-n*>;

### Arguments

***where-expression***

is an arithmetic or logical expression that generally consists of a sequence of operands and operators.

**Tips** The operands and operators described in the next several sections are also valid for the WHERE= data set option.

You can specify multiple where-expressions.

***logical-operator***

can be AND, AND NOT, OR, or OR NOT.

---

## Example

---

## Procedures That Support the WHERE Statement

You can use the WHERE statement with any of the following Base SAS procedures that read a SAS data set:

|                             |            |
|-----------------------------|------------|
| COMPARE                     | REPORT     |
| CORR                        | SORT       |
| DATASETS (APPEND statement) | SQL        |
| FREQ                        | STANDARD   |
| MEANS or SUMMARY            | TABULATE   |
| PRINT                       | TRANSPOSE  |
| RANK                        | UNIVARIATE |

---

---

## Details

- The COMPARE procedure and the APPEND statement in PROC DATASETS accept more than one input data set. For more information, see the documentation for the specific procedure.
- To subset the output data set, use the WHERE= data set option:

```
proc report data=debate nowd
 out=onlyfr(where=(year='1'));
run;
```

For more information about WHERE=, see [SAS DATA Step Statements: Reference](#).

---

## Temporarily Dissociating a Format from a Variable

In this example, PROC PRINT prints only those observations that meet the condition of the WHERE expression.

```
options nodate pageno=1 linesize=64
 pagesize=40;

proc print data=debate noobs;
 where gpa>3.5;
 title 'Team Members with a GPA';
 title2 'Greater than 3.5';
run;
```

---

## Output

**Team Members with a GPA  
Greater than 3.5**

| Name     | Gender | Year      | GPA   |
|----------|--------|-----------|-------|
| Capiccio | m      | Freshman  | 3.598 |
| Tucker   | m      | Freshman  | 3.901 |
| Bagwell  | f      | Sophomore | 3.722 |
| Gold     | f      | Junior    | 3.609 |
| Syme     | f      | Junior    | 3.883 |
| Baglione | f      | Senior    | 4.000 |
| Carr     | m      | Senior    | 3.750 |
| Hall     | m      | Senior    | 3.574 |

---

## Results: TRANSPOSE Procedure

---

### Output Data Set

The TRANSPOSE procedure always produces an output data set, regardless of whether you specify the OUT= option in the PROC TRANSPOSE statement. PROC TRANSPOSE does not print the output data set. Use PROC PRINT, PROC REPORT, or some other SAS reporting tool to print the output data set.

---

### Output Data Set Variables

The output data set contains the following variables:

- variables that result from transposing the values of each variable into an observation.
- a variable that PROC TRANSPOSE creates to identify the source of the values in each observation in the output data set. This variable is a character variable whose values are the names of the variables that are transposed from the input data set. By default, PROC TRANSPOSE names this variable `_NAME_`. To

override the default name, use the NAME= option. The label for the \_NAME\_ variable is **NAME OF FORMER VARIABLE**.

- variables that PROC TRANSPOSE copies from the input data set when you use either the BY or COPY statement. These variables have the same names and values as they do in the input data set. These variables also have the same attributes (for example: type, length, label, informat, and format).
- a character variable whose values are the variable labels of the variables that are being transposed (if any of the variables that the procedure is transposing have labels). Specify the name of the variable by using the LABEL= option. The default is \_LABEL\_.

---

**Note:** If the value of the LABEL= option or the NAME= option is the same as a variable that appears in a BY or COPY statement, then the output data set does not contain a variable whose values are the names or labels of the transposed variables.

---

## Attributes of Transposed Variables

- All transposed variables are the same type and length.
- If all variables that the procedure is transposing are numeric, then the transposed variables are numeric. Thus, if the numeric variable has a character string as a formatted value, then its unformatted numeric value is transposed.
- If any variable that the procedure is transposing is character, then all transposed variables are character. If you are transposing a numeric variable that has a character string as a formatted value, then the formatted value is transposed.
- The length of the transposed variables is equal to the length of the longest variable that is being transposed.

## Names of Transposed Variables

PROC TRANSPOSE uses the following rules to name transposed variables:

- 1 An ID statement specifies a variable or variables in the input data set whose formatted values become names for the transposed variables. If multiple ID variables are specified, the name of the transposed variable is the concatenation of the values of the ID variables. If the DELIMITER= option is specified, its value is inserted between the formatted values of the ID variables when the names of the transposed variables are formed.
- 2 The PREFIX= option specifies a prefix to use in constructing the names of transposed variables. The SUFFIX= option also specifies a suffix to append to the names of the transposed variables.



- 3 If you do not use an ID statement, PREFIX= option, or the SUFFIX= option, then PROC TRANSPOSE looks for an input variable called \_NAME\_ to get the names of the transposed variables.
- 4 If you do not use an ID statement or the PREFIX= option, and if the input data set does not contain a variable named \_NAME\_, then PROC TRANSPOSE assigns the names COL1, COL2, ..., COLn to the transposed variables.

---

## Examples: TRANSPOSE Procedure

---

### Example 1: Performing a Simple Transposition

Features: PROC TRANSPOSE statement option  
OUT=

---

This example performs a default transposition and uses no subordinate statements.

---

#### Program

```
options nodate pageno=1 linesize=80 pagesize=40;

data score;
 input Student $9. +1 StudentID $ Section $ Test1 Test2 Final;
 datalines;
Capalleti 0545 1 94 91 87
Dubose 1252 2 51 65 91
Engles 1167 1 95 97 97
Grant 1230 2 63 75 80
Krupski 2527 2 80 76 71
Lundsford 4860 1 92 40 86
McBane 0674 1 75 78 72
;

proc transpose data=score out=score_transposed;
run;

proc print data=score_transposed noobs;
 title 'Student Test Scores in Variables';
run;
```

## Program Description

**Set the SAS system options.** The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=40;
```

**Create the SCORE data set.** Set SCORE contains students' names, their identification numbers, and their grades on two tests and a final exam.

```
data score;
 input Student $9. +1 StudentID $ Section $ Test1 Test2 Final;
 datalines;
Capalleti 0545 1 94 91 87
Dubose 1252 2 51 65 91
Engles 1167 1 95 97 97
Grant 1230 2 63 75 80
Krupski 2527 2 80 76 71
Lundsford 4860 1 92 40 86
McBane 0674 1 75 78 72
;
```

**Transpose the data set.** PROC TRANSPOSE transposes only the numeric variables, Test1, Test2, and Final, because no VAR statement appears and none of the numeric variables appear in another statement. OUT= puts the result of the transposition in the data set SCORE\_TRANSPOSED.

```
proc transpose data=score out=score_transposed;
run;
```

**Print the SCORE\_TRANSPOSED data set.** The NOOBS option suppresses the printing of observation numbers

```
proc print data=score_transposed noobs;
 title 'Student Test Scores in Variables';
run;
```

## Output

In the output data set SCORE\_TRANSPOSED, the variables COL1 through COL7 contain the individual scores for the students. Each observation contains all the scores for one test. The variable \_NAME\_ contains the names of the variables from the input data set that were transposed.

**Output 62.3** Student Test Scores in Variables

| Student Test Scores in Variables |      |      |      |      |      |      |      |
|----------------------------------|------|------|------|------|------|------|------|
| _NAME_                           | COL1 | COL2 | COL3 | COL4 | COL5 | COL6 | COL7 |
| Test1                            | 94   | 51   | 95   | 63   | 80   | 92   | 75   |
| Test2                            | 91   | 65   | 97   | 75   | 76   | 40   | 78   |
| Final                            | 87   | 91   | 97   | 80   | 71   | 86   | 72   |

---

## Example 2: Naming Transposed Variables

Features: PROC TRANSPOSE statement options

NAME=

PREFIX=

ID statement

Data set: SCORE

This example uses the values of a variable and a user-supplied value to name transposed variables.

---

### Program

```
options nodate pageno=1 linesize=80 pagesize=40;

proc transpose data=score out=idnumber name=Test
 prefix=sn;
 id studentid;
run;

proc print data=idnumber noobs;
 title 'Student Test Scores';
run;
```

---

### Program Description

**Set the SAS system options.** The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=40;
```

**Transpose the data set.** PROC TRANSPOSE transposes only the numeric variables, Test1, Test2, and Final, because no VAR statement appears. OUT= puts the result of the transposition in the IDNUMBER data set. NAME= specifies Test as the name for

the variable that contains the names of the variables in the input data set that the procedure transposes. The procedure names the transposed variables by using the value from PREFIX=, sn, and the value of the ID variable StudentID.

```
proc transpose data=score out=idnumber name=Test
 prefix=sn;
 id studentid;
run;
```

**Print the IDNUMBER data set.** The NOOBS option suppresses the printing of observation numbers.

```
proc print data=idnumber noobs;
 title 'Student Test Scores';
run;
```

## Output

The following data set is the output data set, IDNUMBER.

**Output 62.4** Student Test Scores

| Student Test Scores |        |        |        |        |        |        |        |
|---------------------|--------|--------|--------|--------|--------|--------|--------|
| Test                | sn0545 | sn1252 | sn1167 | sn1230 | sn2527 | sn4860 | sn0674 |
| Test1               | 94     | 51     | 95     | 63     | 80     | 92     | 75     |
| Test2               | 91     | 65     | 97     | 75     | 76     | 40     | 78     |
| Final               | 87     | 91     | 97     | 80     | 71     | 86     | 72     |

## Example 3: Labeling Transposed Variables

Features: PROC TRANSPOSE statement option  
PREFIX=  
IDLABEL statement

Data set: SCORE

This example uses the values of the variable in the IDLABEL statement to label transposed variables.

## Program 1

```
options nodate pageno=1 linesize=80 pagesize=40;
proc transpose data=score out=idlabel name=Test
 prefix=sn;
```

```

 id studentid;

 idlabel student;
run;

proc print data=idlabel label noobs;
 title 'Student Test Scores';
run;

proc contents data=idlabel;
run;

```

---

## Program Description

**Set the SAS system options.** The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=40;
```

**Transpose the data set.** PROC TRANSPOSE transposes only the numeric variables, Test1, Test2, and Final, because no VAR statement appears. OUT= puts the result of the transposition in the IDLABEL data set. NAME= specifies Test as the name for the variable that contains the names of the variables in the input data set that the procedure transposes. The procedure names the transposed variables by using the value from PREFIX=, sn, and the value of the ID variable StudentID.

```
proc transpose data=score out=idlabel name=Test
 prefix=sn;
 id studentid;
```

**Assign labels to the output variables.** PROC TRANSPOSE uses the values of the variable Student to label the transposed variables. The procedure provides the following as the label for the \_NAME\_ variable: NAME OF FORMER VARIABLE

```
 idlabel student;
run;
```

**Print the IDLABEL data set.** The LABEL option causes PROC PRINT to print variable labels for column headings. The NOOBS option suppresses the printing of observation numbers.

```
proc print data=idlabel label noobs;
 title 'Student Test Scores';
run;
```

**Display the IDLABEL variable names and label.** PROC CONTENTS displays the variable names and labels.

```
proc contents data=idlabel;
run;
```

## Output 1

This data set is the output data set, IDLABEL.

**Output 62.5** Student Test Scores, IDLABEL

| Student Test Scores     |           |        |        |       |         |           |        |
|-------------------------|-----------|--------|--------|-------|---------|-----------|--------|
| NAME OF FORMER VARIABLE | Capalleti | Dubose | Engles | Grant | Krupski | Lundsford | McBane |
| Test1                   | 94        | 51     | 95     | 63    | 80      | 92        | 75     |
| Test2                   | 91        | 65     | 97     | 75    | 76      | 40        | 78     |
| Final                   | 87        | 91     | 97     | 80    | 71      | 86        | 72     |

## Program 2

**Display the variable and label names.** PROC CONTENTS displays the variable names and the labels used in the first program.

```
proc contents data=idlabel;
run;
```

## Output 2

In the following output, PROC CONTENTS displays the variables and labels.

**Output 62.6** The Contents Procedure

| Alphabetic List of Variables and Attributes |          |      |     |                         |
|---------------------------------------------|----------|------|-----|-------------------------|
| #                                           | Variable | Type | Len | Label                   |
| 1                                           | Test     | Char | 8   | NAME OF FORMER VARIABLE |
| 2                                           | sn0545   | Num  | 8   | Capalleti               |
| 8                                           | sn0674   | Num  | 8   | McBane                  |
| 4                                           | sn1167   | Num  | 8   | Engles                  |
| 5                                           | sn1230   | Num  | 8   | Grant                   |
| 3                                           | sn1252   | Num  | 8   | Dubose                  |
| 6                                           | sn2527   | Num  | 8   | Krupski                 |
| 7                                           | sn4860   | Num  | 8   | Lundsford               |

## Example 4: Transposing BY Groups

Features:

- BY statement
- VAR statement
- Data set option
- RENAME=

This example illustrates transposing BY groups and selecting variables to transpose.

### Program

```
options nodate pageno=1 linesize=80 pagesize=40;
data fishdata;
 infile datalines missover;
 input Location & $10. Date date7.
 Length1 Weight1 Length2 Weight2 Length3 Weight3
 Length4 Weight4;
 format date date7.;
 datalines;
Cole Pond 2JUN95 31 .25 32 .3 32 .25 33 .3
Cole Pond 3JUL95 33 .32 34 .41 37 .48 32 .28
Cole Pond 4AUG95 29 .23 30 .25 34 .47 32 .3
Eagle Lake 2JUN95 32 .35 32 .25 33 .30
Eagle Lake 3JUL95 30 .20 36 .45
Eagle Lake 4AUG95 33 .30 33 .28 34 .42
;

proc transpose data=fishdata
 out=fishlength(rename=(coll=Measurement));

 var length1-length4;

 by location date;
run;

proc print data=fishlength noobs;
 title 'Fish Length Data for Each Location and Date';
run;
```

### Program Description

**Create the FISHDATA data set and set the SAS system options.** The data in FISHDATA represents length and weight measurements of fish that were caught at two ponds on three separate days. The raw data is sorted by Location and Date. The NODATE option suppresses the display of the date and time in the output.

PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=40;
data fishdata;
 infile datalines missover;
 input Location & $10. Date date7.
 Length1 Weight1 Length2 Weight2 Length3 Weight3
 Length4 Weight4;
 format date date7.;
 datalines;
Cole Pond 2JUN95 31 .25 32 .3 32 .25 33 .3
Cole Pond 3JUL95 33 .32 34 .41 37 .48 32 .28
Cole Pond 4AUG95 29 .23 30 .25 34 .47 32 .3
Eagle Lake 2JUN95 32 .35 32 .25 33 .30
Eagle Lake 3JUL95 30 .20 36 .45
Eagle Lake 4AUG95 33 .30 33 .28 34 .42
;
```

**Transpose the data set.** OUT= puts the result of the transposition in the FISHLENGTH data set. RENAME= renames COL1 in the output data set to Measurement.

```
proc transpose data=fishdata
 out=fishlength(rename=(col1=Measurement));
```

**Specify the variables to transpose.** The VAR statement limits the variables that PROC TRANSPOSE transposes.

```
var length1-length4;
```

**Organize the output data set into BY groups.** The BY statement creates BY groups for each unique combination of values of Location and Date. The procedure does not transpose the BY variables.

```
by location date;
run;
```

**Print the FISHLENGTH data set.** The NOOBS option suppresses the printing of observation numbers.

```
proc print data=fishlength noobs;
 title 'Fish Length Data for Each Location and Date';
run;
```

---

## Output

The following data set is the output data set, FISHLENGTH. For each BY group in the original data set, PROC TRANSPOSE creates four observations, one for each variable that it is transposing. Missing values appear for the variable Measurement (renamed from COL1) when the variables that are being transposed have no value in the input data set for that BY group. Several observations have a missing value for Measurement. For example, in the last observation, a missing value appears because the input data contained no value for Length4 on 04AUG95 at Eagle Lake.



**Output 62.7** Fish Length Data**Fish Length Data for Each Location and Date**

| Location   | Date    | _NAME_  | Measurement |
|------------|---------|---------|-------------|
| Cole Pond  | 02JUN95 | Length1 | 31          |
| Cole Pond  | 02JUN95 | Length2 | 32          |
| Cole Pond  | 02JUN95 | Length3 | 32          |
| Cole Pond  | 02JUN95 | Length4 | 33          |
| Cole Pond  | 03JUL95 | Length1 | 33          |
| Cole Pond  | 03JUL95 | Length2 | 34          |
| Cole Pond  | 03JUL95 | Length3 | 37          |
| Cole Pond  | 03JUL95 | Length4 | 32          |
| Cole Pond  | 04AUG95 | Length1 | 29          |
| Cole Pond  | 04AUG95 | Length2 | 30          |
| Cole Pond  | 04AUG95 | Length3 | 34          |
| Cole Pond  | 04AUG95 | Length4 | 32          |
| Eagle Lake | 02JUN95 | Length1 | 32          |
| Eagle Lake | 02JUN95 | Length2 | 32          |
| Eagle Lake | 02JUN95 | Length3 | 33          |
| Eagle Lake | 02JUN95 | Length4 | .           |
| Eagle Lake | 03JUL95 | Length1 | 30          |
| Eagle Lake | 03JUL95 | Length2 | 36          |
| Eagle Lake | 03JUL95 | Length3 | .           |
| Eagle Lake | 03JUL95 | Length4 | .           |
| Eagle Lake | 04AUG95 | Length1 | 33          |
| Eagle Lake | 04AUG95 | Length2 | 33          |
| Eagle Lake | 04AUG95 | Length3 | 34          |
| Eagle Lake | 04AUG95 | Length4 | .           |

## Example 5: Naming Transposed Variables When the ID Variable Has Duplicate Values

Features: PROC TRANSPOSE statement option  
LET

This example shows how to use values of a variable (ID) to name transposed variables even when the ID variable has duplicate values.

## Program

```

options nodate pageno=1 linesize=64 pagesize=40;

data stocks;
 input Company $14. Date $ Time $ Price;
 datalines;
Horizon Kites jun11 opening 29
Horizon Kites jun11 noon 27
Horizon Kites jun11 closing 27
Horizon Kites jun12 opening 27
Horizon Kites jun12 noon 28
Horizon Kites jun12 closing 30
SkyHi Kites jun11 opening 43
SkyHi Kites jun11 noon 43
SkyHi Kites jun11 closing 44
SkyHi Kites jun12 opening 44
SkyHi Kites jun12 noon 45
SkyHi Kites jun12 closing 45
;

proc transpose data=stocks out=close let;
 by company;
 id date;
run;

proc print data=close noobs;
 title 'Closing Prices for Horizon Kites and SkyHi Kites';
run;

```

## Program Description

**Set the SAS system options.** The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=64 pagesize=40;
```

**Create the STOCKS data set.** STOCKS contains stock prices for two competing kite manufacturers. The prices are recorded for two days, three times a day: at opening, at noon, and at closing. Notice that the input data set contains duplicate values for the Date variable.

```

data stocks;
 input Company $14. Date $ Time $ Price;
 datalines;
Horizon Kites jun11 opening 29
Horizon Kites jun11 noon 27
Horizon Kites jun11 closing 27
Horizon Kites jun12 opening 27
Horizon Kites jun12 noon 28
Horizon Kites jun12 closing 30
SkyHi Kites jun11 opening 43

```

```

SkyHi Kites jun11 noon 43
SkyHi Kites jun11 closing 44
SkyHi Kites jun12 opening 44
SkyHi Kites jun12 noon 45
SkyHi Kites jun12 closing 45
;

```

**Transpose the data set.** LET transposes only the last observation for each BY group. PROC TRANSPOSE transposes only the Price variable. OUT= puts the result of the transposition in the CLOSE data set.

```
proc transpose data=stocks out=close let;
```

**Organize the output data set into BY groups.** The BY statement creates two BY groups, one for each company.

```
by company;
```

**Name the transposed variables.** The values of Date are used as names for the transposed variables.

```
id date;
run;
```

**Print the CLOSE data set.** The NOOBS option suppresses the printing of observation numbers..

```
proc print data=close noobs;
 title 'Closing Prices for Horizon Kites and SkyHi Kites';
run;
```

---

## Output

The following data set is the output data set, CLOSE.

### *Output 62.8* Closing Prices

#### Closing Prices for Horizon Kites and SkyHi Kites

| Company       | _NAME_ | jun11 | jun12 |
|---------------|--------|-------|-------|
| Horizon Kites | Price  | 27    | 30    |
| SkyHi Kites   | Price  | 44    | 45    |

---

## Example 6: Transposing Data for Statistical Analysis

Features: COPY statement  
VAR statement

---

This example arranges data to make it suitable for either a multivariate or a univariate repeated-measures analysis.

The data is from Chapter 8, “Repeated-Measures Analysis of Variance,” in SAS System for Linear Models, Third Edition.

---

## Program 1

```
options nodate pageno=1 linesize=80 pagesize=40;

data weights;
 input Program $ s1-s7;
 datalines;
CONT 85 85 86 85 87 86 87
CONT 80 79 79 78 78 79 78
CONT 78 77 77 77 76 76 77
CONT 84 84 85 84 83 84 85
CONT 80 81 80 80 79 79 80
RI 79 79 79 80 80 78 80
RI 83 83 85 85 86 87 87
RI 81 83 82 82 83 83 82
RI 81 81 81 82 82 83 81
RI 80 81 82 82 82 84 86
WI 84 85 84 83 83 83 84
WI 74 75 75 76 75 76 76
WI 83 84 82 81 83 83 82
WI 86 87 87 87 87 87 86
WI 82 83 84 85 84 85 86
;

data split;
 set weights;
 array s{7} s1-s7;
 Subject + 1;
 do Time=1 to 7;
 Strength=s{time};
 output;
 end;
 drop s1-s7;
run;

proc print data=split(obs=15) noobs;
 title 'SPLIT Data Set';
 title2 'First 15 Observations Only';
run;
```

---

## Program Description

**Set the SAS system options.** The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=40;
```

**Create the WEIGHTS data set.** The data in WEIGHTS represents the results of an exercise therapy study of three weight-lifting programs: CONT is a control group, RI is a program in which the number of repetitions is increased, and WI is a program in which the weight is increased.

```
data weights;
 input Program $ s1-s7;
 datalines;
CONT 85 85 86 85 87 86 87
CONT 80 79 79 78 78 79 78
CONT 78 77 77 77 76 76 77
CONT 84 84 85 84 83 84 85
CONT 80 81 80 80 79 79 80
RI 79 79 79 80 80 78 80
RI 83 83 85 85 86 87 87
RI 81 83 82 82 83 83 82
RI 81 81 81 82 82 83 81
RI 80 81 82 82 82 84 86
WI 84 85 84 83 83 83 84
WI 74 75 75 76 75 76 76
WI 83 84 82 81 83 83 82
WI 86 87 87 87 87 87 86
WI 82 83 84 85 84 85 86
;
```

**Create the SPLIT data set.** This DATA step rearranges WEIGHTS to create the data set SPLIT. The DATA step transposes the strength values and creates two new variables: Time and Subject. SPLIT contains one observation for each repeated measure. SPLIT can be used in a PROC GLM step for a univariate repeated-measures analysis.

```
data split;
 set weights;
 array s{7} s1-s7;
 Subject + 1;
 do Time=1 to 7;
 Strength=s{time};
 output;
 end;
 drop s1-s7;
run;
```

**Print the SPLIT data set.** The NOOBS options suppresses the printing of observation numbers. The OBS= data set option limits the printing to the first 15 observations. SPLIT has 105 observations.

```
proc print data=split(obs=15) noobs;
 title 'SPLIT Data Set';
 title2 'First 15 Observations Only';
run;
```

---

## Output 1

**Output 62.9** *Split Data Set*

| SPLIT Data Set<br>First 15 Observations Only |         |      |          |
|----------------------------------------------|---------|------|----------|
| Program                                      | Subject | Time | Strength |
| CONT                                         | 1       | 1    | 85       |
| CONT                                         | 1       | 2    | 85       |
| CONT                                         | 1       | 3    | 86       |
| CONT                                         | 1       | 4    | 85       |
| CONT                                         | 1       | 5    | 87       |
| CONT                                         | 1       | 6    | 86       |
| CONT                                         | 1       | 7    | 87       |
| CONT                                         | 2       | 1    | 80       |
| CONT                                         | 2       | 2    | 79       |
| CONT                                         | 2       | 3    | 79       |
| CONT                                         | 2       | 4    | 78       |
| CONT                                         | 2       | 5    | 78       |
| CONT                                         | 2       | 6    | 79       |
| CONT                                         | 2       | 7    | 78       |
| CONT                                         | 3       | 1    | 78       |

---

## Program 2

```
options nodate pageno=1 linesize=80 pagesize=40;
proc transpose data=split out=totsplit prefix=Str;
 by program subject;
 copy time strength;

 var strength;
run;

proc print data=totsplit(obs=15) noobs;
 title 'TOTSPPLIT Data Set';
 title2 'First 15 Observations Only';
run;
```

---

## Program Description

### Set the SAS system options.

```
options nodate pageno=1 linesize=80 pagesize=40;
```

**Transpose the SPLIT data set.** PROC TRANSPOSE transposes SPLIT to create TOTSPLIT. The TOTSPLIT data set contains the same variables as SPLIT and a variable for each strength measurement (Str1-Str7). TOTSPLIT can be used for either a multivariate repeated-measures analysis or a univariate repeated-measures analysis.

```
proc transpose data=split out=totsplit prefix=Str;
```

**Organize the output data set into BY groups, and populate each BY group with untransposed values.** The variables in the BY and COPY statements are not transposed. TOTSPLIT contains the variables Program, Subject, Time, and Strength with the same values that are in SPLIT. The BY statement creates the first observation in each BY group, which contains the transposed values of Strength. The COPY statement creates the other observations in each BY group by copying the values of Time and Strength without transposing them.

```
 by program subject;
 copy time strength;
```

**Specify the variable to transpose.** The VAR statement specifies the Strength variable as the only variable to be transposed.

```
 var strength;
run;
```

**Print the TOTSPLIT data set.** The NOOBS options suppresses the printing of observation numbers. The OBS= data set option limits the printing to the first 15 observations. SPLIT has 105 observations.

```
proc print data=totsplit(obs=15) noobs;
 title 'TOTSPLIT Data Set';
 title2 'First 15 Observations Only';
run;
```

---

## Output 2

In the following output, the variables in TOTSPLIT with missing values are used only in a multivariate repeated-measures analysis. The missing values do not preclude this data set from being used in a repeated-measures analysis because the MODEL statement in PROC GLM ignores observations with missing values.

**Output 62.10** TOTSPLIT Data Set

**TOTSPLIT Data Set**  
**First 15 Observations Only**

| Program | Subject | Time | Strength | _NAME_   | Str1 | Str2 | Str3 | Str4 | Str5 | Str6 | Str7 |
|---------|---------|------|----------|----------|------|------|------|------|------|------|------|
| CONT    | 1       | 1    | 85       | Strength | 85   | 85   | 86   | 85   | 87   | 86   | 87   |
| CONT    | 1       | 2    | 85       |          | .    | .    | .    | .    | .    | .    | .    |
| CONT    | 1       | 3    | 86       |          | .    | .    | .    | .    | .    | .    | .    |
| CONT    | 1       | 4    | 85       |          | .    | .    | .    | .    | .    | .    | .    |
| CONT    | 1       | 5    | 87       |          | .    | .    | .    | .    | .    | .    | .    |
| CONT    | 1       | 6    | 86       |          | .    | .    | .    | .    | .    | .    | .    |
| CONT    | 1       | 7    | 87       |          | .    | .    | .    | .    | .    | .    | .    |
| CONT    | 2       | 1    | 80       | Strength | 80   | 79   | 79   | 78   | 78   | 79   | 78   |
| CONT    | 2       | 2    | 79       |          | .    | .    | .    | .    | .    | .    | .    |
| CONT    | 2       | 3    | 79       |          | .    | .    | .    | .    | .    | .    | .    |
| CONT    | 2       | 4    | 78       |          | .    | .    | .    | .    | .    | .    | .    |
| CONT    | 2       | 5    | 78       |          | .    | .    | .    | .    | .    | .    | .    |
| CONT    | 2       | 6    | 79       |          | .    | .    | .    | .    | .    | .    | .    |
| CONT    | 2       | 7    | 78       |          | .    | .    | .    | .    | .    | .    | .    |
| CONT    | 3       | 1    | 78       | Strength | 78   | 77   | 77   | 77   | 76   | 76   | 77   |



# XSL Procedure

---

|                                                                                    |             |
|------------------------------------------------------------------------------------|-------------|
| <b>Overview: XSL Procedure</b> .....                                               | <b>2411</b> |
| What Does the Extensible Style Sheet Language (XSL) Procedure Do? .....            | 2411        |
| Understanding XSL .....                                                            | 2412        |
| <b>Syntax: XSL Procedure</b> .....                                                 | <b>2412</b> |
| PROC XSL Statement .....                                                           | 2412        |
| PARAMETER Statement .....                                                          | 2414        |
| <b>Examples: XSL Procedure</b> .....                                               | <b>2415</b> |
| Example 1: Transforming an XML Document into Another XML Document .....            | 2415        |
| Example 2: Passing a Character String Parameter Value to the XSL Style Sheet ..... | 2416        |
| Example 3: Passing a Numeric Parameter Value to the XSL Style Sheet .....          | 2420        |

---

## Overview: XSL Procedure

---

### What Does the Extensible Style Sheet Language (XSL) Procedure Do?

The XSL procedure transforms an XML document into another format, such as HTML, text, or another XML document type. PROC XSL reads an input XML document, transforms it by using an XSL style sheet, and then writes the output.

To transform the XML document, PROC XSL uses the Saxon-EE version 9.3 software application from Saxonica, which is a collection of tools for processing XML documents. The XSLT processor implements the XSLT 2.0 standard. For information about Saxon, see the website [About Saxon](#).

# Understanding XSL

XSL is a family of transformation languages that enables you to describe how to convert files that are encoded in XML. The languages include the following:

- XSL Transformations (XSLT) for transforming an XML document
- XML Path Language (XPath), which is used by XSLT, for selecting parts of an XML document

For information about XSLT standards, see the website [XSL Transformations \(XSLT\) Version 2.0](#).

# Syntax: XSL Procedure

Restriction: Beginning in 2024.12, the XSL procedure is disabled by default when SAS is locked down. To allow use of the procedure in SAS Viya, the administrator must specify `ENABLE_AMS=XSL` in the `LOCKDOWN` statement. For more information, see “[LOCKDOWN](#)” in *SAS Programmer’s Guide: Essentials* and “[Example 5: Enabling PROC XSL While in Lockdown Mode](#)” in *SAS Viya Platform: Programming Run-Time Servers*.

```
PROC XSL IN=fileref | 'external-file' OUT=fileref | 'external-file' XSL=fileref | 'external-file';
```

```
PARAMETER 'parameter'='value';
```

| Statement | Task                                                   | Example                                                               |
|-----------|--------------------------------------------------------|-----------------------------------------------------------------------|
| PROC XSL  | Transform an XML document                              | <a href="#">Ex. 1</a> , <a href="#">Ex. 2</a> , <a href="#">Ex. 3</a> |
| PARAMETER | Pass a parameter to the XSL style sheet to set a value | <a href="#">Ex. 2</a> , <a href="#">Ex. 3</a>                         |

# PROC XSL Statement

Transforms an XML document.

Note: The XSL procedure supports SAS LOCKDOWN restrictions. For more information, see “[LOCKDOWN](#)” in *SAS Programmer’s Guide: Essentials*.

Example: [“Example 1: Transforming an XML Document into Another XML Document” on page 2415](#)

## Syntax

```
PROC XSL IN=fileref | 'external-file' OUT=fileref | 'external-file' XSL=fileref | 'external-file';
```

### Required Arguments

#### **IN=fileref** | *'external-file'*

specifies the input file. The file must be a well-formed XML document.

##### *fileref*

specifies the SAS fileref that is assigned to the input XML document. To assign a fileref, use the FILENAME statement.

##### *'external-file'*

is the physical location of the input XML document. Include the complete pathname and the filename. Enclose the physical name in single or double quotation marks. The maximum length is 200 characters.

**Restriction** When the LOCKDOWN option or NOXCMD option is set on a session, any operating system commands that are specified in the XSL file fail with an ERROR.

**Example** [“Example 1: Transforming an XML Document into Another XML Document” on page 2415](#)

#### **OUT=fileref** | *'external-file'*

specifies the output file.

##### *fileref*

specifies the SAS fileref that is assigned to the output file. To assign a fileref, use the FILENAME statement.

##### *'external-file'*

is the physical location of the output file. Include the complete pathname and the filename. Enclose the physical name in single or double quotation marks. The maximum length is 200 characters.

**Example** [“Example 1: Transforming an XML Document into Another XML Document” on page 2415](#)

#### **XSL=fileref** | *'external-file'*

specifies the XSL style sheet to transform the XML document. The XSL style sheet is a file that describes how to transform the XML document by using the XSLT language. The XSL style sheet must be a well-formed XML document.

##### *fileref*

specifies the SAS fileref that is assigned to the XSL style sheet. To assign a fileref, use the FILENAME statement.

**'external-file'**

is the physical location of the XSL style sheet. Include the complete pathname and the filename. Enclose the physical name in single or double quotation marks. The maximum length is 200 characters.

Alias      XSLT=

Example    [“Example 1: Transforming an XML Document into Another XML Document” on page 2415](#)

---

## PARAMETER Statement

Changes instances of a parameter in an XSL style sheet to a specified value.

Examples:      [“Example 2: Passing a Character String Parameter Value to the XSL Style Sheet” on page 2416](#)  
                   [“Example 3: Passing a Numeric Parameter Value to the XSL Style Sheet” on page 2420](#)

---

## Syntax

**PARAMETER** '*parameter*'=*value*;

### Required Argument

**'parameter'='value'**

specifies the parameter name and a value to be passed to the XSL style sheet. PROC XSL changes instances of the parameter in the style sheet to the specified value. The specified parameter must exist in the XSL style sheet. Enclose the parameter name in single or double quotation marks. The specified value can be a character string or a numeric value. Enclose the value in single or double quotation marks if the value is a character string.

---

# Examples: XSL Procedure

---

## Example 1: Transforming an XML Document into Another XML Document

Features: PROC XSL statement

---

---

### Details

The following PROC XSL example transforms an XML document into another XML document.

This is the input XML document named XMLInput.xml, which contains data about vehicles. Each second-level repeating element describes a particular car, with the nested elements that contain information about the model and year. The make information is an attribute on the second-level repeating element.

```
<?xml version="1.0" ?>
<vehicles>
 <car make="Ford">
 <model>Mustang</model>
 <year>1965</year>
 </car>
 <car make="Chevrolet">
 <model>Nova</model>
 <year>1967</year>
 </car>
</vehicles>
```

This is the XSL style sheet named XSLTransform.xsl that describes how to transform the XML. The conversion creates <root> as the root-enclosing element and <model> as the second-level repeating element. Each <model> element in the output XML document will include the values from the <car> element and the make= attribute from the input XML document.

```
<?xml version="1.0" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="xml" indent="yes"/>
```

```

<xsl:template match="/vehicles">
 <root> <xsl:apply-templates select="car"/> </root>
</xsl:template>

<xsl:template match="car">
 <model make="{@make}">
 <xsl:value-of select="model" />
 </model>
</xsl:template>

</xsl:stylesheet>

```

---

## Program

**Execute the PROC XSL statement.** The PROC XSL statement specifies the input XML document, the XSL style sheet, and the output XML document.

```

proc xsl
 in='file-path/XmlInput.xml'
 xsl='file-path/XslTransform.xsl'
 out='file-path/XmlOutput.xml';
run;

```

---

## Output: Transforming an XML Document into Another XML Document

**Output 63.1** PROC XSL Output of Transformed XML Document

```

<?xml version="1.0" encoding="UTF-8"?>
<root>
<model make="Ford">Mustang</model>
<model make="Chevrolet">Nova</model>
</root>

```

---

## Example 2: Passing a Character String Parameter Value to the XSL Style Sheet

Features: PROC XSL statement  
PARAMETER statement

---

## Details

This example shows how to use PROC XSL to pass a character string value to a parameter in an XSL style sheet. A parameter, which is a named variable in the style sheet for which you can set a value, is an easy way to customize generated output.

This is the XSL style sheet named Format.xsl. The XSL style sheet extracts elements and attributes from an input XML document to generate HTML output. The style sheet contains the declared parameter DateVar.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:param name="DateVar"/>
 <xsl:template match="TABLE">
 <html>
 <head/>
 <body>
 <h2><xsl:value-of select="$DateVar"/></h2>
 <table border="1" rules="all">
 <tr>
 <td>Name</td>
 <td>Sex</td>
 <td>Age</td>
 <td>Height</td>
 <td>Weight</td>
 </tr>
 <xsl:apply-templates/>
 </table>
 </body>
 </html>
 </xsl:template>
 <xsl:template match="CLASS">
 <tr>
 <td><xsl:value-of select="Name"/></td>
 <td><xsl:value-of select="Sex"/></td>
 <td><xsl:value-of select="Age"/></td>
 <td><xsl:value-of select="Height"/></td>
 <td><xsl:value-of select="Weight"/></td>
 </tr>
 </xsl:template>
</xsl:stylesheet>
```

The input XML document named Class.xml contains classroom data.

```
<?xml version="1.0" encoding="windows-1252" ?>
<TABLE>
 <CLASS>
 <Name> Alfred </Name>
 <Sex> M </Sex>
 <Age> 14 </Age>
 <Height> 69 </Height>
 <Weight> 112.5 </Weight>
```

```

</CLASS>
<CLASS>
 <Name> Alice </Name>
 <Sex> F </Sex>
 <Age> 13 </Age>
 <Height> 56.5 </Height>
 <Weight> 84 </Weight>
</CLASS>
<CLASS>
 <Name> Barbara </Name>
 <Sex> F </Sex>
 <Age> 13 </Age>
 <Height> 65.3 </Height>
 <Weight> 98 </Weight>
</CLASS>
.
.
.
<CLASS>
 <Name> Thomas </Name>
 <Sex> M </Sex>
 <Age> 11 </Age>
 <Height> 57.5 </Height>
 <Weight> 85 </Weight>
</CLASS>
<CLASS>
 <Name> William </Name>
 <Sex> M </Sex>
 <Age> 15 </Age>
 <Height> 66.5 </Height>
 <Weight> 112 </Weight>
</CLASS>
</TABLE>

```

---

## Program

```

proc xsl
 in='file-path/Class.xml'
 xsl='file-path/Format.xsl'
 out='file-path/Class.html';

 parameter 'DateVar'="Report Date: &sysdate";
run;

```

---

## Program Description

**Execute the PROC XSL statement.** The PROC XSL statement specifies the input XML document, the XSL style sheet, and the output HTML file.

```
proc xsl
```



```
in='file-path/Class.xml'
xsl='file-path/Format.xsl'
out='file-path/Class.html';
```

**Pass the parameter value to the XSL style sheet.** The PARAMETER statement passes a character string value for the parameter DateVar to the XSL style sheet. Note that because the value includes a macro variable reference, the value must be in double quotation marks.

```
parameter 'DateVar'="Report Date: &sysdate";
run;
```

## Output: Passing a Character String Parameter Value to the XSL Style Sheet

*Output 63.2 PROC XSL Output Class.html*

### Report Date: 15NOV12

Name	Sex	Age	Height	Weight
Alfred	M	14	69	112.5
Alice	F	13	56.5	84
Barbara	F	13	65.3	98
Carol	F	14	62.8	102.5
Henry	M	14	63.5	102.5
James	M	12	57.3	83
Jane	F	12	59.8	84.5
Janet	F	15	62.5	112.5
Jeffrey	M	13	62.5	84
John	M	12	59	99.5
Joyce	F	11	51.3	50.5
Judy	F	14	64.3	90
Louise	F	12	56.3	77
Mary	F	15	66.5	112
Philip	M	16	72	150
Robert	M	12	64.8	128
Ronald	M	15	67	133
Thomas	M	11	57.5	85
William	M	15	66.5	112

---

## Example 3: Passing a Numeric Parameter Value to the XSL Style Sheet

Features: PROC XSL statement  
PARAMETER statement

---

---

### Details

This example shows how to use PROC XSL to pass a numeric value to a parameter in an XSL style sheet.

This is the XSL style sheet named Discount.xsl. The XSL style sheet extracts elements and attributes from an input XML document to generate XML output. The style sheet contains the declared parameter DiscountPct. PROC XSL passes in a numeric value to the parameter, and a discount amount is calculated.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:output method="xml" indent="yes" />
 <xsl:param name="DiscountPct" />

 <xsl:template match="table">
 <xsl:copy>
 <xsl:apply-templates select="product" />
 </xsl:copy>
 </xsl:template>

 <xsl:template match="product">
 <xsl:copy>
 <xsl:copy-of select="category|type|size|gender|price" />
 <discount>
 <xsl:value-of select="$DiscountPct" />
 </discount>
 <discountAmount>
 <xsl:value-of select="(price * $DiscountPct)" />
 </discountAmount>
 <xsl:copy-of select="in_stock|ship_type" />
 </xsl:copy>
 </xsl:template>

</xsl:stylesheet>
```

The input XML document named Product.xml contains product data:

```
<?xml version="1.0"?>
<table>
```

```
<product>
 <category>shoe</category>
 <type>Nike Air</type>
 <size>12</size>
 <gender>male</gender>
 <price>99</price>
 <in_stock>yes</in_stock>
 <ship_type>overnight</ship_type>
</product>
</table>
```

---

## Program

```
proc xsl
 in='file-path/Product.xml'
 xsl='file-path/Discount.xsl'
 out='file-path/Calculated.xml';

 parameter 'DiscountPct'=.20;
run;
```

---

## Program Description

**Execute the PROC XSL statement.** The PROC XSL statement specifies the input XML document, the XSL style sheet, and the output XML document.

```
proc xsl
 in='file-path/Product.xml'
 xsl='file-path/Discount.xsl'
 out='file-path/Calculated.xml';
```

**Pass the parameter to the XSL style sheet.** The PARAMETER statement passes a numeric value for the parameter DiscountPct to the XSL style sheet. The output XML document will contain the discount value and a calculated discount amount.

```
 parameter 'DiscountPct'=.20;
run;
```

## Output: Passing a Numeric Parameter Value to the XSL Style Sheet

### **Output 63.3** *PROC XSL Output Calculated.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<table>
 <product>
 <category>shoe</category>
 <type>Nike Air</type>
 <size>12</size>
 <gender>male</gender>
 <price>99</price>
 <discount>0.2</discount>
 <discountAmount>19.8</discountAmount>
 <in_stock>yes</in_stock>
 <ship_type>overnight</ship_type>
 </product>
</table>
```

**PART 3**

# Appendixes

<i>Appendix 1</i>	
<b><i>SAS Elementary Statistics Procedures</i></b> .....	<b>2425</b>
<i>Appendix 2</i>	
<b><i>Raw Data and DATA Steps for Base SAS Procedures</i></b> .....	<b>2469</b>
<i>Appendix 3</i>	
<b><i>UNICODE License</i></b> .....	<b>2549</b>



# Appendix 1

## SAS Elementary Statistics Procedures

---

<i>Overview of SAS Elementary Statistics Procedures</i> .....	<b>2426</b>
<i>Keywords and Formulas</i> .....	<b>2426</b>
Simple Statistics .....	2426
Descriptive Statistics .....	2429
Quantile and Related Statistics .....	2432
Hypothesis Testing Statistics .....	2434
Confidence Limits for the Mean .....	2434
Using Weights .....	2435
Data Requirements for Summarization Procedures .....	2436
<i>Statistical Background</i> .....	<b>2436</b>
Populations and Parameters .....	2436
Samples and Statistics .....	2437
Measures of Location .....	2438
Percentiles .....	2439
Quantiles .....	2439
Measures of Variability .....	2444
Measures of Shape .....	2445
The Normal Distribution .....	2446
Sampling Distribution of the Mean .....	2450
Testing Hypotheses .....	2463
.....	<b>2468</b>

---

# Overview of SAS Elementary Statistics Procedures

This appendix provides a brief description of some of the statistical concepts necessary for you to interpret the output of Base SAS procedures for elementary statistics. In addition, this appendix lists statistical notation, formulas, and standard keywords used for common statistics in Base SAS procedures. Brief examples illustrate the statistical concepts.

[Table A1.1 on page 2427](#) lists the most common statistics and the procedures that compute them.

---

## Keywords and Formulas

---

### Simple Statistics

The Base SAS procedures use a standardized set of keywords to refer to statistics. You specify these keywords in SAS statements to request the statistics to be displayed or stored in an output data set.

In the following notation, summation is over observations that contain nonmissing values of the analyzed variable and, except where shown, over nonmissing weights and frequencies of one or more:

$x_i$

is the nonmissing value of the analyzed variable for observation  $i$ .

$f_i$

is the frequency that is associated with  $x_i$  if you use a FREQ statement. If you omit the FREQ statement, then  $f_i = 1$  for all  $i$ .

$w_i$

is the weight that is associated with  $x_i$  if you use a WEIGHT statement. The base procedures automatically exclude the values of  $x_i$  with missing weights from the analysis.

By default, the base procedures treat a negative weight as if it is equal to zero. However, if you use the EXCLNPWGT option in the PROC statement, then the procedure also excludes those values of  $x_i$  with nonpositive weights. Note that



most SAS/STAT procedures, such as PROC TTEST and PROC GLM, exclude values with nonpositive weights by default.

If you omit the WEIGHT statement, then  $w_i = 1$  for all  $i$ .

$n$

is the number of nonmissing values of  $x_i$ ,  $\sum f_i$ . If you use the EXCLNPWGT option and the WEIGHT statement, then  $n$  is the number of nonmissing values with positive weights.

$\bar{x}$

is the mean

$$\sum w_i x_i / \sum w_i$$

$s^2$

is the variance

$$\frac{1}{d} \sum w_i (x_i - \bar{x})^2$$

where  $d$  is the variance divisor (the VARDEF= option) that you specify in the PROC statement. Valid values are as follows:

When VARDEF=	$d$ equals
N	$n$
DF	$n - 1$
WEIGHT	$\sum w_i$
WDF	$\sum w_i - 1$

The default is DF.

$z_i$

is the standardized variable

$$(x_i - \bar{x})/s$$

The standard keywords and formulas for each statistic follow. Some formulas use keywords to designate the corresponding statistic.

**Table A57.1** The Most Common Simple Statistics

Statistic	PROC MEANS and SUMMARY	PROC UNIVARIATE	PROC TABULATE	PROC REPORT	PROC CORR	PROC SQL
Number of missing values	X	X	X	X		X

Statistic		PROC MEANS and SUMMARY	PROC UNIVARIATE	PROC TABULATE	PROC REPORT	PROC CORR	PROC SQL
Number of nonmissing values		X	X	X	X	X	X
Number of observations		X	X				X
Sum of weights		X	X	X	X	X	X
Mean		X	X	X	X	X	X
Sum		X	X	X	X	X	X
Extreme values		X	X				
Minimum		X	X	X	X	X	X
Maximum		X	X	X	X	X	X
Range		X	X	X	X		X
Uncorrected sum of squares		X	X	X	X	X	X
Corrected sum of squares		X	X	X	X	X	X
Variance		X	X	X	X	X	X
Covariance						X	
Standard deviation		X	X	X	X	X	X
Standard error of the mean		X	X	X	X		X
Coefficient of variation		X	X	X	X		X
Skewness		X	X	X			
Kurtosis		X	X	X			
Confidence Limits							
	of the mean	X	X	X			
	of the variance		X				
	of quantiles		X				
Median		X	X	X	X	X	

Statistic	PROC MEANS and SUMMARY	PROC UNIVARIATE	PROC TABULATE	PROC REPORT	PROC CORR	PROC SQL
Mode	X	X	X	X		
Percentiles/Deciles/ Quartiles	X	X	X	X		
t test						
for mean=0	X	X	X	X		X
for mean= $\mu_0$		X				
Nonparametric tests for location		X				
Tests for normality		X				
Correlation coefficients					X	
Cronbach's alpha					X	

## Descriptive Statistics

The keywords for descriptive statistics are

CSS

is the sum of squares corrected for the mean, computed as

$$\sum w_i(x_i - \bar{x})^2$$

CV

is the percent coefficient of variation, computed as

$$(100s)/\bar{x}$$

KURTOSIS | KURT

is the kurtosis, which measures heaviness of tails. When VARDEF=DF, the kurtosis is computed as

$$c_{4n} \sum z_i^4 - \frac{3(n-1)^2}{(n-2)(n-3)}$$

where  $c_{4n}$  is  $\frac{n(n+1)}{(n-1)(n-2)(n-3)}$ . The weighted kurtosis is computed as

$$\begin{aligned}
&= c_{4n} \sum ((x_i - \bar{x})/\hat{\sigma}_i)^4 - \frac{3(n-1)^2}{(n-2)(n-3)} \\
&= c_{4n} \sum w_i^2 ((x_i - \bar{x})/\hat{\sigma})^4 - \frac{3(n-1)^2}{(n-2)(n-3)}
\end{aligned}$$

When VARDEF=N, the kurtosis is computed as

$$= \frac{1}{n} \sum z_i^4 - 3$$

and the weighted kurtosis is computed as

$$\begin{aligned}
&= \frac{1}{n} \sum ((x_i - \bar{x})/\hat{\sigma}_i)^4 - 3 \\
&= \frac{1}{n} \sum w_i^2 ((x_i - \bar{x})/\hat{\sigma})^4 - 3
\end{aligned}$$

where  $\sigma_i^2$  is  $\sigma^2/w_i$ . The formula is invariant under the transformation  $w_i^* = zw_i$ ,  $z > 0$ . When you use VARDEF=WDF or VARDEF=WEIGHT, the kurtosis is set to missing.

---

**Note:** PROC MEANS and PROC TABULATE do not compute weighted kurtosis.

---

MAX

is the maximum value of  $x_i$ .

MEAN

is the arithmetic mean  $\bar{x}$ .

MIN

is the minimum value of  $x_i$ .

MODE

is the most frequent value of  $x_i$ .

---

**Note:** When QMETHOD=P2, PROC REPORT, PROC MEANS, and PROC TABULATE do not compute MODE.

---

N

is the number of  $x_i$  values that are not missing. Observations with  $f_i$  less than one and  $w_i$  equal to missing or  $w_i \leq 0$  (when you use the EXCLNPWGT option) are excluded from the analysis and are not included in the calculation of N.

NMISS

is the number of  $x_i$  values that are missing. Observations with  $f_i$  less than one and  $w_i$  equal to missing or  $w_i \leq 0$  (when you use the EXCLNPWGT option) are excluded from the analysis and are not included in the calculation of NMISS.

NOBS

is the total number of observations and is calculated as the sum of N and NMISS. However, if you use the WEIGHT statement, then NOBS is calculated as the sum of N, NMISS, and the number of observations excluded because of missing or nonpositive weights.

**RANGE**

is the range and is calculated as the difference between maximum value and minimum value.

**SKEWNESS | SKEW**

is skewness, which measures the tendency of the deviations to be larger in one direction than in the other. When VARDEF=DF, the skewness is computed as

$$c_{3n} \sum z_i^3$$

where  $c_{3n}$  is  $\frac{n}{(n-1)(n-2)}$ . The weighted skewness is computed as

$$\begin{aligned} &= c_{3n} \sum ((x_i - \bar{x})/\hat{\sigma}_j)^3 \\ &= c_{3n} \sum w_i^{3/2} ((x_i - \bar{x})/\hat{\sigma})^3 \end{aligned}$$

When VARDEF=N, the skewness is computed as

$$= \frac{1}{n} \sum z_i^3$$

and the weighted skewness is computed as

$$\begin{aligned} &= \frac{1}{n} \sum ((x_i - \bar{x})/\hat{\sigma}_j)^3 \\ &= \frac{1}{n} \sum w_i^{3/2} ((x_i - \bar{x})/\hat{\sigma})^3 \end{aligned}$$

The formula is invariant under the transformation  $w_i^* = zw_i$ ,  $z > 0$ . When you use VARDEF=WDF or VARDEF=WEIGHT, the skewness is set to missing.

---

**Note:** PROC MEANS and PROC TABULATE do not compute weighted skewness.

---

**STDDEV | STD**

is the standard deviation  $s$  and is computed as the square root of the variance,  $s^2$ .

**STDERR | STDMEAN**

is the standard error of the mean, computed as

$$s/\sqrt{\sum w_i}$$

when VARDEF=DF, which is the default. Otherwise, STDERR is set to missing.

**SUM**

is the sum, computed as

$$\sum w_i x_i$$

**SUMWGT**

is the sum of the weights,  $W$ , computed as

$$\sum w_i$$

USS

is the uncorrected sum of squares, computed as

$$\sum w_i x_i^2$$

VAR

is the variance  $s^2$ .

---

## Quantile and Related Statistics

The keywords for quantiles and related statistics are

MEDIAN

is the middle value.

P1

is the 1<sup>st</sup> percentile.

P5

is the 5<sup>th</sup> percentile.

P10

is the 10<sup>th</sup> percentile.

P90

is the 90<sup>th</sup> percentile.

P95

is the 95<sup>th</sup> percentile.

P99

is the 99<sup>th</sup> percentile.

Q1

is the lower quartile (25<sup>th</sup> percentile).

Q3

is the upper quartile (75<sup>th</sup> percentile).

QRANGE

is interquartile range and is calculated as

$$Q_3 - Q_1$$

You use the QNTLDEF= option (PCTLDEF= in PROC UNIVARIATE) to specify the method that the procedure uses to compute percentiles. Let  $n$  be the number of nonmissing values for a variable, and let  $x_1, x_2, \dots, x_n$  represent the ordered values of the variable such that  $x_1$  is the smallest value,  $x_2$  is next smallest value, and  $x_n$  is the largest value. For the  $t$ th percentile between 0 and 1, let  $p = t/100$ . Then specify  $j$  as the integer part of  $np$  and  $g$  as the fractional part of  $np$  or  $(n+1)p$ , so that

$$\begin{array}{ll} np = j + g & \text{when QNTLDEF} = 1, 2, 3, \text{ or } 5 \\ (n+1)p = j + g & \text{when QNTLDEF} = 4 \end{array}$$

Here, QNTLDEF= specifies the method that the procedure uses to compute the  $t$ th percentile, as shown in the table that follows.

When you use the WEIGHT statement, the  $t$ th percentile is computed as

$$y = \begin{cases} \frac{1}{2}(x_i + x_{i+1}) & \text{if } \sum_{j=1}^i w_j = pW \\ x_{i+1} & \text{if } \sum_{j=1}^i w_j < pW < \sum_{j=1}^{i+1} w_j \end{cases}$$

where  $w_j$  is the weight associated with  $x_i$  and  $W = \sum_{i=1}^n w_i$  is the sum of the weights.

When the observations have identical weights, the weighted percentiles are the same as the unweighted percentiles with QNTLDEF=5.

**Table A57.2** Methods for Computing Quantile Statistics

QNTLDEF =	Description	Formula	
1	weighted average at $x_{np}$	$y = (1 - g)x_j + gx_{j+1}$ where $x_o$ is taken to be $x_1$	
2	observation numbered closest to $np$	$y = x_i$	if $g \neq \frac{1}{2}$
		$y = x_j$	if $g = \frac{1}{2}$ and $j$ is even
		$y = x_{j+1}$	if $g = \frac{1}{2}$ and $j$ is odd
		where $i$ is the integer part of $np + \frac{1}{2}$	
3	empirical distribution function	$y = x_j$	if $g = 0$
		$y = x_{j+1}$	if $g > 0$
4	weighted average aimed at $x_{(n+1)p}$	$y = (1 - g)x_j + gx_{j+1}$ where $x_{n+1}$ is taken to be $x_n$	
5	empirical distribution function with averaging	$y = \frac{1}{2}(x_j + x_{j+1})$	if $g = 0$
		$y = x_{j+1}$	if $g > 0$

## Hypothesis Testing Statistics

The keywords for hypothesis testing statistics are

T

is the Student's  $t$  statistic to test the null hypothesis that the population mean is equal to  $\mu_0$  and is calculated as

$$\frac{\bar{x} - \mu_0}{s/\sqrt{\sum w_i}}$$

By default,  $\mu_0$  is equal to zero. You can use the MU0= option in the PROC UNIVARIATE statement to specify  $\mu_0$ . You must use VARDEF=DF, which is the default variance divisor, otherwise T is set to missing.

By default, when you use a WEIGHT statement, the procedure counts the  $x_i$  values with nonpositive weights in the degrees of freedom. Use the EXCLNPWGT option in the PROC statement to exclude values with nonpositive weights. Most SAS/STAT procedures, such as PROC TTEST and PROC GLM automatically exclude values with nonpositive weights.

PROBT | PRT

is the two-tailed  $p$ -value for Student's  $t$  statistic, T, with  $n - 1$  degrees of freedom. This value is the probability under the null hypothesis of obtaining a more extreme value of T than is observed in this sample.

## Confidence Limits for the Mean

The keywords for confidence limits are

CLM

is the two-sided confidence limit for the mean. A two-sided  $100(1 - \alpha)$  percent confidence interval for the mean has upper and lower limits

$$\bar{x} \pm t_{(1 - \alpha/2; n - 1)} \frac{s}{\sqrt{\sum w_i}}$$

where  $s$  is  $\sqrt{\frac{1}{n-1} \sum (x_i - \bar{x})^2}$ ,  $t_{(1 - \alpha/2; n - 1)}$  is the  $(1 - \alpha/2)$  critical value of the Student's  $t$  statistics with  $n - 1$  degrees of freedom, and  $\alpha$  is the value of the ALPHA= option which by default is 0.05. Unless you use VARDEF=DF, which is the default variance divisor, CLM is set to missing.

LCLM

is the one-sided confidence limit below the mean. The one-sided  $100(1 - \alpha)$  percent confidence interval for the mean has the lower limit

$$\bar{x} - t_{(1 - \alpha; n - 1)} \frac{s}{\sqrt{\sum w_i}}$$



Unless you use VARDEF=DF, which is the default variance divisor, LCLM is set to missing.

#### UCLM

is the one-sided confidence limit above the mean. The one-sided  $100(1 - \alpha)$  percent confidence interval for the mean has the upper limit

$$\bar{x} + t_{(1 - \alpha; n - 1)} \frac{s}{\sqrt{\sum w_i}}$$

Unless you use VARDEF=DF, which is the default variance divisor, UCLM is set to missing.

---

## Using Weights

As an example of the WEIGHT statement, suppose 20 people are asked to estimate the size of an object 30 cm wide. Each person is placed at a different distance from the object. As the distance from the object increases, the estimates should become less precise. The SAS data set SIZE contains the estimate (ObjectSize) in centimeters at each distance (Distance) in meters and the precision (Precision) for each estimate. Notice that the largest deviation (an overestimate by 20 cm) came at the greatest distance (7.5 meters from the object). As a measure of precision,  $1/\text{Distance}$ , gives more weight to estimates that were made closer to the object and less weight to estimates that were made at greater distances. The PROC MEANS step computes the average estimate of the object size while ignoring the weights. Without a WEIGHT variable, PROC MEANS uses the default weight of 1 for every observation. Thus, the estimates of object size at all distances are given equal weight. The average estimate of the object size exceeds the actual size by 3.55 cm.

```
options nodate pageno=1 linesize=64 pagesize=60;

data size;
 input Distance ObjectSize @@;
 Precision=1/distance;
 datalines;
1.5 30 1.5 20 1.5 30 1.5 25
3 43 3 33 3 25 3 30
4.5 25 4.5 36 4.5 48 4.5 33
6 43 6 36 6 23 6 48
7.5 30 7.5 25 7.5 50 7.5 38
;

proc means data=size maxdec=3 n mean var stddev;
 var objectsize;
 title1 'Unweighted Analysis of the SIZE Data Set';
run;
```

## Unweighted Analysis of the SIZE Data Set

### The MEANS Procedure

Analysis Variable : ObjectSize			
N	Mean	Variance	Std Dev
20	33.550	80.892	8.994

## Data Requirements for Summarization Procedures

Here are the minimal data requirements to compute unweighted statistics and do not describe recommended sample sizes. Statistics are reported as missing if VARDEF=DF (the default) and the following requirements are not met:

- N and NMISS are computed regardless of the number of missing or nonmissing observations.
- SUM, MEAN, MAX, MIN, RANGE, USS, and CSS require at least one nonmissing observation.
- VAR, STD, STDERR, CV, T, PRT, and PROBT require at least two nonmissing observations.
- SKEWNESS requires at least three nonmissing observations.
- KURTOSIS requires at least four nonmissing observations.
- SKEWNESS, KURTOSIS, T, PROBT, and PRT require that STD is greater than zero.
- CV requires that MEAN is not equal to zero.
- CLM, LCLM, UCLM, STDERR, T, PRT, and PROBT require that VARDEF=DF.

## Statistical Background

### Populations and Parameters

Usually, there is a clearly defined set of elements in which you are interested. This set of elements is called the *universe*, and a set of values associated with these elements is called a *population* of values. The statistical term *population* has nothing to do with people. A statistical population is a collection of values, not a collection of people. For example, a universe is all the students at a particular

school, and there could be two populations of interest: one of height values and one of weight values. Or, a universe is the set of all widgets manufactured by a particular company, and the population of values could be the length of time each widget is used before it fails.

A population of values can be described in terms of its *cumulative distribution function*, which gives the proportion of the population less than or equal to each possible value. A discrete population can also be described by a *probability function*, which gives the proportion of the population equal to each possible value. A continuous population can often be described by a *density function*, which is the derivative of the cumulative distribution function. A density function can be approximated by a histogram that gives the proportion of the population lying within each of a series of intervals of values. A probability density function is like a histogram with an infinite number of infinitely small intervals.

In technical literature, when the term *distribution* is used without qualification, it generally refers to the cumulative distribution function. In informal writing, *distribution* sometimes means the density function instead. Often the word *distribution* is used simply to refer to an abstract population of values rather than some concrete population. Thus, the statistical literature refers to many types of abstract distributions, such as normal distributions, exponential distributions, Cauchy distributions, and so on. When a phrase such as *normal distribution* is used, it frequently does not matter whether the cumulative distribution function or the density function is intended.

It might be expedient to describe a population in terms of a few measures that summarize interesting features of the distribution. One such measure, computed from the population values, is called a *parameter*. Many different parameters can be specified to measure different aspects of a distribution.

The most commonly used parameter is the (arithmetic) *mean*. If the population contains a finite number of values, then the population mean is computed as the sum of all the values in the population divided by the number of elements in the population. For an infinite population, the concept of the mean is similar but requires more complicated mathematics.

$E(x)$  denotes the mean of a population of values symbolized by  $x$ , such as height, where  $E$  stands for *expected value*. You can also consider expected values of derived functions of the original values. For example, if  $x$  represents height, then  $E(x^2)$  is the expected value of height squared, that is, the mean value of the population obtained by squaring every value in the population of heights.

---

## Samples and Statistics

It is often impossible to measure all of the values in a population. A collection of measured values is called a *sample*. A mathematical function of a sample of values is called a *statistic*. A statistic is to a sample as a parameter is to a population. It is customary to denote statistics by Roman letters and parameters by Greek letters. For example, the population mean is often written as  $\mu$ , whereas the sample mean is written as  $\bar{x}$ . The field of *statistics* is largely concerned with the study of the behavior of sample statistics.

Samples can be selected in a variety of ways. Most SAS procedures assume that the data constitute a *simple random sample*, which means that the sample was selected in such a way that all possible samples were equally likely to be selected.

Statistics from a sample can be used to make inferences, or reasonable guesses, about the parameters of a population. For example, if you take a random sample of 30 students from the high school, then the mean height for those 30 students is a reasonable guess, or *estimate*, of the mean height of all the students in the high school. Other statistics, such as the standard error, can provide information about how good an estimate is likely to be.

For any population parameter, several statistics can estimate it. Often, however, there is one particular statistic that is customarily used to estimate a given parameter. For example, the sample mean is the usual estimator of the population mean. In the case of the mean, the formulas for the parameter and the statistic are the same. In other cases, the formula for a parameter might be different from that of the most commonly used estimator. The most commonly used estimator is not necessarily the best estimator in all applications.

---

## Measures of Location

---

### Overview of Measures of Location

Measures of location include the mean, the median, and the mode. These measures describe the center of a distribution. In the definitions that follow, notice that if the entire sample changes by adding a fixed amount to each observation, then these measures of location are shifted by the same fixed amount.

---

### The Mean

The population mean  $\mu = E(x)$  is usually estimated by the sample mean  $\bar{x}$ .

---

### The Median

The population median is the central value, lying above and below half of the population values. The sample median is the middle value when the data are arranged in ascending or descending order. For an even number of observations, the midpoint between the two middle values is usually reported as the median.

---

## The Mode

The mode is the value at which the density of the population is at a maximum. Some densities have more than one local maximum (peak) and are said to be *multimodal*. The sample mode is the value that occurs most often in the sample. By default, PROC UNIVARIATE reports the lowest such value if there is a tie for the most-often-occurring sample value. PROC UNIVARIATE lists all possible modes when you specify the MODES option in the PROC statement. If the population is continuous, then all sample values occur once, and the sample mode has little use.

---

## Percentiles

Percentiles, including quantiles, quartiles, and the median, are useful for a detailed study of a distribution. For a set of measurements arranged in order of magnitude, the  $p$ th percentile is the value that has  $p$  percent of the measurements below it and  $(100-p)$  percent above it. The median is the 50th percentile. Because it might not be possible to divide your data so that you get exactly the desired percentile, the UNIVARIATE procedure uses a more precise definition.

The upper quartile of a distribution is the value below which 75% of the measurements fall (the 75th percentile). Twenty-five percent of the measurements fall below the lower quartile value.

---

## Quantiles

In the following example, SAS artificially generates the data with a pseudorandom number function. The UNIVARIATE procedure computes a variety of quantiles and measures of location and writes the values to a SAS data set. A DATA step then uses the SYMPUT routine to assign the values of the statistics to macro variables. The macro %FORMGEN uses these macro variables to produce value labels for the FORMAT procedure. PROC CHART uses the resulting format to display the values of the statistics on a histogram.

```
options nodate pageno=1 linesize=80 pagesize=52;

title 'Example of Quantiles and Measures of Location';

data random;
 drop n;
 do n=1 to 1000;
 X=floor(exp(rannor(314159)*.8+1.8));
 output;
 end;
run;
```

```

proc univariate data=random nextrobs=0;
 var x;
 output out=location
 mean=Mean mode=Mode median=Median
 q1=Q1 q3=Q3 p5=P5 p10=P10 p90=P90 p95=P95
 max=Max;
run;

proc print data=location noobs;
run;

data _null_;
 set location;
 call symput('MEAN',round(mean,1));
 call symput('MODE',mode);
 call symput('MEDIAN',round(median,1));
 call symput('Q1',round(q1,1));
 call symput('Q3',round(q3,1));
 call symput('P5',round(p5,1));
 call symput('P10',round(p10,1));
 call symput('P90',round(p90,1));
 call symput('P95',round(p95,1));
 call symput('MAX',min(50,max));
run;

%macro formgen;
%do i=1 %to &max;
 %let value=&i;
 %if &i=&p5 %then %let value=&value P5;
 %if &i=&p10 %then %let value=&value P10;
 %if &i=&q1 %then %let value=&value Q1;
 %if &i=&mode %then %let value=&value Mode;
 %if &i=&median %then %let value=&value Median;
 %if &i=&mean %then %let value=&value Mean;
 %if &i=&q3 %then %let value=&value Q3;
 %if &i=&p90 %then %let value=&value P90;
 %if &i=&p95 %then %let value=&value P95;
 %if &i=&max %then %let value=>=&value;
 &i="&value"
%end;
%mend;

proc format print;
 value stat %formgen;
run;
options pagesize=42 linesize=80;

proc chart data=random;
 vbar x / midpoints=1 to &max by 1;
 format x stat.;
 footnote 'P5 = 5TH PERCENTILE';
 footnote2 'P10 = 10TH PERCENTILE';
 footnote3 'P90 = 90TH PERCENTILE';
 footnote4 'P95 = 95TH PERCENTILE';
 footnote5 'Q1 = 1ST QUARTILE ';
 footnote6 'Q3 = 3RD QUARTILE ';
run;

```

## Example of Quantiles and Measures of Location

The UNIVARIATE Procedure  
Variable: X

Moments			
N	1000	Sum Weights	1000
Mean	7.605	Sum Observations	7605
Std Deviation	7.38169794	Variance	54.4894645
Skewness	2.73038523	Kurtosis	11.1870588
Uncorrected SS	112271	Corrected SS	54434.975
Coeff Variation	97.0637467	Std Error Mean	0.23342978

Basic Statistical Measures			
Location		Variability	
Mean	7.605000	Std Deviation	7.38170
Median	5.000000	Variance	54.48946
Mode	3.000000	Range	62.00000
		Interquartile Range	6.00000

Tests for Location: Mu0=0				
Test	Statistic		p Value	
Student's t	t	32.57939	Pr >  t	<.0001
Sign	M	494.5	Pr >=  M	<.0001
Signed Rank	S	244777.5	Pr >=  S	<.0001

Tests for Location: Mu0=0				
Test	Statistic		p Value	
Student's t	t	32.57939	Pr >  t	<.0001
Sign	M	494.5	Pr >=  M	<.0001
Signed Rank	S	244777.5	Pr >=  S	<.0001

Quantiles (Definition 5)	
Quantile	Estimate
100% Max	62.0
99%	37.5
95%	21.5
90%	16.0
75% Q3	9.0
50% Median	5.0
25% Q1	3.0
10%	2.0
5%	1.0
1%	0.0
0% Min	0.0

### Example of Quantiles and Measures of Location

Mean	Max	P95	P90	Q3	Median	Q1	P10	P5	Mode
7.605	62	21.5	16	9	5	3	2	1	3





---

# Measures of Variability

---

## Overview of Measures of Variability

Another group of statistics is important in studying the distribution of a population. These statistics measure the *variability*, also called the spread, of values. In the definitions given in the sections that follow, notice that if the entire sample is changed by the addition of a fixed amount to each observation, then the values of these statistics are unchanged. If each observation in the sample is multiplied by a constant, however, then the values of these statistics are appropriately rescaled.

---

## The Range

The sample range is the difference between the largest and smallest values in the sample. For many populations, at least in statistical theory, the range is infinite, so the sample range might not tell you much about the population. The sample range tends to increase as the sample size increases. If all sample values are multiplied by a constant, then the sample range is multiplied by the same constant.

---

## The Interquartile Range

The interquartile range is the difference between the upper and lower quartiles. If all sample values are multiplied by a constant, then the sample interquartile range is multiplied by the same constant.

---

## The Variance

The population variance, usually denoted by  $\sigma^2$ , is the expected value of the squared difference of the values from the population mean:

$$\sigma^2 = E(x - \mu)^2$$

The sample variance is denoted by  $s^2$ . The difference between a value and the mean is called a *deviation from the mean*. Thus, the variance approximates the mean of the squared deviations.

When all the values lie close to the mean, the variance is small but never less than zero. When values are more scattered, the variance is larger. If all sample values are multiplied by a constant, then the sample variance is multiplied by the square of the constant.

Sometimes values other than  $n - 1$  are used in the denominator. The VARDEF= option controls what divisor the procedure uses.

---

## The Standard Deviation

The standard deviation is the square root of the variance, or root-mean-square deviation from the mean, in either a population or a sample. The usual symbols are  $\sigma$  for the population and  $s$  for a sample. The standard deviation is expressed in the same units as the observations, rather than in squared units. If all sample values are multiplied by a constant, then the sample standard deviation is multiplied by the same constant.

---

## Coefficient of Variation

The coefficient of variation is a unitless measure of relative variability. It is defined as the ratio of the standard deviation to the mean expressed as a percentage. The coefficient of variation is meaningful only if the variable is measured on a ratio scale. If all sample values are multiplied by a constant, then the sample coefficient of variation remains unchanged.

---

# Measures of Shape

---

## Skewness

The variance is a measure of the overall size of the deviations from the mean. Since the formula for the variance squares the deviations, both positive and negative deviations contribute to the variance in the same way. In many distributions, positive deviations might tend to be larger in magnitude than negative deviations, or vice versa. *Skewness* is a measure of the tendency of the deviations to be larger in one direction than in the other. For example, the data in the last example are skewed to the right.

Population skewness is defined as

$$E(x - \mu)^3 / \sigma^3$$

Because the deviations are cubed rather than squared, the signs of the deviations are maintained. Cubing the deviations also emphasizes the effects of large deviations. The formula includes a divisor of  $\sigma^3$  to remove the effect of scale, so multiplying all values by a constant does not change the skewness. Skewness can thus be interpreted as a tendency for one tail of the population to be heavier than the other. Skewness can be positive or negative and is unbounded.

---

## Kurtosis

The heaviness of the tails of a distribution affects the behavior of many statistics. Therefore, it is useful to have a measure of tail heaviness. One such measure is *kurtosis*. The population kurtosis is usually defined as

$$\frac{E(x - \mu)^4}{\sigma^4} - 3$$

---

**Note:** Some statisticians omit the subtraction of 3.

---

Because the deviations are raised to the fourth power, positive and negative deviations make the same contribution, and large deviations are strongly emphasized. Because of the divisor  $\sigma^4$ , multiplying each value by a constant has no effect on kurtosis.

Population kurtosis must lie between  $-2$  and  $+\infty$ , inclusive. If  $M_3$  represents population skewness and  $M_4$  represents population kurtosis, then

$$M_4 > (M_3)^2 - 2$$

Statistical literature sometimes reports that kurtosis measures the *peakedness* of a density. However, heavy tails have much more influence on kurtosis than does the shape of the distribution near the mean (Kaplansky 1945; Ali 1974; Johnson, et al. 1980).

Sample skewness and kurtosis are rather unreliable estimators of the corresponding parameters in small samples. They are better estimators when your sample is very large. However, large values of skewness or kurtosis might merit attention even in small samples because such values indicate that statistical methods that are based on normality assumptions might be inappropriate.

---

## The Normal Distribution

One especially important family of theoretical distributions is the *normal* or *Gaussian* distribution. A normal distribution is a smooth symmetric function often referred to as "bell-shaped." Its skewness and kurtosis are both zero. A normal distribution can be completely specified by only two parameters: the mean and the standard deviation. Approximately 68% of the values in a normal population are within one standard deviation of the population mean; approximately 95% of the values are within two standard deviations of the mean; and about 99.7% are within three standard deviations. Use of the term *normal* to describe this particular type of distribution does not imply that other types of distributions are necessarily abnormal or pathological.

Many statistical methods are designed under the assumption that the population being sampled is normally distributed. Nevertheless, most real-life populations do

not have normal distributions. Before using any statistical method based on normality assumptions, you should consult the statistical literature to find out how sensitive the method is to nonnormality and, if necessary, check your sample for evidence of nonnormality.

In the following example, SAS generates a sample from a normal distribution with a mean of 50 and a standard deviation of 10. The UNIVARIATE procedure performs tests for location and normality. Because the data are from a normal distribution, all  $p$ -values from the tests for normality are greater than 0.15. The CHART procedure displays a histogram of the observations. The shape of the histogram is a bell-like, normal density.

```
options nodate pageno=1 linesize=80 pagesize=52;

title '10000 Obs Sample from a Normal Distribution';
title2 'with Mean=50 and Standard Deviation=10';

data normaldat;
 drop n;
 do n=1 to 10000;
 X=10*rannor(53124)+50;
 output;
 end;
run;

proc univariate data=normaldat nextrobs=0 normal
 mu0=50 loccount;
 var x;
run;

proc format;
 picture msd
 20='20 3*Std' (noedit)
 30='30 2*Std' (noedit)
 40='40 1*Std' (noedit)
 50='50 Mean ' (noedit)
 60='60 1*Std' (noedit)
 70='70 2*Std' (noedit)
 80='80 3*Std' (noedit)
 other=' ';
run;
options linesize=80 pagesize=42;

proc chart;
 vbar x / midpoints=20 to 80 by 2;
 format x msd.;
run;
```

### 10000 Obs Sample from a Normal Distribution with Mean=50 and Standard Deviation=10

The UNIVARIATE Procedure  
Variable: X

Moments			
N	10000	Sum Weights	10000
Mean	50.0323744	Sum Observations	500323.744
Std Deviation	9.92013874	Variance	98.4091525
Skewness	-0.019929	Kurtosis	-0.0163755
Uncorrected SS	26016378	Corrected SS	983993.116
Coeff Variation	19.8274395	Std Error Mean	0.09920139

Basic Statistical Measures			
Location		Variability	
Mean	50.03237	Std Deviation	9.92014
Median	50.06492	Variance	98.40915
Mode	.	Range	76.51343
		Interquartile Range	13.28179

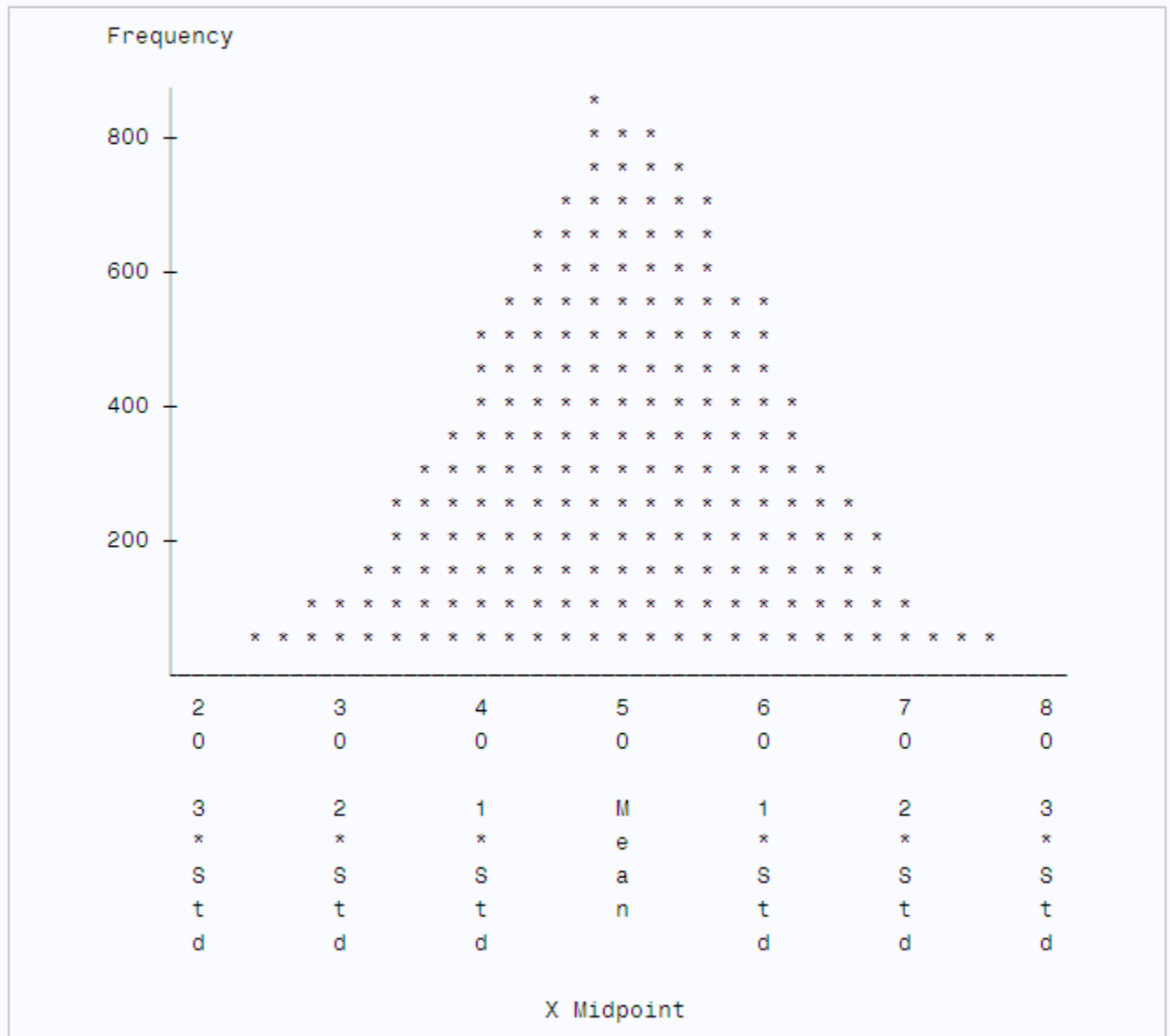
Tests for Location: Mu0=50				
Test	Statistic		p Value	
Student's t	t	0.32635	Pr >  t	0.7442
Sign	M	26	Pr >=  M	0.6101
Signed Rank	S	174063	Pr >=  S	0.5466

Location Counts: Mu0=50.00	
Count	Value
Num Obs > Mu0	5026
Num Obs ^= Mu0	10000
Num Obs < Mu0	4974

Tests for Normality				
Test	Statistic		p Value	
Kolmogorov-Smirnov	D	0.006595	Pr > D	>0.1500
Cramer-von Mises	W-Sq	0.049963	Pr > W-Sq	>0.2500
Anderson-Darling	A-Sq	0.371151	Pr > A-Sq	>0.2500

Quantiles (Definition 5)	
Quantile	Estimate
100% Max	90.2105
99%	72.6780
95%	66.2221
90%	62.6678
75% Q3	56.7280
50% Median	50.0649
25% Q1	43.4462
10%	37.1139
5%	33.5454
1%	26.9189
0% Min	13.6971

### 10000 Obs Sample from a Normal Distribution with Mean=50 and Standard Deviation=10



## Sampling Distribution of the Mean

If you repeatedly draw samples of size  $n$  from a population and compute the mean of each sample, then the sample means themselves have a distribution. Consider a new population consisting of the means of all the samples that could possibly be drawn from the original population. The distribution of this new population is called a *sampling distribution*.

It can be proven mathematically that if the original population has mean  $\mu$  and standard deviation  $\sigma$ , then the sampling distribution of the mean also has mean  $\mu$ ,



but its standard deviation is  $\sigma/\sqrt{n}$ . The standard deviation of the sampling distribution of the mean is called the *standard error of the mean*. The standard error of the mean provides an indication of the accuracy of a sample mean as an estimator of the population mean.

If the original population has a normal distribution, then the sampling distribution of the mean is also normal. If the original distribution is not normal but does not have excessively long tails, then the sampling distribution of the mean can be approximated by a normal distribution for large sample sizes.

The following example consists of three separate programs that show how the sampling distribution of the mean can be approximated by a normal distribution as the sample size increases. The first DATA step uses the RANEXP function to create a sample of 1000 observations from an exponential distribution. The theoretical population mean is 1.00, and the sample mean is 1.01, to two decimal places. The population standard deviation is 1.00; the sample standard deviation is 1.04.

The following example is an example of a nonnormal distribution. The population skewness is 2.00, which is close to the sample skewness of 1.97. The population kurtosis is 6.00, but the sample kurtosis is only 4.80.

```
options nodate pageno=1 linesize=80 pagesize=42;

title '1000 Observation Sample';
title2 'from an Exponential Distribution';

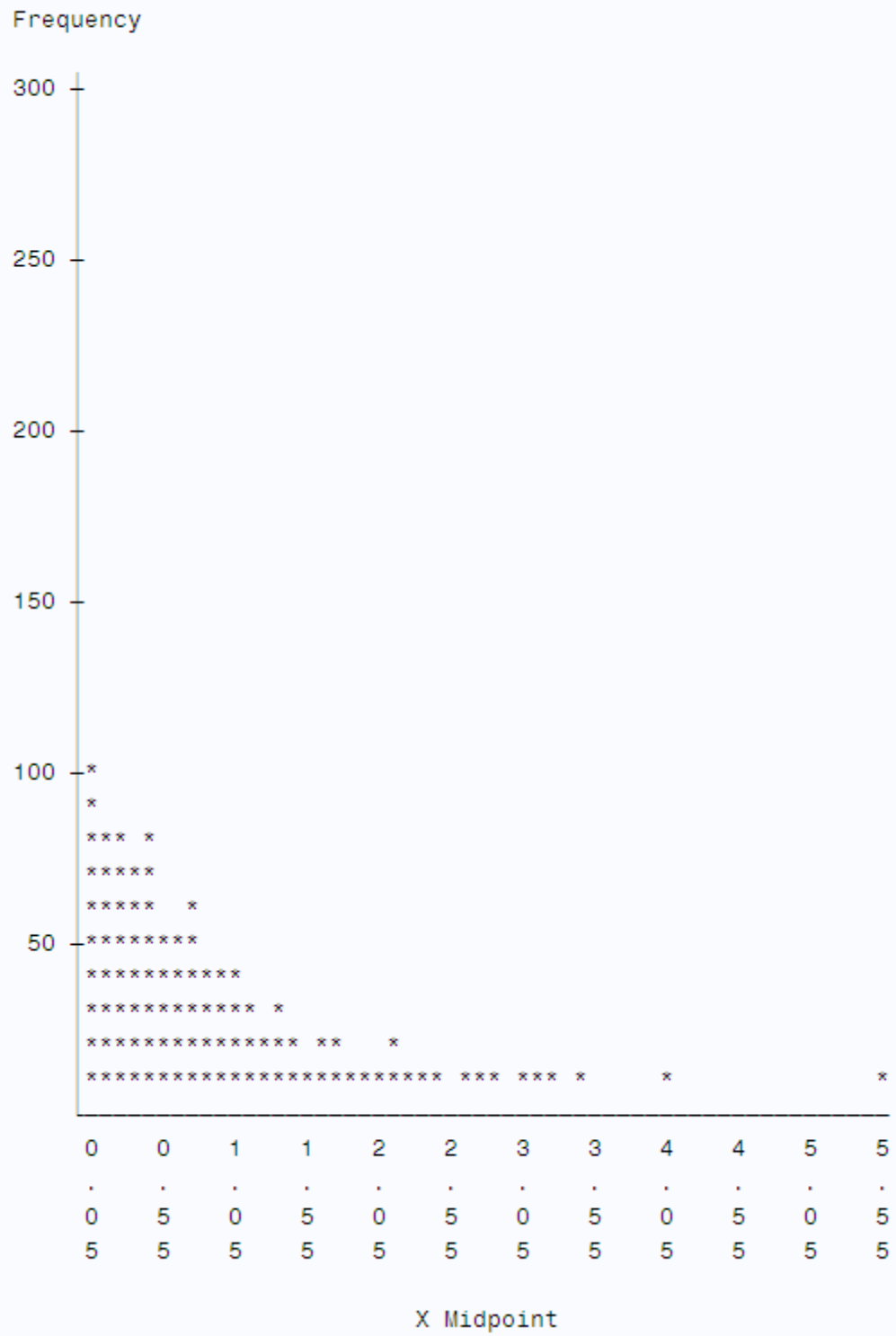
data expodat;
 drop n;
 do n=1 to 1000;
 X=ranexp(18746363);
 output;
 end;
run;
proc format;
 value axisfmt
 .05='0.05'
 .55='0.55'
 1.05='1.05'
 1.55='1.55'
 2.05='2.05'
 2.55='2.55'
 3.05='3.05'
 3.55='3.55'
 4.05='4.05'
 4.55='4.55'
 5.05='5.05'
 5.55='5.55'
 other=' ';
run;

proc chart data=expodat ;
 vbar x / axis=300
 midpoints=0.05 to 5.55 by .1;
 format x axisfmt.;
run;

options pagesize=64;
```

```
proc univariate data=expodat noextrobs=0 normal
 mu0=1;
 var x;
run;
```

### 1000 Observation Sample from an Exponential Distribution



# 1000 Observation Sample from an Exponential Distribution

The UNIVARIATE Procedure  
Variable: X

Moments			
N	1000	Sum Weights	1000
Mean	1.01176214	Sum Observations	1011.76214
Std Deviation	1.04371187	Variance	1.08933447
Skewness	1.96963112	Kurtosis	4.80150594
Uncorrected SS	2111.90777	Corrected SS	1088.24514
Coeff Variation	103.15783	Std Error Mean	0.03300507

Basic Statistical Measures			
Location		Variability	
Mean	1.011762	Std Deviation	1.04371
Median	0.689502	Variance	1.08933
Mode	.	Range	6.63851
		Interquartile Range	1.06252

Tests for Location: Mu0=1				
Test	Statistic		p Value	
Student's t	t	0.356374	Pr >  t	0.7216
Sign	M	-140	Pr >=  M	<.0001
Signed Rank	S	-50781	Pr >=  S	<.0001

Tests for Normality				
Test	Statistic		p Value	
Shapiro-Wilk	W	0.801498	Pr < W	<0.0001
Kolmogorov-Smirnov	D	0.166308	Pr > D	<0.0100
Cramer-von Mises	W-Sq	9.507975	Pr > W-Sq	<0.0050
Anderson-Darling	A-Sq	54.5478	Pr > A-Sq	<0.0050

Quantiles (Definition 5)	
Quantile	Estimate
100% Max	6.63906758
99%	5.04491651
95%	3.13482318
90%	2.37803632
75% Q3	1.35733401
50% Median	0.68950221
25% Q1	0.29481436
10%	0.10219011
5%	0.05192799
1%	0.01195590
0% Min	0.00055441

The next DATA step generates 1000 different samples from the same exponential distribution. Each sample contains ten observations. The MEANS procedure computes the mean of each sample. In the data set that is created by PROC MEANS, each observation represents the mean of a sample of ten observations from an exponential distribution. Thus, the data set is a sample from the sampling distribution of the mean for an exponential population.

PROC UNIVARIATE displays statistics for this sample of means. Notice that the mean of the sample of means is .99, almost the same as the mean of the original population. Theoretically, the standard deviation of the sampling distribution is  $\sigma/\sqrt{n} = 1.00/\sqrt{10} = .32$ , whereas the standard deviation of this sample from the sampling distribution is .30. The skewness (.55) and kurtosis (-.006) are closer to zero in the sample from the sampling distribution than in the original sample from the exponential distribution because the sampling distribution is closer to a normal distribution than is the original exponential distribution. The CHART procedure displays a histogram of the 1000-sample means. The shape of the histogram is much closer to a bell-like, normal density, but it is still distinctly lopsided.

```
options nodate pageno=1 linesize=80 pagesize=48;

title '1000 Sample Means with 10 Obs per Sample';
title2 'Drawn from an Exponential Distribution';

data samp10;
 drop n;
 do Sample=1 to 1000;
 do n=1 to 10;
 X=ranexp(433879);
 output;
 end;
 end;
```

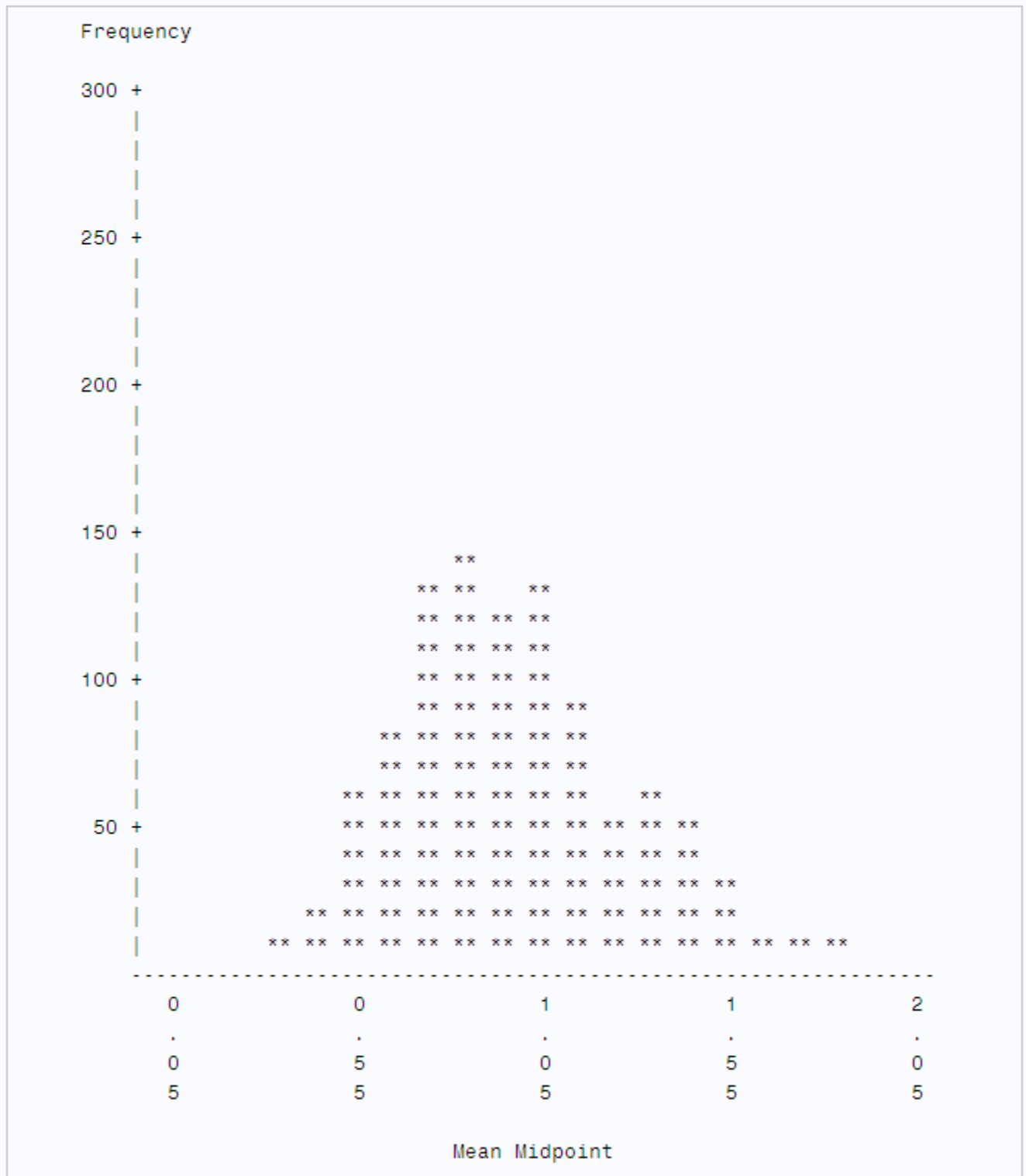
```
proc means data=samp10 noprint;
 output out=mean10 mean=Mean;
 var x;
 by sample;
run;

proc format;
 value axisfmt
 .05='0.05'
 .55='0.55'
 1.05='1.05'
 1.55='1.55'
 2.05='2.05'
 other=' ';
run;

proc chart data=mean10;
 vbar mean/axis=300
 midpoints=0.05 to 2.05 by .1;
 format mean axisfmt.;
run;

options pagesize=64;
proc univariate data=mean10 nextrobs=0 normal
 mu0=1;
 var mean;
run;
```

### 1000 Sample Means with 10 Obs per Sample Drawn from an Exponential Distribution



# 1000 Sample Means with 10 Obs per Sample Drawn from an Exponential Distribution

The UNIVARIATE Procedure  
Variable: Mean

Moments			
N	1000	Sum Weights	1000
Mean	0.9906857	Sum Observations	990.685697
Std Deviation	0.30732649	Variance	0.09444957
Skewness	0.54575615	Kurtosis	-0.0060892
Uncorrected SS	1075.81327	Corrected SS	94.3551193
Coeff Variation	31.0215931	Std Error Mean	0.00971852

Basic Statistical Measures			
Location		Variability	
Mean	0.990686	Std Deviation	0.30733
Median	0.956152	Variance	0.09445
Mode	.	Range	1.79783
		Interquartile Range	0.41703

Tests for Location: Mu0=1				
Test	Statistic		p Value	
Student's t	t	-0.95841	Pr >  t	0.3381
Sign	M	-53	Pr >=  M	0.0009
Signed Rank	S	-22687	Pr >=  S	0.0129

Tests for Normality				
Test	Statistic		p Value	
Shapiro-Wilk	W	0.9779	Pr < W	<0.0001
Kolmogorov-Smirnov	D	0.055498	Pr > D	<0.0100
Cramer-von Mises	W-Sq	0.953926	Pr > W-Sq	<0.0050
Anderson-Darling	A-Sq	5.945023	Pr > A-Sq	<0.0050



Quantiles (Definition 5)	
Quantile	Estimate
100% Max	2.053899
99%	1.827503
95%	1.557175
90%	1.416611
75% Q3	1.181006
50% Median	0.956152
25% Q1	0.763973
10%	0.621787
5%	0.553568
1%	0.433820
0% Min	0.256069

In the following DATA step, the size of each sample from the exponential distribution is increased to 50. The standard deviation of the sampling distribution is smaller than in the previous example because the size of each sample is larger. Also, the sampling distribution is even closer to a normal distribution, as can be seen from the histogram and the skewness.

```
options nodate pageno=1 linesize=80 pagesize=48;

title '1000 Sample Means with 50 Obs per Sample';
title2 'Drawn from an Exponential Distribution';

data samp50;
 drop n;
 do sample=1 to 1000;
 do n=1 to 50;
 X=ranexp(72437213);
 output;
 end;
 end;

proc means data=samp50 noprint;
 output out=mean50 mean=Mean;
 var x;
 by sample;
run;

proc format;
 value axisfmt
 .05='0.05'
 .55='0.55'
 1.05='1.05'
 1.55='1.55'
 2.05='2.05'
```

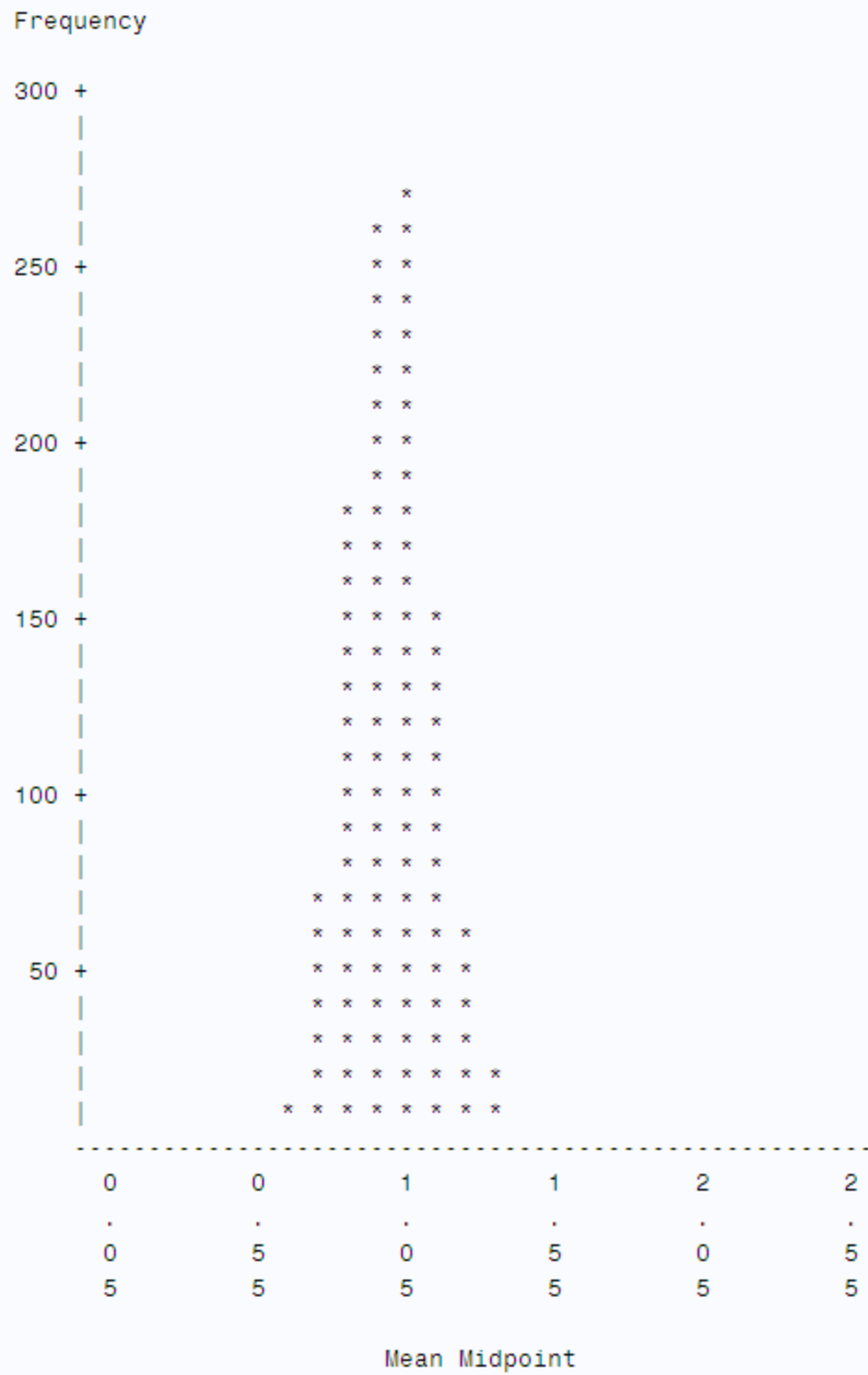
```
 2.55='2.55'
 other=' '
run;

proc chart data=mean50;
 vbar mean / axis=300
 midpoints=0.05 to 2.55 by .1;
 format mean axisfmt.;
run;

options pagesize=64;

proc univariate data=mean50 nextrobs=0 normal
 mu0=1;
 var mean;
run;
```

### 1000 Sample Means with 50 Obs per Sample Drawn from an Exponential Distribution



# 1000 Sample Means with 50 Obs per Sample Drawn from an Exponential Distribution

The UNIVARIATE Procedure  
Variable: Mean

Moments			
N	1000	Sum Weights	1000
Mean	0.99679697	Sum Observations	996.796973
Std Deviation	0.13815404	Variance	0.01908654
Skewness	0.19062633	Kurtosis	-0.1438604
Uncorrected SS	1012.67166	Corrected SS	19.067451
Coeff Variation	13.8597969	Std Error Mean	0.00436881

Basic Statistical Measures			
Location		Variability	
Mean	0.996797	Std Deviation	0.13815
Median	0.996023	Variance	0.01909
Mode	.	Range	0.87040
		Interquartile Range	0.18956

Tests for Location: Mu0=1				
Test	Statistic		p Value	
Student's t	t	-0.73316	Pr >  t	0.4636
Sign	M	-13	Pr >=  M	0.4292
Signed Rank	S	-10767	Pr >=  S	0.2388

Tests for Normality				
Test	Statistic		p Value	
Shapiro-Wilk	W	0.996493	Pr < W	0.0247
Kolmogorov-Smirnov	D	0.023687	Pr > D	>0.1500
Cramer-von Mises	W-Sq	0.084468	Pr > W-Sq	0.1882
Anderson-Darling	A-Sq	0.66039	Pr > A-Sq	0.0877

Quantiles (Definition 5)	
Quantile	Estimate
100% Max	1.454957
99%	1.337016
95%	1.231508
90%	1.179223
75% Q3	1.086515
50% Median	0.996023
25% Q1	0.896953
10%	0.814906
5%	0.780783
1%	0.706588
0% Min	0.584558

## Testing Hypotheses

### Defining a Hypothesis

The purpose of the statistical methods that have been discussed so far is to estimate a population parameter by means of a sample statistic. Another class of statistical methods is used for testing hypotheses about population parameters or for measuring the amount of evidence against a hypothesis.

Consider the universe of students in a college. Let the variable  $X$  be the number of pounds by which a student's weight deviates from the ideal weight for a person of the same sex, height, and build. You want to find out whether the population of students is, on the average, underweight or overweight. To this end, you have taken a random sample of  $X$  values from nine students, with results as given in the following DATA step:

```

title 'Deviations from Normal Weight';

data x;
 input X @@;
 datalines;
-7 -2 1 3 6 10 15 21 30
;

```

You can define several hypotheses of interest. One hypothesis is that, on the average, the students are of exactly ideal weight. If  $\mu$  represents the population

mean of the  $X$  values, then you can write this hypothesis, called the *null* hypothesis, as  $H_0: \mu = 0$ . The other two hypotheses, called *alternative* hypotheses, are that the students are underweight on the average,  $H_1: \mu < 0$ , and that the students are overweight on the average,  $H_2: \mu > 0$ .

The null hypothesis is so called because it corresponds in many situations to the assumption of “no effect” or “no difference.” However, this interpretation is not appropriate for all testing problems. The null hypothesis is like a straw man that can be toppled by statistical evidence. You decide between the alternative hypotheses according to which way the straw man falls.

A naive way to approach this problem would be to look at the sample mean  $\bar{x}$  and decide among the three hypotheses according to the following rule:

- If  $\bar{x} < 0$ , then decide on  $H_1: \mu < 0$ .
- If  $\bar{x} = 0$ , then decide on  $H_0: \mu = 0$ .
- If  $\bar{x} > 0$ , then decide on  $H_2: \mu > 0$ .

The trouble with this approach is that there might be a high probability of making an incorrect decision. If  $H_0$  is true, then you are nearly certain to make a wrong decision because the chances of  $\bar{x}$  being exactly zero are almost nil. If  $\mu$  is slightly less than zero, so that  $H_1$  is true, then there might be nearly a 50% chance that  $\bar{x}$  will be greater than zero in repeated sampling, so the chances of incorrectly choosing  $H_2$  would also be nearly 50%. Thus, you have a high probability of making an error if  $\bar{x}$  is near zero. In such cases, there is not enough evidence to make a confident decision, so the best response might be to reserve judgment until you can obtain more evidence.

The question is, how far from zero must  $\bar{x}$  be for you to be able to make a confident decision? The answer can be obtained by considering the sampling distribution of  $\bar{x}$ . If  $X$  has an approximately normal distribution, then  $\bar{x}$  has an approximately normal sampling distribution. The mean of the sampling distribution of  $\bar{x}$  is  $\mu$ . Assume temporarily that  $\sigma$ , the standard deviation of  $X$ , is known to be 12. Then the standard error of  $\bar{x}$  for samples of nine observations is  $\sigma/\sqrt{n} = 12/\sqrt{9} = 4$ .

You know that about 95% of the values from a normal distribution are within two standard deviations of the mean, so about 95% of the possible samples of nine  $X$  values have a sample mean  $\bar{x}$  between  $0 - 2(4)$  and  $0 + 2(4)$ , or between  $-8$  and  $8$ . Consider the chances of making an error with the following decision rule:

- If  $\bar{x} < -8$ , then decide on  $H_1: \mu < 0$ .
- If  $-8 \leq \bar{x} \leq 8$ , then reserve judgment.
- If  $\bar{x} > 8$ , then decide on  $H_2: \mu > 0$ .

If  $H_0$  is true, then in about 95% of the possible samples  $\bar{x}$  will be between the *critical values*  $-8$  and  $8$ , so you will reserve judgment. In these cases the statistical evidence is not strong enough to fell the straw man. In the other 5% of the samples, you will make an error. In 2.5% of the samples, you will incorrectly choose  $H_1$ ; in 2.5%, you will incorrectly choose  $H_2$ .

The price that you pay for controlling the chances of making an error is the necessity of reserving judgment when there is not sufficient statistical evidence to reject the null hypothesis.

## Significance and Power

The probability of rejecting the null hypothesis, if it is true, is called the *Type I error rate* of the statistical test and is typically denoted as  $\alpha$ . In this example, an  $\bar{x}$  value less than  $-8$  or greater than  $8$  is said to be *statistically significant* at the 5% level. You can adjust the type I error rate according to your needs by choosing different critical values. For example, critical values of  $-4$  and  $4$  would produce a significance level of about 32%, and  $-12$  and  $12$  would give a type I error rate of about 0.3%.

The decision rule is a *two-tailed test* because the alternative hypotheses allow for population means either smaller or larger than the value specified in the null hypothesis. If you were interested only in the possibility of the students being overweight on the average, then you could use a one-tailed test:

- If  $\bar{x} \leq 8$ , then reserve judgment.
- If  $\bar{x} > 8$ , then decide on  $H_2: \mu > 0$ .

For this one-tailed test, the type I error rate is 2.5%, half that of the two-tailed test.

The probability of rejecting the null hypothesis, if it is false, is called the *power* of the statistical test and is typically denoted as  $1 - \beta$ .  $\beta$  is called the *Type II error rate*, which is the probability of not rejecting a false null hypothesis. The power depends on the true value of the parameter. In the example, assume that the population mean is 4. The power for detecting  $H_2$  is the probability of getting a sample mean greater than 8. The critical value 8 is one standard error higher than the population mean 4. The chance of getting a value at least one standard deviation greater than the mean from a normal distribution is about 16%, so the power for detecting the alternative hypothesis  $H_2$  is about 16%. If the population mean were 8, then the power for  $H_2$  would be 50%, whereas a population mean of 12 would yield a power of about 84%.

The smaller the type I error rate is, the less the chance of making an incorrect decision, but the higher the chance of having to reserve judgment. In choosing a type I error rate, you should consider the resulting power for various alternatives of interest.

## Student's t Distribution

In practice, you usually cannot use any decision rule that uses a critical value based on  $\sigma$  because you do not usually know the value of  $\sigma$ . However, you can use  $s$  as an estimate of  $\sigma$ . Consider the following statistic:

$$t = \frac{\bar{x} - \mu_0}{s/\sqrt{n}}$$

This  $t$  statistic is the difference between the sample mean and the hypothesized mean  $\mu_0$  divided by the estimated standard error of the mean.

If the null hypothesis is true and the population is normally distributed, then the  $t$  statistic has what is called a *Student's  $t$  distribution* with  $n - 1$  degrees of freedom. This distribution looks very similar to a normal distribution, but the tails of the Student's  $t$  distribution are heavier. As the sample size gets larger, the sample standard deviation becomes a better estimator of the population standard deviation, and the  $t$  distribution gets closer to a normal distribution.

You can base a decision rule on the  $t$  statistic:

- If  $t < -2.3$ , then decide on  $H_1: \mu < 0$ .
- If  $-2.3 \leq t \leq 2.3$ , then reserve judgment.
- If  $t > 2.3$ , then decide on  $H_0: \mu > 0$ .

The value 2.3 was obtained from a table of Student's  $t$  distribution to give a type I error rate of 5% for 8 (that is,  $9 - 1 = 8$ ) degrees of freedom. Most common statistics texts contain a table of Student's  $t$  distribution. If you do not have a statistics text handy, then you can use the DATA step and the TINV function to print any values from the  $t$  distribution.

By default, PROC UNIVARIATE computes a  $t$  statistic for the null hypothesis that  $\mu_0 = 0$ , along with related statistics. Use the MU0= option in the PROC statement to specify another value for the null hypothesis.

This example uses the data on deviations from normal weight, which consist of nine observations. First, PROC MEANS computes the  $t$  statistic for the null hypothesis that  $\mu = 0$ . Then, the TINV function in a DATA step computes the value of Student's  $t$  distribution for a two-tailed test at the 5% level of significance and eight degrees of freedom.

```
data devnorm;
 title 'Deviations from Normal Weight';
 input X @@;
 datalines;
-7 -2 1 3 6 10 15 21 30
;

proc means data=devnorm maxdec=3 n mean
 std stderr t probt;
run;

title 'Student's t Critical Value';

data _null_;
 file print;
 t=tinv(.975,8);
 put t 5.3;
run;
```



## Deviations from Normal Weight

## The MEANS Procedure

Analysis Variable : X					
N	Mean	Std Dev	Std Error	t Value	Pr >  t
9	8.556	11.759	3.920	2.18	0.0606

## Student's t Critical Value

2.306

Deviations from Normal Weight 1  
The MEANS Procedure

Analysis Variable : X

N	Mean	Std Dev	Std Error	t Value	Pr >  t
9	8.556	11.759	3.920	2.18	0.0606

Student's t Critical Value

2

2.306

In the current example, the value of the  $t$  statistic is 2.18, which is less than the critical  $t$  value of 2.3 (for a 5% significance level and eight degrees of freedom). Thus, at a 5% significance level, you must reserve judgment. If you had chosen to use a 10% significance level, then the critical value of the  $t$  distribution would have been 1.86 and you could have rejected the null hypothesis. The sample size is so small, however, that the validity of your conclusion depends strongly on how close the distribution of the population is to a normal distribution.

## Probability Values

Another way to report the results of a statistical test is to compute a *probability value* or *p-value*. A *p-value* gives the probability in repeated sampling of obtaining a statistic as far in the directions specified by the alternative hypothesis as is the value actually observed. A two-tailed *p-value* for a  $t$  statistic is the probability of obtaining an absolute  $t$  value that is greater than the observed absolute  $t$  value. A one-tailed *p-value* for a  $t$  statistic for the alternative hypothesis  $\mu > \mu_0$  is the probability of obtaining a  $t$  value greater than the observed  $t$  value. Once the *p-value* is computed, you can perform a hypothesis test by comparing the *p-value*

with the desired significance level. If the  $p$ -value is less than or equal to the type I error rate of the test, then the null hypothesis can be rejected. The two-tailed  $p$ -value, labeled  $\Pr > |t|$  in the PROC MEANS output, is .0606, so the null hypothesis could be rejected at the 10% significance level but not at the 5% level.

A  $p$ -value is a measure of the strength of the evidence against the null hypothesis. The smaller the  $p$ -value, the stronger the evidence for rejecting the null hypothesis.

---

**Note:** For a more thorough discussion, consult an introductory statistics textbook such as Mendenhall and Beaver (1998); Ott and Mendenhall (1994); or Snedecor and Cochran (1989).

---



---

## References

- Ali, M. M. 1974. "Stochastic Ordering and Kurtosis Measure." *Journal of the American Statistical Association* 69: 543-545.
- Johnson, M. E., G. L. Tietjen, and R. J. Beckman. 1980. "A New Family of Probability Distributions with Applications to Monte Carlo Studies." *Journal of the American Statistical Association* 75: 276-279.
- Kaplansky, I. 1945. "A Common Error Concerning Kurtosis." *Journal of the American Statistical Association*, 40: 259-263.
- Mendenhall, W. and R. Beaver. 1998. *Introduction to Probability and Statistics*. 10th ed. Belmont, CA: Wadsworth Publishing Company.
- Ott, R. and W. Mendenhall. 1994. *Understanding Statistics*. 6th ed. North Scituate, MA: Duxbury Press.
- Schlotzhauer, S. D. and R. C. Littell. 1997. *SAS System for Elementary Statistical Analysis*. Second ed. Cary, NC: SAS Institute Inc.
- Snedecor, G. W. and W. C. Cochran. 1989. *Statistical Methods*. 8th ed. Ames, IA: Iowa State University Press.

## Appendix 2

# Raw Data and DATA Steps for Base SAS Procedures

---

<i>Overview of Raw Data and DATA Steps for Base SAS Procedures</i> .....	<b>2470</b>
<i>CARSURVEY</i> .....	<b>2471</b>
<i>CENSUS</i> .....	<b>2472</b>
<i>CHARITY</i> .....	<b>2473</b>
<i>CONTROL Library</i> .....	<b>2475</b>
Contents of the CONTROL Library .....	2475
CONTROL.ALL .....	2476
CONTROL.BODYFAT .....	2477
CONTROL.CONFOUND .....	2478
CONTROL.CORONARY .....	2478
CONTROL.DRUG1 .....	2479
CONTROL.DRUG2 .....	2479
CONTROL.DRUG3 .....	2480
CONTROL.DRUG4 .....	2480
CONTROL.DRUG5 .....	2480
CONTROL.GROUP .....	2481
CONTROL.MLSCL .....	2486
CONTROL.NAMES .....	2487
CONTROL.OXYGEN .....	2487
CONTROL.PERSONL .....	2488
CONTROL.PHARM .....	2494
CONTROL.POINTS .....	2494
CONTROL.PRENAT .....	2494
CONTROL.RESULTS .....	2497
CONTROL.SLEEP .....	2498
CONTROL.SYNDROME .....	2500
CONTROL.TENSION .....	2501
CONTROL.TEST2 .....	2501
CONTROL.TRAIN .....	2502
CONTROL.VISION .....	2502
CONTROL.WEIGHT .....	2502
CONTROL.WGHT .....	2504

<i>CUSTOMER_RESPONSE</i> .....	2506
<i>DJIA</i> .....	2508
<i>EDUCATION</i> .....	2509
<i>EMPDATA</i> .....	2510
<i>ENERGY</i> .....	2512
<i>EXP Library</i> .....	2513
EXP.RESULTS .....	2513
EXP.SUR .....	2514
<i>EXPREV</i> .....	2514
<i>GROC</i> .....	2516
<i>MATCH_11</i> .....	2517
<i>PROCLIB.DELAY</i> .....	2519
<i>PROCLIB.EMP95</i> .....	2520
<i>PROCLIB.EMP96</i> .....	2521
<i>PROCLIB.INTERNAT</i> .....	2522
<i>PROCLIB.LAKES</i> .....	2522
<i>PROCLIB.MARCH</i> .....	2523
<i>PROCLIB.PAYLIST2</i> .....	2524
<i>PROCLIB.PAYROLL</i> .....	2525
<i>PROCLIB.PAYROLL2</i> .....	2528
<i>PROCLIB.SCHEDULE</i> .....	2528
<i>PROCLIB.STAFF</i> .....	2531
<i>PROCLIB.STAFF2</i> .....	2534
<i>PROCLIB.SUPERV</i> .....	2535
<i>RADIO</i> .....	2535
<i>SALES</i> .....	2548

---

## Overview of Raw Data and DATA Steps for Base SAS Procedures

The following raw data examples and DATA step examples are for use with the Base SAS Procedures.

The programs for examples in this document generally show you how to create the data sets that are used. Some examples show only partial data. For these examples, the complete data is shown in this appendix.

---

# CARSURVEY

```
data carsurvey;
 input Rater Age Progressa Remark Jupiter Dynamo;
 datalines;
1 38 94 98 84 80
2 49 96 84 80 77
3 16 64 78 76 73
4 27 89 73 90 92
5 50 93 79 84 34
6 57 92 89 75 89
7 21 88 90 89 91
8 39 88 87 76 64
9 26 77 94 93 47
10 17 68 72 85 79
11 38 94 93 84 70
12 29 78 97 74 33
13 41 89 83 75 82
14 37 54 98 70 83
15 52 92 85 88 78
16 23 85 89 89 95
17 61 92 88 77 85
18 24 87 88 88 87
19 18 54 50 62 74
20 62 90 91 90 86
21 49 94 98 84 80
22 16 96 84 80 77
23 27 64 78 76 73
24 50 89 73 90 92
25 57 93 79 84 34
26 21 92 86 75 93
27 39 88 97 89 91
28 26 88 87 76 64
29 17 77 94 93 47
30 38 68 72 85 79
31 29 94 93 84 70
32 41 78 97 74 33
33 37 89 83 75 82
34 52 54 98 70 83
35 23 92 85 88 78
36 61 85 93 89 66
37 24 92 88 77 85
38 18 87 88 88 87
39 62 54 50 62 74
40 38 90 91 90 86
41 57 94 98 84 80
42 16 96 84 80 77
43 19 64 78 76 73
```

```

44 59 89 73 90 92
45 57 93 79 84 34
46 21 92 86 75 90
47 39 88 97 89 91
48 26 88 87 76 64
49 17 77 94 93 47
50 56 68 72 85 79
51 29 94 93 84 70
52 41 78 97 74 33
53 37 89 83 75 82
54 52 54 98 70 83
55 17 92 85 88 78
56 61 85 93 89 66
57 24 92 85 77 85
58 18 87 88 88 87
59 62 54 50 62 74
60 38 90 91 90 86
61 38 94 98 84 80
62 49 96 84 80 77
63 16 64 78 76 73
64 27 89 73 90 92
65 50 93 79 84 34
66 57 92 89 75 89
67 21 88 93 89 91
68 39 88 87 76 64
69 26 77 94 93 47
70 17 68 72 85 79
71 38 94 93 84 70
72 29 78 97 74 33
73 41 89 83 75 82
74 37 54 98 70 83
75 52 92 85 88 78
76 23 85 93 89 88
77 61 92 88 77 85
78 24 87 88 88 91
79 18 54 50 62 74
80 62 90 91 90 86
;

```

---

## CENSUS

```

data census;
 input Density CrimeRate State $ 14-27 PostalCode $ 29-30;
 datalines;
263.3 4575.3 Ohio OH
62.1 7017.1 Washington WA
103.4 5161.9 South Carolina SC
53.4 3438.6 Mississippi MS
180.0 8503.2 Florida FL
80.8 2190.7 West Virginia WV
428.7 5477.6 Maryland MD

```

```

71.2 4707.5 Missouri MO
43.9 4245.2 Arkansas AR
7.3 6371.4 Nevada NV
264.3 3163.2 Pennsylvania PA
11.5 4156.3 Idaho ID
44.1 6025.6 Oklahoma OK
51.2 4615.8 Minnesota MN
55.2 4271.2 Vermont VT
27.4 6969.9 Oregon OR
205.3 5416.5 Illinois IL
94.1 5792.0 Georgia GA
9.1 2678.0 South Dakota SD
9.4 2833.0 North Dakota ND
102.4 3371.7 New Hampshire NH
54.3 7722.4 Texas TX
76.6 4451.4 Alabama AL
307.6 4938.8 Delaware DE
151.4 6506.4 California CA
111.6 4665.6 Tennessee TN
120.4 4649.9 North Carolina NC
;

```

---

# CHARITY

```

data Charity;
 input School $ 1-7 Year 9-12 Name $ 14-20 MoneyRaised 22-26
 HoursVolunteered 28-29;
 datalines;
Monroe 2016 Allison 31.65 19
Monroe 2016 Barry 23.76 16
Monroe 2016 Candace 21.11 5
Monroe 2016 Danny 6.89 23
Monroe 2016 Edward 53.76 31
Monroe 2016 Fiona 48.55 13
Monroe 2016 Gert 24.00 16
Monroe 2016 Harold 27.55 17
Monroe 2016 Ima 15.98 9
Monroe 2016 Jack 20.00 23
Monroe 2016 Katie 22.11 2
Monroe 2016 Lisa 18.34 17
Monroe 2016 Tonya 55.16 40
Monroe 2016 Max 26.77 34
Monroe 2016 Ned 28.43 22
Monroe 2016 Opal 32.66 14
Monroe 2017 Patsy 18.33 18
Monroe 2017 Quentin 16.89 15
Monroe 2017 Randall 12.98 17
Monroe 2017 Sam 15.88 5
Monroe 2017 Tyra 21.88 23
Monroe 2017 Myrtle 47.33 26
Monroe 2017 Frank 41.11 22

```

Monroe	2017	Cameron	65.44	14
Monroe	2017	Vern	17.89	11
Monroe	2017	Wendell	23.00	10
Monroe	2017	Bob	26.88	6
Monroe	2017	Leah	28.99	23
Monroe	2018	Becky	30.33	26
Monroe	2018	Sally	35.75	27
Monroe	2018	Edgar	27.11	12
Monroe	2018	Dawson	17.24	16
Monroe	2018	Lou	5.12	16
Monroe	2018	Damien	18.74	17
Monroe	2018	Mona	27.43	7
Monroe	2018	Della	56.78	15
Monroe	2018	Monique	29.88	19
Monroe	2018	Carl	31.12	25
Monroe	2018	Reba	35.16	22
Monroe	2018	Dax	27.65	23
Monroe	2018	Gary	23.11	15
Monroe	2018	Suzie	26.65	11
Monroe	2018	Benito	47.44	18
Monroe	2018	Thomas	21.99	23
Monroe	2018	Annie	24.99	27
Monroe	2018	Paul	27.98	22
Monroe	2018	Alex	24.00	16
Monroe	2018	Lauren	15.00	17
Monroe	2018	Julia	12.98	15
Monroe	2018	Keith	11.89	19
Monroe	2018	Jackie	26.88	22
Monroe	2018	Pablo	13.98	28
Monroe	2018	L.T.	56.87	33
Monroe	2018	Willard	78.65	24
Monroe	2018	Kathy	32.88	11
Monroe	2018	Abby	35.88	10
Kennedy	2016	Arturo	34.98	14
Kennedy	2016	Grace	27.55	25
Kennedy	2016	Winston	23.88	22
Kennedy	2016	Vince	12.88	21
Kennedy	2016	Claude	15.62	5
Kennedy	2016	Mary	28.99	34
Kennedy	2016	Abner	25.89	22
Kennedy	2016	Jay	35.89	35
Kennedy	2016	Alicia	28.77	26
Kennedy	2016	Freddy	29.00	27
Kennedy	2016	Eloise	31.67	25
Kennedy	2016	Jenny	43.89	22
Kennedy	2016	Thelma	52.63	21
Kennedy	2016	Tina	19.67	21
Kennedy	2016	Eric	24.89	12
Kennedy	2016	Bubba	37.88	12
Kennedy	2016	G.L.	25.89	21
Kennedy	2016	Bert	28.89	21
Kennedy	2017	Clay	26.44	21
Kennedy	2017	Leeann	27.17	17
Kennedy	2017	Georgia	38.90	11
Kennedy	2017	Bill	42.23	25
Kennedy	2017	Holly	18.67	27



```

Kennedy 2017 Benny 19.09 25
Kennedy 2017 Cammie 28.77 28
Kennedy 2017 Amy 27.08 31
Kennedy 2017 Doris 22.22 24
Kennedy 2017 Robbie 19.80 24
Kennedy 2017 Ted 27.07 25
Kennedy 2017 Sarah 24.44 12
Kennedy 2017 Megan 28.89 11
Kennedy 2017 Jeff 31.11 12
Kennedy 2017 Taz 30.55 11
Kennedy 2017 George 27.56 11
Kennedy 2017 Heather 38.67 15
Kennedy 2018 Nancy 29.90 26
Kennedy 2018 Rusty 30.55 28
Kennedy 2018 Mimi 37.67 22
Kennedy 2018 J.C. 23.33 27
Kennedy 2018 Clark 27.90 25
Kennedy 2018 Rudy 27.78 23
Kennedy 2018 Samuel 34.44 18
Kennedy 2018 Forrest 28.89 26
Kennedy 2018 Luther 72.22 24
Kennedy 2018 Trey 6.78 18
Kennedy 2018 Albert 23.33 19
Kennedy 2018 Che-Min 26.66 33
Kennedy 2018 Preston 32.22 23
Kennedy 2018 Larry 40.00 26
Kennedy 2018 Anton 35.99 28
Kennedy 2018 Sid 27.45 25
Kennedy 2018 Will 28.88 21
Kennedy 2018 Morty 34.44 25
;

```

---

# CONTROL Library

---

## Contents of the CONTROL Library

---

Here are the contents of the CONTROL library that is used in the DATASETS procedure section.

### Directory

Libref	CONTROL
Engine	V9
Physical Name	\myfiles\control
File Name	\myfiles\control

Member	Obs, Entries
--------	--------------

File
------

#	Name	Type	or Indexes	Vars	Label	Size	Last Modified
1	A1	CATALOG	23			62464	04Jan18:14:20:12
2	A2	CATALOG	1			17408	04Jan18:14:20:12
3	ALL	DATA	23	17		13312	04Jan18:14:20:12
4	BODYFAT	DATA	1	2		5120	04Jan18:14:20:12
5	CONFOUND	DATA	8	4		5120	04Jan18:14:20:12
6	CORONARY	DATA	39	4		5120	04Jan18:14:20:12
7	DRUG1	DATA	6	2	JAN18 Data	5120	04Jan18:14:20:12
8	DRUG2	DATA	13	2	MAY18 Data	5120	04Jan18:14:20:12
9	DRUG3	DATA	11	2	JUL17 Data	5120	04Jan18:14:20:12
10	DRUG4	DATA	7	2	JAN17 Data	5120	04Jan18:14:20:12
11	DRUG5	DATA	1	2	JUL17 Data	5120	04Jan18:14:20:12
12	ETEST1	CATALOG	1			17408	04Jan18:14:20:16
13	ETEST2	CATALOG	1			17408	04Jan18:14:20:16
14	ETEST3	CATALOG	1			17408	04Jan18:14:20:16
15	ETEST4	CATALOG	1			17408	04Jan18:14:20:16
16	ETEST5	CATALOG	1			17408	04Jan18:14:20:16
17	ETESTS	CATALOG	1			17408	04Jan18:14:20:16
18	FORMATS	CATALOG	6			17408	04Jan18:14:20:16
19	GROUP	DATA	148	11		25600	04Jan18:14:20:16
20	MLSCL	DATA	32	4	Multiple Sclerosis Data	5120	04Jan18:14:20:16
21	NAMES	DATA	7	4		5120	04Jan11:14:20:16
22	OXYGEN	DATA	31	7		9216	04Jan18:14:20:16
23	PERSONL	DATA	148	11		25600	04Jan18:14:20:16
24	PHARM	DATA	6	3	Sugar Study	5120	04Jan18:14:20:16
25	POINTS	DATA	6	6		5120	04Jan18:14:20:16
26	PRENAT	DATA	149	6		17408	04Jan18:14:20:16
27	RESULTS	DATA	10	5		5120	04Jan18:14:20:16
28	SLEEP	DATA	108	6		9216	04Jan18:14:20:16
29	SYNDROME	DATA	46	8		9216	04Jan18:14:20:16
30	TENSION	DATA	4	3		5120	04Jan18:14:20:16
31	TEST2	DATA	15	5		5120	04Jan18:14:20:16
32	TRAIN	DATA	7	2		5120	04Jan18:14:20:16
33	VISION	DATA	16	3		5120	04Jan18:14:20:16
34	WEIGHT	DATA	83	13	California Results	13312	04Jan18:14:20:16
35	WGHT	DATA	83	13	California Results	13312	04Jan18:14:20:16

16

---

## CONTROL.ALL

Here are the raw data and DATA steps for all the data files in the CONTROL library.

```
data control.all;
 input FMTNAME $8. START $9. END $8. LABEL $20. MIN best4. MAX best4.
 DEFAULT best4. LENGTH best4. FUZZ best8. PREFIX $2. MULT best8.
 FILL $1. NOEDIT best4. TYPE $2. SEXCL $2. EEXCL $2. HLO $7. ;
 label FMTNAME='Format name'
 START='Starting value for format'
 END='Ending value for format'
```

```

 LABEL='Format value label'
 MIN='Minimum length'
 MAX='Maximum length'
 DEFAULT='Default length'
 LENGTH='Format length'
 FUZZ='Fuzz value'
 PREFIX='Prefix characters'
 MULT='Multiplier'
 FILL='Fill character'
 NOEDIT='Is picture string noedit?'
 TYPE='Type of format'
 SEXCL='Start exclusion'
 EEXCL='End exclusion'
 HLO='Additional information';
 datalines;
BENEFIT LOW 7304 WORDDATE20. 1 40 20 20 1E-12 0.00 0 N N
N LF
BENEFIT 7305 HIGH ** Not Eligible ** 1 40 20 20 1E-12 0.00 0 N N N
DOLLARS LOW HIGH 000,000 1 40 7 7 1E-12 $ 1.96 0 P N
N LH
NOZEROS LOW 0.01 999 1 40 5 5 1E-12 . 1000.00 0 P N
Y L
NOZEROS 0.01 0.1 99 1 40 5 5 1E-12 . 100.00 0 P N Y
NOZEROS 0.1 1 0.000 1 40 5 5 1E-12 . 1000.00 0 P N Y
NOZEROS 1 HIGH 0.000 1 40 5 5 1E-12 1000.00 0 P N
N H
BRIT BR1 BR1 Birmingham 1 40 14 14 0 0.00 0 C N N
BRIT BR2 BR2 Plymouth 1 40 14 14 0 0.00 0 C N N
BRIT BR3 BR3 York 1 40 14 14 0 0.00 0 C N N
BRIT *OTHER****OTHER* INCORRECT CODE 1 40 14 14 0 0.00 0 C N
N 0
SKILL A D~ Test A 1 40 6 6 0 0.00 0 C N N
SKILL E M~ Test B 1 40 6 6 0 0.00 0 C N N
SKILL N Z~ Test C 1 40 6 6 0 0.00 0 C N N
SKILL a d~ Test A 1 40 6 6 0 0.00 0 C N N
SKILL e m~ Test B 1 40 6 6 0 0.00 0 C N N
SKILL n z~ Test C 1 40 6 6 0 0.00 0 C N N
EVAL 0 4 _SAME_ 1 40 1 1 0 0.00 0 I N
N I
EVAL C C 1 1 40 1 1 0 0.00 0 I N N
EVAL E E 2 1 40 1 1 0 0.00 0 I N N
EVAL N N 0 1 40 1 1 0 0.00 0 I N N
EVAL O O 4 1 40 1 1 0 0.00 0 I N N
EVAL S S 3 1 40 1 1 0 0.00 0 I N N
;
run;

```

---

## CONTROL.BODYFAT

```

data control.bodyfat;
 input NAME $ AGE $;
 datalines;
jeff 44

```

```
;
run;
```

---

## CONTROL.CONFOUND

```
data control.confound;
 input SMOKING $8. STATUS $8. CANCER $8. WT;
 datalines;
Yes Single Yes 34
Yes Single No 120
Yes Married Yes 7
Yes Married No 30
Yes Single Yes 2
Yes Single No 30
Yes Married Yes 6
Yes Married No 145
;
run;
```

---

## CONTROL.CORONARY

```
data control.coronary;
 input SEX ECG AGE CA;
 datalines;
0 0 28 0
0 0 34 0
0 0 38 0
0 0 41 0
0 0 44 0
0 0 45 1
0 0 46 0
0 0 47 0
0 0 50 0
0 0 51 0
0 0 51 0
0 0 53 0
0 0 55 1
0 0 59 0
0 0 60 1
0 0 32 1
0 0 33 0
0 0 35 0
0 0 39 0
0 0 40 0
0 0 46 0
0 0 48 1
0 0 49 0
0 0 49 0
0 0 52 0
```

```

0 0 53 1
0 0 54 1
0 0 55 0
0 0 57 1
0 0 46 1
0 0 48 0
0 0 57 1
0 0 60 1
0 0 30 0
0 0 34 0
0 0 36 1
0 0 38 1
0 0 39 0
0 0 42 0
;
run;
```

CONTROL.DRUG1

```
data control.drug1 (label='JAN2018 DATA');
 input CHAR $8. NUM;
 datalines;
junk 0
junk 0
junk 0
junk 0
junk 0
junk 0
junk 0
;
run;
```

## CONTROL.DRUG2

[illegible]

[illegible][illegible]

```
data control.drug5 (label='JUL2017 DATA');
 input CHAR $8. NUM;
 datalines;
junk 0
;
run;
```

# CONTROL.GROUP

```
data control.group;
 input IDNUM $ 1-4 LNAME $ 5-18 FNAME $ 19-32 CITY $ 33-47 STATE $
 48-50 SEX $ 52-53 JOBCODE $ 54-57 SALARY comma8. BIRTH
 HIRED date9. HPHONE $ 86-97;
 format salary comma8.;
 format hired date9.;
 informat hired date9.;
 datalines;
```

1919 ADAMS	GERALD	STAMFORD	CT M TA2	34,377	11212	04JUN2007
203/781-1255						
1653 AHMAD	AZEEM	BRIDGEPORT	CT F ME2	35,109	9054	09AUG2010
203/675-7715						
1400 ALVAREZ	GLORIA	NEW YORK	NY M ME1	29,770	10170	16OCT2010
212/586-0808						
1350 ARTHUR	BARBARA	NEW YORK	NY F FA3	32,887	9374	29JUL2010
718/383-1549						
1401 AVERY	JERRY	PATERSON	NJ M TA3	38,823	3999	17NOV2005
201/732-8787						
1499 BAREFOOT	JOSEPH	PRINCETON	NJ M ME3	43,026	5229	07JUN2010
201/812-5665						
1101 BASQUEZ	RICHARDO	NEW YORK	NY M SCP	18,724	8192	01OCT2010
212/586-8060						
1333 BEAULIEU	ARMANDO	NEW YORK	NY M TA2	32,616	7759	10FEB2011
718/384-2849						
1402 BLALOCK	RALPH	NEW YORK	NY M TA2	32,615	8417	02DEC2010
718/384-2849						
1479 BOSTIC	MARIE	NEW YORK	NY F TA3	38,786	10583	05OCT2007
718/384-8816						
1403 BOWDEN	EARL	BRIDGEPORT	CT M ME1	28,073	10620	21DEC2016
203/675-3434						
1739 BOYCE	JONATHAN	NEW YORK	NY M PT1	66,518	9125	27JAN2011
212/587-1247						
1658 BRADLEY	JEREMY	NEW YORK	NY M SCP	17,944	9959	29FEB2012
212/587-3622						
1428 BRADY	CHRISTINE	STAMFORD	CT F PT1	68,768	7399	16NOV2011
203/781-1212						
1782 BROWN	JASON	STAMFORD	CT M ME2	35,346	7643	22FEB2012
203/781-0019						
1244 BRYANT	LEONARD	NEW YORK	NY M ME2	36,926	8643	17JAN2008
718/383-3334						
1383 BURNETTE	THOMAS	NEW YORK	NY M BCK	25,824	10251	20OCT2009
718/384-3569						
1574 CAHILL	MARSHALL	NEW YORK	NY M FA2	28,573	7422	20DEC2012
718/383-2338						
1789 CANALES	VIVIANA	NEW YORK	NY M SCP	18,327	6232	11APR2008
212/587-9000						
1404 CARTER	DONALD	NEW YORK	NY M PT2	91,377	4803	01JAN2010
718/384-2946						
1437 CARTER	DOROTHY	BRIDGEPORT	CT F FA3	33,105	7568	31AUG2004
203/675-4117						

1639 CARTER 203/781-8839	KAREN	STAMFORD	CT F TA3	40,261	6386 28JAN2014
1269 CASTON 203/781-3335	FRANKLIN	STAMFORD	CT M NA1	41,691	4506 28NOV2012
1065 CHAPMAN 718/384-5618	NEIL	NEW YORK	NY M ME2	35,091	1486 07JAN2007
1876 CHIN 212/588-5634	JACK	NEW YORK	NY M TA3	39,676	6714 27APR2015
1037 CHOW 203/781-8868	JANE	STAMFORD	CT F TA1	28,559	8866 13SEP2012
1129 COOK 718/383-2313	BRENDA	NEW YORK	NY F ME2	34,930	8012 17AUG2011
1988 COOPER 212/587-1228	ANTHONY	NEW YORK	NY M FA3	32,218	7273 18SEP2004
1405 DAVIDSON 201/732-2323	JASON	PATERSON	NJ M SCP	18,057	9560 26JAN2012
1430 DEAN 203/675-1647	SANDRA	BRIDGEPORT	CT F TA2	32,926	8094 27APR2017
1983 DEAN 718/384-1647	SHARON	NEW YORK	NY F FA3	33,420	789 30APR2005
1134 DELGADO 203/781-1528	MARIA	STAMFORD	CT F TA2	33,463	10656 21DEC2016
1118 DENNIS 718/383-1122	ROGER	NEW YORK	NY M PT3	111,380	1476 18DEC2000
1438 DESAI 203/781-2229	AAKASH	STAMFORD	CT F TA3	39,224	9205 18NOV2007
1125 DUNLAP 718/383-2094	DONNA	NEW YORK	NY F FA2	28,889	10539 11DEC2007
1475 EATON 718/383-2828	ALICIA	NEW YORK	NY F FA2	27,788	8019 13JUL2010
1117 EDGERTON 212/588-1239	JOSHUA	NEW YORK	NY M TA3	39,772	8556 13AUG2012
1935 FERNANDEZ 203/675-2962	KATRINA	BRIDGEPORT	CT F NA2	51,082	5200 16OCT2011
1124 FIELDS 914/455-2998	DIANA	WHITE PLAINS	NY F FA1	23,178	6765 01OCT2010
1422 FLETCHER 201/812-0902	MARIE	PRINCETON	NJ F FA1	22,455	8921 06APR2011
1616 FLOWERS 718/384-3329	ANNETTE	NEW YORK	NY F TA2	34,138	11017 04JUN2013
1406 FOSTER 203/675-6363	GERALD	BRIDGEPORT	CT M ME2	35,186	7737 17FEB2007
1120 GARCIA 718/384-4930	JACK	NEW YORK	NY M ME1	28,620	11942 07OCT2013
1094 GOMEZ 203/675-7181	ALAN	BRIDGEPORT	CT M FA1	22,269	11049 17APR2011
1389 GORDON 718/384-9326	LEVI	NEW YORK	NY M BCK	25,029	7135 18AUG2010
1905 GRAHAM 212/586-8815	ALVIN	NEW YORK	NY M PT1	65,112	11794 29MAY2012
1407 GRANT 914/468-1616	DANIEL	MT. VERNON	NY M PT1	68,097	10674 18MAR2010
1114 GREEN 212/588-1092	JANICE	NEW YORK	NY F TA2	32,929	10853 27JUN2007
1410 HARRIS 203/781-0937	CHARLES	STAMFORD	CT M PT2	84,686	9984 07NOV2006



1439 HARRISON 203/675-4987	FELICIA	BRIDGEPORT	CT F PT1	70,737	8831 10SEP2010
1409 HARTFORD 203/781-9697	RAYMOND	STAMFORD	CT M ME3	41,552	3761 22OCT2001
1408 HENDERSON 201/812-4789	WILLIAM	PRINCETON	NJ M TA2	34,139	7393 14OCT2007
1121 HERNANDEZ 718/384-3313	MICHAEL	NEW YORK	NY M ME1	29,113	7939 07DEC2011
1991 HOLMES 203/675-0007	GABRIEL	BRIDGEPORT	CT F TA1	27,646	8162 12DEC2012
1102 HOLMES 914/455-0976	SHANE	WHITE PLAINS	NY M TA2	34,543	7213 15APR2011
1356 HOLMES 212/586-8411	SHAWN	NEW YORK	NY M ME2	36,870	6478 22FEB2003
1545 HUNTER 203/781-1119	CLYDE	STAMFORD	CT M PT1	66,131	7163 29MAY2010
1292 HUNTER 203/675-4830	HELEN	BRIDGEPORT	CT F ME2	36,692	9067 02JUL2009
1440 JACKSON 203/781-0088	LAURA	STAMFORD	CT F ME2	35,758	8305 09APR2011
1368 JEPSEN 203/781-8413	RONALD	STAMFORD	CT M FA2	27,809	7832 03NOV2014
1369 JOHNSON 212/587-5385	ANTHONY	NEW YORK	NY M TA2	33,706	8032 13MAR2017
1411 JOHNSON 201/732-3678	JACKSON	PATERSON	NJ M FA2	27,266	7817 01DEC2017
1113 JONES 718/383-3003	LESLIE	NEW YORK	NY F FA1	22,368	10241 17OCT2011
1704 JOSHI 718/384-0049	ABHAY	NEW YORK	NY M BCK	25,466	9738 28JUN2007
1900 KING 718/383-3698	WILLIAM	NEW YORK	NY M ME2	35,106	8180 27OCT2007
1126 KOSTECKA 212/586-1229	NICHOLAS	NEW YORK	NY F TA3	40,900	8548 21NOV2010
1677 KRAMER 203/675-7432	JACKSON	BRIDGEPORT	CT M BCK	26,008	8709 27MAR2009
1441 LAWRENCE 201/812-3337	KATHY	PRINCETON	NJ F FA2	27,159	10915 23MAR2011
1421 LEE 914/468-9143	RUSSELL	MT. VERNON	NY M TA2	33,156	6947 28FEB2010
1119 LI 212/586-2344	JEFF	NEW YORK	NY M TA1	26,925	8206 06SEP2008
1834 LONG 718/384-0040	RUSSELL	NEW YORK	NY M BCK	26,897	8074 02JUL2012
1777 LUFKIN 718/383-4413	ROY	NEW YORK	NY M PT3	109,631	4283 21JUN2001
1663 MAHANNAHS 212/587-7742	SHANTHA	NEW YORK	NY M BCK	26,453	9872 11AUG2011
1106 MARSHBURN 203/781-1457	JASPER	STAMFORD	CT M PT2	89,633	6519 16AUG2004
1103 MCDANIEL 212/586-0013	RONDA	NEW YORK	NY F FA1	23,739	10273 23JUL2012
1477 MEYERS 203/675-8125	PRESTON	BRIDGEPORT	CT M FA2	28,567	8846 07MAR2008
1476 MONROE 203/781-2837	JOYCE	STAMFORD	CT F TA2	34,804	7890 17MAR2017

1379 MORGAN 203/781-2216	ALFRED	STAMFORD	CT M ME3	42,265	8515 10JUN2004
1104 MORGAN 718/383-9740	CHRISTOPHER	NEW YORK	NY M SCP	17,947	8515 10JUN2011
1009 MORGAN 212/586-7753	GEORGE	NEW YORK	NY M TA1	28,881	7000 26MAR2012
1412 MURPHEY 201/812-4414	JOHN	PRINCETON	NJ M ME1	27,800	6013 05DEC2011
1115 MURPHY 718/384-1982	ALICE	NEW YORK	NY F FA3	32,700	7539 28FEB2010
1128 NELSON 203/675-1166	FELICIA	BRIDGEPORT	CT F TA2	32,778	9274 20OCT2010
1442 NEWKIRK 201/812-3331	SANDRA	PRINCETON	NJ F PT2	84,537	9744 12APR2008
1417 NEWKIRK 201/732-6611	WILLIAM	PATERSON	NJ M NA2	52,271	8944 07MAR2009
1478 NEWTON 212/587-5549	JAMES	NEW YORK	NY M PT2	84,204	7160 24OCT2010
1673 NICHOLLS 203/781-7770	HENRY	STAMFORD	CT M BCK	25,478	7362 15JUL2011
1839 NORRIS 718/384-1767	DIANE	NEW YORK	NY F NA1	43,434	7638 03JUL2013
1347 O'NEAL 718/384-0230	BRYAN	NEW YORK	NY M TA3	40,080	10125 06SEP2014
1423 OSWALD 914/468-9171	LESLIE	MT. VERNON	NY F ME2	35,774	10361 19AUG2010
1200 OVERMAN 203/781-1835	MICHELLE	STAMFORD	CT F ME1	27,817	7680 14AUG2012
1970 PAPI 718/383-3895	PAOLO	NEW YORK	NY F FA1	22,616	9034 12MAR2011
1521 PAPIA 212/587-7603	ISMAEL	NEW YORK	NY M ME3	41,527	8502 13JUL2008
1354 PAPIA 914/455-2337	FRANCISCO	WHITE PLAINS	NY F SCP	18,336	7819 16JUN2012
1424 PATTERSON 212/587-8991	RENEE	NEW YORK	NY F FA2	28,979	7155 11DEC2009
1132 PEARCE 718/384-1986	CAROL	NEW YORK	NY F FA1	22,414	4533 22OCT2003
1845 PEARSON 718/384-2311	JAMES	NEW YORK	NY M BCK	25,997	7263 22MAR2000
1556 PENNINGTON 718/383-5681	MICHAEL	NEW YORK	NY M PT1	71,350	8939 11DEC2011
1413 PETERS 201/812-2478	RANDALL	PRINCETON	NJ M FA2	27,436	5737 02JAN2010
1123 PETERSON 718/383-0077	SUZANNE	NEW YORK	NY F TA1	28,407	8339 05DEC2012
1907 PHELPS 203/781-1118	WILLIAM	STAMFORD	CT M TA2	33,330	7624 06JUL2007
1436 PORTER 718/383-5777	SUSAN	NEW YORK	NY F TA2	34,476	8928 12MAR2007
1385 RAYNOR 203/675-2846	MILTON	BRIDGEPORT	CT M ME3	43,901	8051 01APR2016
1432 REED 914/468-5454	MARILYN	MT. VERNON	NY F ME2	35,328	7977 10FEB2015
1111 RHODES 201/812-1837	JEREMY	PRINCETON	NJ M NA1	40,587	7977 31OCT2012

1116 RICHARDS 212/587-1224	CASEY	NEW YORK	NY F FA1	22,863	7210 21MAR2011
1352 RIVERS 718/383-3345	SIMON	NEW YORK	NY M NA2	53,799	7641 16OCT2006
1555 RODRIGUEZ 203/675-2401	JULIA	BRIDGEPORT	CT F FA2	27,500	6649 04JUL2012
1038 RODRIGUEZ 203/675-2048	MARIA	BRIDGEPORT	CT F TA1	26,534	10905 23NOV2011
1420 ROUSE 201/732-9834	JEREMY	PATERSON	NJ M ME3	43,072	9181 22JUL2017
1561 SANDERS 212/588-6615	RAYMOND	NEW YORK	NY M TA2	34,515	8734 07OCT2007
1434 SANDERSON 203/781-1333	EDITH	STAMFORD	CT F FA2	28,623	8227 28OCT2010
1414 SARKAR 203/675-1715	ABHEEK	BRIDGEPORT	CT M FA1	23,645	8118 12APR2012
1112 SAYERS 718/384-4895	RANDY	NEW YORK	NY M TA1	26,906	9099 07DEC2012
1390 SMART 718/383-1141	JONATHAN	NEW YORK	NY M FA2	27,762	9181 23JUN2011
1332 STEPHENSON 203/675-1497	ADAM	BRIDGEPORT	CT M NA1	42,179	7565 04JUN2011
1890 STEPHENSON 718/384-9874	ROBERT	NEW YORK	NY M PT2	85,897	7871 25NOV2017
1429 THOMPSON 203/781-3857	ALICE	STAMFORD	CT F TA1	27,940	7363 07AUG2012
1107 THOMPSON 718/384-3785	WAYNE	NEW YORK	NY M PT2	89,978	5273 10FEB2009
1908 TRENTON 212/586-6262	MELISSA	NEW YORK	NY F TA2	32,996	7283 23APR2010
1830 TRIPP 203/675-2479	KATHY	BRIDGEPORT	CT F PT2	84,472	6356 29JAN2013
1882 TUCKER 718/384-0216	ALAN	NEW YORK	NY M ME3	41,539	6400 21NOV2008
1050 TUTTLE 914/455-2119	THOMAS	WHITE PLAINS	NY M ME2	35,168	8595 24AUG2016
1425 UNDERWOOD 203/781-0978	JENNY	STAMFORD	CT F FA1	23,980	8032 28FEB2013
1928 UPCHURCH 914/455-5009	LARRY	WHITE PLAINS	NY M PT2	89,859	5372 13JUL2010
1480 UPDIKE 212/587-8729	THERESA	NEW YORK	NY F TA3	39,584	6455 25MAR2001
1100 VANDEUSEN 212/586-2531	RICHARD	NEW YORK	NY M BCK	25,005	7640 07MAY2008
1995 VARNER 718/384-7113	ELIZABETH	NEW YORK	NY F ME1	28,811	8636 19SEP2013
1135 VEGA 718/384-5913	ANNA	NEW YORK	NY F FA2	27,322	7568 31MAR2010
1415 VEGA 718/384-2823	FRANKLIN	NEW YORK	NY M FA2	28,279	6642 12FEB2008
1076 VENTER 718/383-2321	RANDALL	NEW YORK	NY M PT1	66,559	5765 03OCT2011
1426 VICK 201/812-2424	THERESA	PRINCETON	NJ F TA2	32,992	9835 25JUN2010
1564 WALTERS 212/587-3257	ANNE	NEW YORK	NY F SCP	18,834	8137 01JUL2012

```

1221 WALTERS DIANE NEW YORK NY F FA2 27,897 10126 04OCT2011
718/384-1918
1133 WANG CHIN NEW YORK NY M TA1 27,702 9690 12FEB2012
212/587-1956
1435 WARD ELAINE NEW YORK NY F TA3 38,809 7071 08FEB2010
718/383-4987
1418 WATSON BERNARD NEW YORK NY M ME1 28,006 6297 06JAN2012
718/383-1298
1017 WELCH DARIUS NEW YORK NY M TA3 40,859 6571 16OCT2011
212/586-5535
1443 WELLS AGNES STAMFORD CT F NA1 42,275 10548 29AUG2011
203/781-5546
1131 WELLS NADINE NEW YORK NY F TA2 32,576 8030 19APR2011
718/383-1045
1427 WHALEY CAROLYN MT. VERNON NY F TA2 34,047 11261 30JAN2010
914/468-4528
1036 WONG LESLIE NEW YORK NY F TA3 39,393 9270 23OCT2004
212/587-2570
1130 WOOD DEBORAH NEW YORK NY F FA1 23,917 7806 05JUN2012
212/587-0013
1127 WOOD SANDRA NEW YORK NY F TA2 33,012 9079 07DEC2006
212/587-2881
1433 YANCEY ROBIN PRINCETON NJ F FA3 32,983 9685 17JAN2007
201/812-1874
1431 YOUNG DEBORAH STAMFORD CT F FA3 33,231 8926 05APR2008
203/781-2987
1122 YOUNG JOANN NEW YORK NY F FA2 27,957 8521 27NOV2008
718/384-2021
1105 YOUNG LAWRENCE NEW YORK NY M ME2 34,806 8095 13AUG2010
718/384-0008
;
run;

```

---

## CONTROL.MLSCL

```

data control.mlscl (label='Multiple Sclerosis Data');
 input GROUP OBS1 OBS2 WT;
 datalines;
 1 4 4 10
 1 4 1 3
 1 4 2 7
 1 4 3 3
 1 1 4 1
 1 1 1 38
 1 1 2 5
 1 1 3 0
 1 2 4 0
 1 2 1 33
 1 2 2 11
 1 2 3 3
 1 3 4 6
 1 3 1 10
 1 3 2 14
 ;

```

```

1 3 3 5
2 4 1 1
2 4 2 2
2 4 3 4
2 4 4 14
2 3 1 2
2 3 2 13
2 3 3 3
2 3 4 4
2 2 1 3
2 2 2 11
2 2 3 4
2 2 4 0
2 1 1 5
2 1 2 3
2 1 3 0
2 1 4 0
;
run;

```

---

## CONTROL.NAMES

```

data control.names;
 input LABEL $ 1-16 START $ 17-24 FMTNAME $ 31-35 TYPE $ 41-41;
 datalines;
Capalleti, Jimmy 2355 bonus C
Chen, Len 5889 bonus C
Davis, Brad 3878 bonus C
Leung, Brenda 4409 bonus C
Patel, Mary 2398 bonus C
Smith, Robert 5862 bonus C
Zook, Carla 7385 bonus C
;
run;

```

---

## CONTROL.OXYGEN

```

data control.oxygen;
 input AGE WEIGHT RUNTIME RSTPULSE RUNPULSE MAXPULSE OXYGEN;
 datalines;
44 89.47 11.37 62 178 182 44.609
40 75.07 10.07 62 185 185 45.313
44 85.84 8.65 45 156 168 54.297
42 68.15 8.17 40 166 172 59.571
38 89.02 9.22 55 178 180 49.874
47 77.45 11.63 58 176 176 44.811
40 75.98 11.95 70 176 180 45.681
43 81.19 10.85 64 162 170 49.091
44 81.42 13.08 63 174 176 39.442

```

```

38 81.87 8.63 48 170 186 60.055
44 73.03 10.13 45 168 168 50.541
45 87.66 14.03 56 186 192 37.388
45 66.45 11.12 51 176 176 44.754
47 79.15 10.60 47 162 164 47.273
54 83.12 10.33 50 166 170 51.855
49 81.42 8.95 44 180 185 49.156
51 69.63 10.95 57 168 172 40.836
51 77.91 10.00 48 162 168 46.672
48 91.63 10.25 48 162 164 46.774
49 73.37 10.08 76 168 168 50.388
57 73.37 12.63 58 174 176 39.407
54 79.38 11.17 62 156 165 46.080
52 76.32 9.63 48 164 166 45.441
50 70.87 8.92 48 146 155 54.625
51 67.25 11.08 48 172 172 45.118
54 91.63 12.88 44 168 172 39.203
51 73.71 10.47 59 186 188 45.790
57 59.08 9.93 49 148 155 50.545
49 76.32 9.40 56 186 188 48.673
48 61.24 11.50 52 170 176 47.920
52 82.78 10.50 53 170 172 47.467
;
run;

```

---

## CONTROL.PERSONL

```

data control.personl;
 input IDNUM $ 1-4 LNAME $ 4-18 FNAME $ 19-33 CITY $ 32-46 STATE $ 47-48
 SEX $ 50-50 JOBCODE $ 52-54 SALARY BIRTH date. @63
 HIRED date9. @73 HPHONE $ 84-99;
 format birth date9.;
 informat birth date.;
 format hired date9.;
 informat hired date.;
 datalines;
1919 ADAMS GERALD STAMFORD CT M TA2 34376 12SEP1990 04JUN2007
203/781-1255
1653 AHMAD AZEEM BRIDGEPORT CT F ME2 35108 15OCT1984 09AUG2010
203/675-7715
1400 ALVAREZ GLORIA NEW YORK NY M ME1 29769 05NOV1987 16OCT2010
212/586-0808
1350 ARTHUR BARBARA NEW YORK NY F FA3 32886 31AUG1985 29JUL2010
718/383-1549
1401 AVERY JERRY PATERSON NJ M TA3 38822 13DEC1970 17NOV2005
201/732-8787
1499 BAREFOOT JOSEPH PRINCETON NJ M ME3 43025 26APR1974 07JUN2010
201/812-5665
1101 BASQUEZ RICHARDO NEW YORK NY M SCP 18723 06JUN1982 01OCT2010
212/586-8060
1333 BEAULIEU ARMANDO STAMFORD CT M PT2 88606 30MAR1981 10FEB2011
203/781-1777

```

1402 BLALOCK 718/384-2849	RALPH	NEW YORK	NY M TA2	32615	17JAN1983	02DEC2010
1479 BOSTIC 718/384-8816	MARIE	NEW YORK	NY F TA3	38785	22DEC1988	05OCT2007
1403 BOWDEN 203/675-3434	EARL	BRIDGEPORT	CT M ME1	28072	28JAN1989	21DEC2016
1739 BOYCE 212/587-1247	JONATHAN	NEW YORK	NY M PT1	66517	25DEC1984	27JAN2011
1658 BRADLEY 212/587-3622	JEREMY	NEW YORK	NY M SCP	17943	08APR1987	29FEB2012
1428 BRADY 203/781-1212	CHRISTINE	STAMFORD	CT F PT1	68767	04APR1980	16NOV2011
1782 BROWN 203/781-0019	JASON	STAMFORD	CT M ME2	35345	04DEC1980	22FEB2012
1244 BRYANT 718/383-3334	LEONARD	NEW YORK	NY M ME2	36925	31AUG1983	17JAN2008
1383 BURNETTE 718/384-3569	THOMAS	NEW YORK	NY M BCK	25824	25JAN1985	20OCT2009
1574 CAHILL 718/383-2338	MARSHALL	NEW YORK	NY M FA2	28572	27APR1980	20DEC2012
1789 CANALES 212/587-9000	VIVIANA	NEW YORK	NY M SCP	18326	23JAN1977	11APR2008
1404 CARTER 718/384-2946	DONALD	NEW YORK	NY M PT2	91376	24FEB1973	01JAN2010
1437 CARTER 203/675-4117	DOROTHY	BRIDGEPORT	CT F FA3	33104	20SEP1980	31AUG2004
1639 CARTER 203/781-8839	KAREN	STAMFORD	CT F TA3	40260	26JUN1977	28JAN2014
1269 CASTON 203/781-3335	FRANKLIN	STAMFORD	CT M NA1	41690	03MAY1972	28NOV2012
1065 CHAPMAN 718/384-5618	NEIL	NEW YORK	NY M ME2	35090	26JAN1977	07JAN2007
1876 CHIN 212/588-5634	JACK	NEW YORK	NY M TA3	39675	20MAY1978	27APR2015
1037 CHOW 203/781-8868	JANE	STAMFORD	CT F TA1	28558	10APR1984	13SEP2012
1129 COOK 718/383-2313	BRENDA	NEW YORK	NY F ME2	34929	08DEC1981	17AUG2011
1988 COOPER 212/587-1228	ANTHONY	NEW YORK	NY M FA3	32217	30NOV1979	18SEP2004
1405 DAVIDSON 201/732-2323	JASON	PATERSON	NJ M SCP	18056	05MAR1986	26JAN2012
1430 DEAN 203/675-1647	SANDRA	BRIDGEPORT	CT F TA2	32925	28FEB1982	27APR2017
1983 DEAN 718/384-1647	SHARON	NEW YORK	NY F FA3	33419	28FEB1962	30APR2005
1134 DELGADO 203/781-1528	MARIA	STAMFORD	CT F TA2	33462	05MAR1989	21DEC2016
1118 DENNIS 718/383-1122	ROGER	NEW YORK	NY M PT3	111379	16JAN1964	18DEC2000
1438 DESAI 203/781-2229	AAKASH	STAMFORD	CT F TA3	39223	15MAR1985	18NOV2007
1125 DUNLAP 718/383-2094	DONNA	NEW YORK	NY F FA2	28888	08NOV1988	11DEC2007
1475 EATON 718/383-2828	ALICIA	NEW YORK	NY F FA2	27787	15DEC1981	13JUL2010

1117 EDGERTON 212/588-1239	JOSHUA	NEW YORK	NY M TA3	39771 05JUN1983 13AUG2012
1935 FERNANDEZ 203/675-2962	KATRINA	BRIDGEPORT	CT F NA2	51081 28MAR1974 16OCT2011
1124 FIELDS 914/455-2998	DIANA	WHITE PLAINS	NY F FA1	23177 10JUL1978 01OCT2010
1422 FLETCHER 201/812-0902	MARIE	PRINCETON	NJ F FA1	22454 04JUN1984 06APR2011
1616 FLOWERS 718/384-3329	ANNETTE	NEW YORK	NY F TA2	34137 01MAR1990 04JUN2013
1406 FOSTER 203/675-6363	GERALD	BRIDGEPORT	CT M ME2	35185 08MAR1981 17FEB2007
1120 GARCIA 718/384-4930	JACK	NEW YORK	NY M ME1	28619 11SEP1992 07OCT2013
1094 GOMEZ 203/675-7181	ALAN	BRIDGEPORT	CT M FA1	22268 02APR1990 17APR2011
1389 GORDON 718/384-9326	LEVI	NEW YORK	NY M BCK	25028 15JUL1979 18AUG2010
1905 GRAHAM 212/586-8815	ALVIN	NEW YORK	NY M PT1	65111 16APR1992 29MAY2012
1407 GRANT 914/468-1616	DANIEL	MT. VERNON	NY M PT1	68096 23MAR1989 18MAR2010
1114 GREEN 212/588-1092	JANICE	NEW YORK	NY F TA2	32928 18SEP1989 27JUN2007
1410 HARRIS 203/781-0937	CHARLES	STAMFORD	CT M PT2	84685 03MAY1987 07NOV2006
1439 HARRISON 203/675-4987	FELICIA	BRIDGEPORT	CT F PT1	70736 06MAR1984 10SEP2010
1409 HARTFORD 203/781-9697	RAYMOND	STAMFORD	CT M ME3	41551 19APR1970 22OCT2001
1408 HENDERSON 201/812-4789	WILLIAM	PRINCETON	NJ M TA2	34138 29MAR1980 14OCT2007
1121 HERNANDEZ 718/384-3313	MICHAEL	NEW YORK	NY M ME1	29112 26SEP1981 07DEC2011
1991 HOLMES 203/675-0007	GABRIEL	BRIDGEPORT	CT F TA1	27645 07MAY1982 12DEC2012
1102 HOLMES 914/455-0976	SHANE	WHITE PLAINS	NY M TA2	34542 01OCT1979 15APR2011
1356 HOLMES 212/586-8411	SHAWN	NEW YORK	NY M ME2	36869 26SEP1977 22FEB2003
1545 HUNTER 203/781-1119	CLYDE	STAMFORD	CT M PT1	66130 12AUG1979 29MAY2010
1292 HUNTER 203/675-4830	HELEN	BRIDGEPORT	CT F ME2	36691 28OCT1984 02JUL2009
1440 JACKSON 203/781-0088	LAURA	STAMFORD	CT F ME2	35757 27SEP1982 09APR2011
1368 JEPSEN 203/781-8413	RONALD	STAMFORD	CT M FA2	27808 11JUN1981 03NOV2014
1369 JOHNSON 212/587-5385	ANTHONY	NEW YORK	NY M TA2	33705 28DEC1981 13MAR2017
1411 JOHNSON 201/732-3678	JACKSON	PATERSON	NJ M FA2	27265 27MAY1981 01DEC2017
1113 JONES 718/383-3003	LESLIE	NEW YORK	NY F FA1	22367 15JAN1988 17OCT2011
1704 JOSHI 718/384-0049	ABHAY	NEW YORK	NY M BCK	25465 30AUG1986 28JUN2007



1900 KING 718/383-3698	WILLIAM	NEW YORK	NY M ME2	35105	25MAY1982	27OCT2007
1126 KOSTECKA 212/586-1229	NICHOLAS	NEW YORK	NY F TA3	40899	28MAY1983	21NOV2010
1677 KRAMER 203/675-7432	JACKSON	BRIDGEPORT	CT M BCK	26007	05NOV1983	27MAR2009
1441 LAWRENCE 201/812-3337	KATHY	PRINCETON	NJ F FA2	27158	19NOV1989	23MAR2011
1421 LEE 914/468-9143	RUSSELL	MT. VERNON	NY M TA2	33155	08JAN1979	28FEB2010
1119 LI 212/586-2344	JEFF	NEW YORK	NY M TA1	26924	20JUN1982	06SEP2008
1834 LONG 718/384-0040	RUSSELL	NEW YORK	NY M BCK	26896	08FEB1982	02JUL2012
1777 LUFKIN 718/383-4413	ROY	NEW YORK	NY M PT3	109630	23SEP1971	21JUN2001
1663 MAHANNAHS 212/587-7742	SHANTHA	NEW YORK	NY M BCK	26452	11JAN1987	11AUG2011
1106 MARSHBURN 203/781-1457	JASPER	STAMFORD	CT M PT2	89632	06NOV1977	16AUG2004
1103 MCDANIEL 212/586-0013	RONDA	NEW YORK	NY F FA1	23738	16FEB1988	23JUL2012
1477 MEYERS 203/675-8125	PRESTON	BRIDGEPORT	CT M FA2	28566	21MAR1984	07MAR2008
1476 MONROE 203/781-2837	JOYCE	STAMFORD	CT F TA2	34803	30MAY1986	17MAR2017
1379 MORGAN 203/781-2216	ALFRED	STAMFORD	CT M ME3	42264	08AUG1981	10JUN2004
1104 MORGAN 718/383-9740	CHRISTOPHER	NEW YORK	NY M SCP	17946	25APR1983	10JUN2011
1009 MORGAN 212/586-7753	GEORGE	NEW YORK	NY M TA1	28880	02MAR1979	26MAR2012
1412 MURPHEY 201/812-4414	JOHN	PRINCETON	NJ M ME1	27799	18JUN1976	05DEC2011
1115 MURPHY 718/384-1982	ALICE	NEW YORK	NY F FA3	32699	22AUG1980	28FEB2010
1128 NELSON 203/675-1166	FELICIA	BRIDGEPORT	CT F TA2	32777	23MAY1985	20OCT2010
1442 NEWKIRK 201/812-3331	SANDRA	PRINCETON	NJ F PT2	84536	05SEP1986	12APR2008
1417 NEWKIRK 201/732-6611	WILLIAM	PATERSON	NJ M NA2	52270	27JUN1984	07MAR2009
1478 NEWTON 212/587-5549	JAMES	NEW YORK	NY M PT2	84203	09AUG1979	24OCT2010
1673 NICHOLLS 203/781-7770	HENRY	STAMFORD	CT M BCK	25477	27FEB1980	15JUL2011
1839 NORRIS 718/384-1767	DIANE	NEW YORK	NY F NA1	44433	29NOV1980	03JUL2013
1347 O'NEAL 718/384-0230	BRYAN	NEW YORK	NY M TA3	40079	21SEP1987	06SEP2014
1423 OSWALD 914/468-9171	LESLIE	MT. VERNON	NY F ME2	35773	14MAY1988	19AUG2010
1200 OVERMAN 203/781-1835	MICHELLE	STAMFORD	CT F ME1	27816	10JAN1981	14AUG2012
1970 PAPI 718/383-3895	PAOLO	NEW YORK	NY F FA1	22615	25SEP1984	12MAR2011

1521 PAPIA 212/587-7603	ISMAEL	NEW YORK	NY M ME3	41526 12APR1983 13JUL2008
1354 PAPIA 914/455-2337	FRANCISCO	WHITE PLAINS	NY F SCP	18335 29MAY1981 16JUN2012
1424 PATTERSON 212/587-8991	RENEE	NEW YORK	NY F FA2	28978 04AUG1979 11DEC2009
1132 PEARCE 718/384-1986	CAROL	NEW YORK	NY F FA1	22413 30MAY1972 22OCT2003
1845 PEARSON 718/384-2311	JAMES	NEW YORK	NY M BCK	25996 20NOV1979 22MAR2000
1556 PENNINGTON 718/383-5681	MICHAEL	NEW YORK	NY M PT1	71349 22JUN1984 11DEC2011
1413 PETERS 201/812-2478	RANDALL	PRINCETON	NJ M FA2	27435 16SEP1975 02JAN2010
1123 PETERSON 718/383-0077	SUZANNE	NEW YORK	NY F TA1	28407 31OCT1982 05DEC2012
1907 PHELPS 203/781-1118	WILLIAM	STAMFORD	CT M TA2	33329 15NOV1980 06JUL2007
1436 PORTER 718/383-5777	SUSAN	NEW YORK	NY F TA2	34475 11JUN1984 12MAR2007
1385 RAYNOR 203/675-2846	MILTON	BRIDGEPORT	CT M ME3	43900 16JAN1982 01APR2016
1432 REED 914/468-5454	MARILYN	MT. VERNON	NY F ME2	35327 03NOV1981 10FEB2015
1111 RHODES 201/812-1837	JEREMY	PRINCETON	NJ M NA1	40586 14JUL1983 31OCT2012
1116 RICHARDS 212/587-1224	CASEY	NEW YORK	NY F FA1	22862 28SEP1979 21MAR2011
1352 RIVERS 718/383-3345	SIMON	NEW YORK	NY M NA2	53798 02DEC1980 16OCT2006
1555 RODRIGUEZ 203/675-2401	JULIA	BRIDGEPORT	CT F FA2	27499 16MAR1978 04JUL2012
1038 RODRIGUEZ 203/675-2048	MARIA	BRIDGEPORT	CT F TA1	26533 09NOV1989 23NOV2011
1420 ROUSE 201/732-9834	JEREMY	PATERSON	NJ M ME3	43071 19FEB1985 22JUL2017
1561 SANDERS 212/588-6615	RAYMOND	NEW YORK	NY M TA2	34514 30NOV1983 07OCT2007
1434 SANDERSON 203/781-1333	EDITH	STAMFORD	CT F FA2	28622 11JUL1982 28OCT2010
1414 SARKAR 203/675-1715	ABHEEK	BRIDGEPORT	CT M FA1	23644 24MAR1982 12APR2012
1112 SAYERS 718/384-4895	RANDY	NEW YORK	NY M TA1	26905 29NOV1984 07DEC2012
1390 SMART 718/383-1141	JONATHAN	NEW YORK	NY M FA2	27761 19FEB1985 23JUN2011
1332 STEPHENSON 203/675-1497	ADAM	BRIDGEPORT	CT M NA1	42178 17SEP1980 04JUN2011
1890 STEPHENSON 718/384-9874	ROBERT	NEW YORK	NY M PT2	85896 20JUL1981 25NOV2017
1429 THOMPSON 203/781-3857	ALICE	STAMFORD	CT F TA1	27939 28FEB1980 07AUG2012
1107 THOMPSON 718/384-3785	WAYNE	NEW YORK	NY M PT2	89977 09JUN1974 10FEB2009
1908 TRENTON 212/586-6262	MELISSA	NEW YORK	NY F TA2	32995 10DEC1979 23APR2010

1830 TRIPP 203/675-2479	KATHY	BRIDGEPORT	CT F PT2	84471	27MAY1977	29JAN2013
1882 TUCKER 718/384-0216	ALAN	NEW YORK	NY M ME3	41538	10JUL1977	21NOV2008
1050 TUTTLE 914/455-2119	THOMAS	WHITE PLAINS	NY M ME2	35167	14JUL1983	24AUG2016
1425 UNDERWOOD 203/781-0978	JENNY	STAMFORD	CT F FA1	23979	28DEC1981	28FEB2013
1928 UPCHURCH 914/455-5009	LARRY	WHITE PLAINS	NY M PT2	89858	16SEP1974	13JUL2010
1480 UPDIKE 212/587-8729	THERESA	NEW YORK	NY F TA3	39583	03SEP1977	25MAR2001
1100 VANDEUSEN 212/586-2531	RICHARD	NEW YORK	NY M BCK	25004	01DEC1980	07MAY2008
1995 VARNER 718/384-7113	ELIZABETH	NEW YORK	NY F ME1	28810	24AUG1983	19SEP2013
1135 VEGA 718/384-5913	ANNA	NEW YORK	NY F FA2	27321	20SEP1980	31MAR2010
1415 VEGA 718/384-2823	FRANKLIN	NEW YORK	NY M FA2	28278	09MAR1978	12FEB2008
1076 VENTER 718/383-2321	RANDALL	NEW YORK	NY M PT1	66558	14OCT1975	03OCT2011
1426 VICK 201/812-2424	THERESA	PRINCETON	NJ F TA2	32991	05DEC1986	25JUN2010
1564 WALTERS 212/587-3257	ANNE	NEW YORK	NY F SCP	18833	12APR1982	01JUL2012
1221 WALTERS 718/384-1918	DIANE	NEW YORK	NY F FA2	27896	22SEP1987	04OCT2011
1133 WANG 212/587-1956	CHIN	NEW YORK	NY M TA1	27701	13JUL1986	12FEB2012
1435 WARD 718/383-4987	ELAINE	NEW YORK	NY F TA3	38808	12MAY1979	08FEB2010
1418 WATSON 718/383-1298	BERNARD	NEW YORK	NY M ME1	28005	29MAR1977	06JAN2012
1017 WELCH 212/586-5535	DARIUS	NEW YORK	NY M TA3	40858	28DEC1977	16OCT2011
1443 WELLS 203/781-5546	AGNES	STAMFORD	CT F NA1	42274	17NOV1988	29AUG2011
1131 WELLS 718/383-1045	NADINE	NEW YORK	NY F TA2	32575	26DEC1981	19APR2011
1427 WHALEY 914/468-4528	CAROLYN	MT. VERNON	NY F TA2	34046	31OCT1990	30JAN2010
1036 WONG 212/587-2570	LESLIE	NEW YORK	NY F TA3	39392	19MAY1985	23OCT2004
1130 WOOD 212/587-0013	DEBORAH	NEW YORK	NY F FA1	23916	16MAY1981	05JUN2012
1127 WOOD 212/587-2881	SANDRA	NEW YORK	NY F TA2	33011	09NOV1984	07DEC2006
1433 YANCEY 201/812-1874	ROBIN	PRINCETON	NJ F FA3	32982	08JUL1986	17JAN2007
1431 YOUNG 203/781-2987	DEBORAH	STAMFORD	CT F FA3	33230	09JUN1984	05APR2008
1122 YOUNG 718/384-2021	JOANN	NEW YORK	NY F FA2	27956	01MAY1983	27NOV2008
1105 YOUNG 718/384-0008	LAWRENCE	NEW YORK	NY M ME2	34805	01MAR1982	13AUG2010

```
;
run;
```

---

## CONTROL.PHARM

```
data control.pharm (label='Sugar Study');
 input DRUG $8. RESPONSE $8. WT ;
 datalines;
 A cured 14
 A uncured 22
 B cured 24
 B uncured 19
 C cured 17
 C uncured 13
 ;
run;
```

---

## CONTROL.POINTS

```
data control.points;
 input EMPID $8. Q1 Q2 Q3 Q4 TOTPTS;
 datalines;
2355 3 4 4 3 14
5889 2 2 2 2 8
3878 1 2 2 2 7
4409 0 1 1 1 3
2398 2 2 1 1 6
5862 1 1 1 2 5
 ;
run;
```

---

## CONTROL.PRENAT

```
data control.prenat;
 input IDNUM $ 1-4 LNAME $ 6-20 FNAME $ 22-36 CITY $ 39-53
 STATE $ 55-56 HPHONE $ 58-69;
 datalines;
1919 ADAMS GERALD STAMFORD CT 203/781-1255
1653 ALIBRANDI MARIA BRIDGEPORT CT 203/675-7715
1400 ALHERTANI ABDULLAH NEW YORK NY 212/586-0808
1350 ALVAREZ MERCEDES NEW YORK NY 718/383-1549
1401 ALVAREZ CARLOS PATERSON NJ 201/732-8787
1499 BAREFOOT JOSEPH PRINCETON NJ 201/812-5665
1101 BAUCOM WALTER NEW YORK NY 212/586-8060
1333 BANADYGA JUSTIN STAMFORD CT 203/781-1777
```

1402	BLALOCK	RALPH	NEW YORK	NY 718/384-2849
1479	BALLETTI	MARIE	NEW YORK	NY 718/384-8816
1403	BOWDEN	EARL	BRIDGEPORT	CT 203/675-3434
1739	BRANCACCIO	JOSEPH	NEW YORK	NY 212/587-1247
1658	BREUHAUS	JEREMY	NEW YORK	NY 212/587-3622
1428	BRADY	CHRISTINE	STAMFORD	CT 203/781-1212
1782	BREWCZAK	JAKOB	STAMFORD	CT 203/781-0019
1244	BUCCI	ANTHONY	NEW YORK	NY 718/383-3334
1383	BURNETTE	THOMAS	NEW YORK	NY 718/384-3569
1574	CAHILL	MARSHALL	NEW YORK	NY 718/383-2338
1789	CARAWAY	DAVIS	NEW YORK	NY 212/587-9000
1404	COHEN	LEE	NEW YORK	NY 718/384-2946
1437	CARTER	DOROTHY	BRIDGEPORT	CT 203/675-4117
1639	CARTER-COHEN	KAREN	STAMFORD	CT 203/781-8839
1269	CASTON	FRANKLIN	STAMFORD	CT 203/781-3335
1065	COPAS	FREDERICO	NEW YORK	NY 718/384-5618
1876	CHIN	JACK	NEW YORK	NY 212/588-5634
1037	CHOW	JANE	STAMFORD	CT 203/781-8868
1129	COUNIHAN	BRENDA	NEW YORK	NY 718/383-2313
1988	COOPER	ANTHONY	NEW YORK	NY 212/587-1228
1405	DACKO	JASON	PATERSON	NJ 201/732-2323
1430	DABROWSKI	SANDRA	BRIDGEPORT	CT 203/675-1647
1983	DEAN	SHARON	NEW YORK	NY 718/384-1647
1134	DELGADO	MARIA	STAMFORD	CT 203/781-1528
1118	DENNIS	ROGER	NEW YORK	NY 718/383-1122
1438	DABBOUSSI	KAMILLA	STAMFORD	CT 203/781-2229
1125	DUNLAP	DONNA	NEW YORK	NY 718/383-2094
1475	ELGES	MARGARETE	NEW YORK	NY 718/383-2828
1117	EDGERTON	JOSHUA	NEW YORK	NY 212/588-1239
1935	FERNANDEZ	KATRINA	BRIDGEPORT	CT 203/675-2962
1124	FIELDS	DIANA	WHITE PLAINS	NY 914/455-2998
1422	FUJIHARA	KYOKO	PRINCETON	NJ 201/812-0902
1616	FUENTAS	CARLA	NEW YORK	NY 718/384-3329
1406	FOSTER	GERALD	BRIDGEPORT	CT 203/675-6363
1120	GARCIA	JACK	NEW YORK	NY 718/384-4930
1094	GOMEZ	ALAN	BRIDGEPORT	CT 203/675-7181
1389	GOLDSTEIN	LEVI	NEW YORK	NY 718/384-9326
1905	GRAHAM	ALVIN	NEW YORK	NY 212/586-8815
1407	GREGORSKI	DANIEL	MT. VERNON	NY 914/468-1616
1114	GREENWALD	JANICE	NEW YORK	NY 212/588-1092
1410	HARRIS	CHARLES	STAMFORD	CT 203/781-0937
1439	HASENHAUER	CHRISTINA	BRIDGEPORT	CT 203/675-4987
1409	HAVELKA	RAYMOND	STAMFORD	CT 203/781-9697
1408	HENDERSON	WILLIAM	PRINCETON	NJ 201/812-4789
1121	HERNANDEZ	ROBERTO	NEW YORK	NY 718/384-3313
1991	HOWARD	GRETCHEN	BRIDGEPORT	CT 203/675-0007
1102	HERMANN	JOACHIM	WHITE PLAINS	NY 914/455-0976
1356	HOWARD	MICHAEL	NEW YORK	NY 212/586-8411
1545	HERRERO	CLYDE	STAMFORD	CT 203/781-1119
1292	HUNTER	HELEN	BRIDGEPORT	CT 203/675-4830
1440	JACKSON	LAURA	STAMFORD	CT 203/781-0088
1368	JEPSEN	RONALD	STAMFORD	CT 203/781-8413
1369	JONSON	ANTHONY	NEW YORK	NY 212/587-5385
1411	JOHNSEN	JACK	PATERSON	NJ 201/732-3678
1113	JOHNSON	LESLIE	NEW YORK	NY 718/383-3003
1704	JONES	NATHAN	NEW YORK	NY 718/384-0049

1900	KING	WILLIAM	NEW YORK	NY 718/383-3698
1126	KIMANI	ANNE	NEW YORK	NY 212/586-1229
1677	KRAMER	JACKSON	BRIDGEPORT	CT 203/675-7432
1441	LAWRENCE	KATHY	PRINCETON	NJ 201/812-3337
1421	LEE	RUSSELL	MT. VERNON	NY 914/468-9143
1119	LI	JEFF	NEW YORK	NY 212/586-2344
1834	LEBLANC	RUSSELL	NEW YORK	NY 718/384-0040
1777	LUFKIN	ROY	NEW YORK	NY 718/383-4413
1663	MARKS	JOHN	NEW YORK	NY 212/587-7742
1106	MARSHBURN	JASPER	STAMFORD	CT 203/781-1457
1103	MCDANIEL	RONDA	NEW YORK	NY 212/586-0013
1477	MEYERS	PRESTON	BRIDGEPORT	CT 203/675-8125
1476	MONROE	JOYCE	STAMFORD	CT 203/781-2837
1379	MORGAN	ALFRED	STAMFORD	CT 203/781-2216
1104	MORGAN	CHRISTOPHER	NEW YORK	NY 718/383-9740
1009	MORGAN	GEORGE	NEW YORK	NY 212/586-7753
1412	MURPHEY	JOHN	PRINCETON	NJ 201/812-4414
1115	MURPHY	ALICE	NEW YORK	NY 718/384-1982
1128	NELSON	FELICIA	BRIDGEPORT	CT 203/675-1166
1442	NEWKIRK	SANDRA	PRINCETON	NJ 201/812-3331
1417	NEWKIRK	WILLIAM	PATERSON	NJ 201/732-6611
1478	NEWTON	JAMES	NEW YORK	NY 212/587-5549
1673	NICHOLLS	HENRY	STAMFORD	CT 203/781-7770
1839	NORRIS	DIANE	NEW YORK	NY 718/384-1767
1347	O'NEAL	BRYAN	NEW YORK	NY 718/384-0230
1423	OSWALD	LESLIE	MT. VERNON	NY 914/468-9171
1200	OVERMAN	MICHELLE	STAMFORD	CT 203/781-1835
1970	PARKER	ANNE	NEW YORK	NY 718/383-3895
1521	PARKER	JAY	NEW YORK	NY 212/587-7603
1354	PARKER	MARY	WHITE PLAINS	NY 914/455-2337
1424	PATTERSON	RENEE	NEW YORK	NY 212/587-8991
1132	PEARCE	CAROL	NEW YORK	NY 718/384-1986
1845	PEARSON	JAMES	NEW YORK	NY 718/384-2311
1556	PENNINGTON	MICHAEL	NEW YORK	NY 718/383-5681
1413	PETERS	RANDALL	PRINCETON	NJ 201/812-2478
1123	PETERSON	SUZANNE	NEW YORK	NY 718/383-0077
1907	PHELPS	WILLIAM	STAMFORD	CT 203/781-1118
1436	PORTER	SUSAN	NEW YORK	NY 718/383-5777
1385	RAYNOR	MILTON	BRIDGEPORT	CT 203/675-2846
1432	REED	MARILYN	MT. VERNON	NY 914/468-5454
1111	RHODES	JEREMY	PRINCETON	NJ 201/812-1837
1116	RICHARDS	CASEY	NEW YORK	NY 212/587-1224
1352	RIVERS	SIMON	NEW YORK	NY 718/383-3345
1555	RODRIGUEZ	JULIA	BRIDGEPORT	CT 203/675-2401
1038	RODRIGUEZ	MARIA	BRIDGEPORT	CT 203/675-2048
1420	ROUSE	JEREMY	PATERSON	NJ 201/732-9834
1561	SANDERS	RAYMOND	NEW YORK	NY 212/588-6615
1434	SANDERSON	EDITH	STAMFORD	CT 203/781-1333
1414	SANDERSON	NATHAN	BRIDGEPORT	CT 203/675-1715
1112	SANYERS	RANDY	NEW YORK	NY 718/384-4895
1390	SMART	JONATHAN	NEW YORK	NY 718/383-1141
1332	STEPHENSON	ADAM	BRIDGEPORT	CT 203/675-1497
1890	STEPHENSON	ROBERT	NEW YORK	NY 718/384-9874
1429	THOMPSON	ALICE	STAMFORD	CT 203/781-3857
1107	THOMPSON	WAYNE	NEW YORK	NY 718/384-3785
1908	TRENTON	MELISSA	NEW YORK	NY 212/586-6262

```

1830 TRIPP KATHY BRIDGEPORT CT 203/675-2479
1882 TUCKER ALAN NEW YORK NY 718/384-0216
1050 TUTTLE THOMAS WHITE PLAINS NY 914/455-2119
1425 UNDERWOOD JENNY STAMFORD CT 203/781-0978
1928 UPCHURCH LARRY WHITE PLAINS NY 914/455-5009
1480 UPDIKE THERESA NEW YORK NY 212/587-8729
1100 VANDEUSEN RICHARD NEW YORK NY 212/586-2531
1995 VARNER ELIZABETH NEW YORK NY 718/384-7113
1135 VEGA ANNA NEW YORK NY 718/384-5913
1415 VEGA FRANKLIN NEW YORK NY 718/384-2823
1076 VENTER RANDALL NEW YORK NY 718/383-2321
1426 VICK THERESA PRINCETON NJ 201/812-2424
1564 WALTERS ANNE NEW YORK NY 212/587-3257
1221 WALTERS DIANE NEW YORK NY 718/384-1918
1133 WANG CHIN NEW YORK NY 212/587-1956
1435 WARD ELAINE NEW YORK NY 718/383-4987
1418 WATSON BERNARD NEW YORK NY 718/383-1298
1017 WELCH DARIUS NEW YORK NY 212/586-5535
1443 WELLS AGNES STAMFORD CT 203/781-5546
1131 WELLS NADINE NEW YORK NY 718/383-1045
1427 WHALEY CAROLYN MT. VERNON NY 914/468-4528
1036 WONG LESLIE NEW YORK NY 212/587-2570
1130 WOOD DEBORAH NEW YORK NY 212/587-0013
1127 WOOD SANDRA NEW YORK NY 212/587-2881
1433 YANCEY ROBIN PRINCETON NJ 201/812-1874
1431 YOUNG DEBORAH STAMFORD CT 203/781-2987
1122 YOUNG JOANN NEW YORK NY 718/384-2021
1105 YOUNG LAWRENCE NEW YORK NY 718/384-0008
;
run;

```

---

## CONTROL.RESULTS

```

data control.results;
 input ID TREAT $8. INITWT WT3MOS AGE;
 datalines;
 1 Other 166.28 146.98 35
 2 Other 214.42 210.22 54
 3 Other 172.46 159.42 33
 5 Other 175.41 160.66 37
 6 Other 173.13 169.40 20
 7 Other 181.25 170.94 30
 10 Other 239.83 214.48 48
 11 Other 175.32 162.66 51
 12 Other 227.01 211.06 29
 13 Other 274.82 251.82 31
;
run;

```

## CONTROL.SLEEP

```
data control.sleep;
 input GROUP TIME SOL WASO FNA TST;
 datalines;
1.00 1.00 38.69 48.43 0 0
2.36 424.50 0.00 0.00 0 0
1.00 2.00 15.83 9.67 0 0
2.16 500.30 0.00 0.00 0 0
1.00 1.00 93.04 87.10 0 0
1.86 302.40 0.00 0.00 0 0
1.00 2.00 65.00 16.67 0 0
1.50 305.00 0.00 0.00 0 0
1.00 1.00 19.82 74.38 0 0
2.00 359.20 0.00 0.00 0 0
1.00 2.00 13.75 13.90 0 0
1.40 378.13 0.00 0.00 0 0
1.00 1.00 47.35 60.52 0 0
2.89 248.10 0.00 0.00 0 0
1.00 2.00 72.14 86.07 0 0
4.14 308.50 0.00 0.00 0 0
1.00 1.00 99.65 151.79 0 0
3.86 263.93 0.00 0.00 0 0
1.00 2.00 65.35 118.33 0 0
2.71 374.17 0.00 0.00 0 0
1.00 1.00 47.15 105.36 0 0
2.50 254.60 0.00 0.00 0 0
1.00 2.00 66.00 92.60 0 0
2.20 297.40 0.00 0.00 0 0
1.00 1.00 30.84 84.97 0 0
3.24 242.90 0.00 0.00 0 0
1.00 2.00 7.86 23.86 0 0
1.28 340.70 0.00 0.00 0 0
1.00 1.00 117.41 36.93 0 0
1.00 204.20 0.00 0.00 0 0
1.00 2.00 50.42 24.86 0 0
1.85 231.00 0.00 0.00 0 0
1.00 1.00 6.50 41.15 0 0
2.67 308.00 0.00 0.00 0 0
1.00 2.00 2.00 0.00 0 0
0.00 454.40 0.00 0.00 0 0
2.00 1.00 54.29 84.93 0 0
3.43 394.70 0.00 0.00 0 0
2.00 2.00 13.57 48.00 0 0
1.00 405.00 0.00 0.00 0 0
2.00 1.00 4.43 58.43 0 0
2.89 375.70 0.00 0.00 0 0
2.00 2.00 5.00 49.28 0 0
3.57 423.60 0.00 0.00 0 0
2.00 1.00 32.50 38.43 0 0
4.57 357.20 0.00 0.00 0 0
2.00 2.00 17.14 33.14 0 0
```



3.57	315.70	0.00	0.00	0	0
2.00	1.00	33.57	83.43	0	0
4.36	282.50	0.00	0.00	0	0
2.00	2.00	22.85	37.86	0	0
2.42	288.57	0.00	0.00	0	0
2.00	1.00	53.21	120.00	0	0
3.50	366.80	0.00	0.00	0	0
2.00	2.00	29.00	97.40	0	0
2.80	388.00	0.00	0.00	0	0
2.00	1.00	56.79	67.73	0	0
3.19	326.00	0.00	0.00	0	0
2.00	2.00	52.50	64.16	0	0
4.00	45.80	0.00	0.00	0	0
2.00	1.00	23.50	35.19	0	0
6.60	416.80	0.00	0.00	0	0
2.00	2.00	25.71	38.43	0	0
8.50	446.30	0.00	0.00	0	0
2.00	1.00	43.00	95.59	0	0
1.75	288.90	0.00	0.00	0	0
2.00	2.00	45.00	85.71	0	0
1.43	272.10	0.00	0.00	0	0
3.00	1.00	24.70	67.17	0	0
3.90	399.90	0.00	0.00	0	0
3.00	2.00	3.86	22.50	0	0
3.00	425.14	0.00	0.00	0	0
3.00	1.00	70.72	80.43	0	0
3.00	286.30	0.00	0.00	0	0
3.00	2.00	22.50	139.33	0	0
3.50	283.50	0.00	0.00	0	0
3.00	1.00	48.23	40.57	0	0
1.65	407.20	0.00	0.00	0	0
3.00	2.00	30.00	38.00	0	0
1.57	372.90	0.00	0.00	0	0
3.00	1.00	43.93	34.57	0	0
1.29	393.40	0.00	0.00	0	0
3.00	2.00	30.71	58.43	0	0
2.28	348.60	0.00	0.00	0	0
3.00	1.00	95.00	35.22	0	0
2.06	409.90	0.00	0.00	0	0
3.00	2.00	66.43	68.57	0	0
2.14	345.60	0.00	0.00	0	0
3.00	1.00	18.93	86.43	0	0
5.36	349.50	0.00	0.00	0	0
3.00	2.00	17.50	116.50	0	0
5.00	337.70	0.00	0.00	0	0
3.00	1.00	125.00	69.15	0	0
2.00	242.50	0.00	0.00	0	0
3.00	2.00	60.00	81.43	0	0
1.57	304.30	0.00	0.00	0	0
3.00	1.00	38.57	68.61	0	0
3.64	394.50	0.00	0.00	0	0
3.00	2.00	18.33	19.83	0	0
2.00	448.83	0.00	0.00	0	0
3.00	1.00	43.00	121.72	0	0
2.63	247.80	0.00	0.00	0	0
3.00	2.00	40.00	98.83	0	0

```

1.86 199.67 0.00 0.00 0 0
3.00 1.00 11.61 70.36 0 0
2.86 348.40 0.00 0.00 0 0
3.00 2.00 20.43 70.57 0 0
3.14 335.40 0.00 0.00 0 0
;
run;

```

## CONTROL.SYNDROME

```

data control.syndrome;
 input FLIGHT $ 1-3 @10 DATE DATE7. @22 DEPART TIME5. ORIG $ 31-33
 DEST $ 39-41 MILES BOARDED CAPACITY;
 format date DATE7.;
 format depart TIME5.;
 informat date DATE7.;
 informat depart TIME5.;
 datalines;
114 01MAR18 7:10 LGA LAX 2475 172 210
202 01MAR18 10:43 LGA ORD 740 151 210
219 01MAR18 9:31 LGA LON 3442 198 250
622 01MAR18 12:19 LGA FRA 3857 207 250
132 01MAR18 15:35 LGA YYZ 366 115 178
271 01MAR18 13:17 LGA PAR 3635 138 250
302 01MAR18 20:22 LGA WAS 229 105 180
114 02MAR18 7:10 LGA LAX 2475 119 210
202 02MAR18 10:43 LGA ORD 740 120 210
219 02MAR18 9:31 LGA LON 3442 147 250
622 02MAR18 12:19 LGA FRA 3857 176 250
132 02MAR18 15:35 LGA YYZ 366 106 178
302 02MAR18 20:22 LGA WAS 229 78 180
271 02MAR18 13:17 LGA PAR 3635 104 250
114 03MAR18 7:10 LGA LAX 2475 197 210
202 03MAR18 10:43 LGA ORD 740 118 210
219 03MAR18 9:31 LGA LON 3442 197 250
622 03MAR18 12:19 LGA FRA 3857 180 250
132 03MAR18 15:35 LGA YYZ 366 75 178
271 03MAR18 13:17 LGA PAR 3635 147 250
302 03MAR18 20:22 LGA WAS 229 123 180
114 04MAR18 7:10 LGA LAX 2475 178 210
202 04MAR18 10:43 LGA ORD 740 148 210
219 04MAR18 9:31 LGA LON 3442 232 250
622 04MAR18 12:19 LGA FRA 3857 137 250
132 04MAR18 15:35 LGA YYZ 366 117 178
271 04MAR18 13:17 LGA PAR 3635 146 250
302 04MAR18 20:22 LGA WAS 229 115 180
114 05MAR18 7:10 LGA LAX 2475 117 210
202 05MAR18 10:43 LGA ORD 740 104 210
219 05MAR18 9:31 LGA LON 3442 160 250
622 05MAR18 12:19 LGA FRA 3857 185 250
132 05MAR18 15:35 LGA YYZ 366 157 178
271 05MAR18 13:17 LGA PAR 3635 177 250
114 06MAR18 7:10 LGA LAX 2475 128 210

```

```

202 06MAR18 10:43 LGA ORD 740 115 210
219 06MAR18 9:31 LGA LON 3442 163 250
132 06MAR18 15:35 LGA YYZ 366 150 178
302 06MAR18 20:22 LGA WAS 229 66 180
114 07MAR18 7:10 LGA LAX 2475 160 210
202 07MAR18 10:43 LGA ORD 740 175 210
219 07MAR18 9:31 LGA LON 3442 241 250
622 07MAR18 12:19 LGA FRA 3857 210 250
132 07MAR18 15:35 LGA YYZ 366 164 178
271 07MAR18 13:17 LGA PAR 3635 155 250
302 07MAR18 20:22 LGA WAS 229 135 180
;
run;

```

---

## CONTROL.TENSION

```

data control.tension;
 input TENSION $8. CHD $8. COUNT ;
 datalines;
yes yes 97
yes no 307
no yes 200
no no 1409
;
run;

```

---

## CONTROL.TEST2

```

data control.test2;
 input STD1 $ TEST1 $ STD2 $ TEST2 $ WT;
 datalines ;
neg neg neg neg 509
neg neg neg pos 4
neg neg pos neg 17
neg neg pos pos 3
neg pos neg neg 13
neg pos neg pos 8
neg pos pos pos 8
pos neg neg neg 14
pos neg neg pos 1
pos neg pos neg 17
pos neg pos pos 9
pos pos neg neg 7
pos pos neg pos 4
pos pos pos neg 9
pos pos pos pos 170
;
run;

```

---

## CONTROL.TRAIN

```
data control.train;
 input NAME $ 1-16 IDNUM $ 17-24;
 datalines;
Capalleti, Jimmy 2355
Chen, Len 5889
Davis, Brad 3878
Leung, Brenda 4409
Patel, Mary 2398
Smith, Robert 5862
Zook, Carla 7385
;
run;
```

---

## CONTROL.VISION

```
data control.vision;
 input RIGHT LEFT COUNT;
 datalines;
1 1 1520
1 2 266
1 3 124
1 4 66
2 1 234
2 2 1512
2 3 432
2 4 78
3 1 117
3 2 362
3 3 1772
3 4 205
4 1 36
4 2 82
4 3 179
4 4 492
;
run;
```

---

## CONTROL.WEIGHT

```
data control.weight (label='California Results');
 input ID TREAT $8. IBW INITWT WT3MOS WT6MOS WT9MOS AGE MMPI1 MMPI2
```

MMPI3 MMPI4 MMPI5;												
datalines;												
1	Other	149	166.28	146.98	138.26	.	35	62	68	67	55	67
2	Other	137	214.42	210.22	.	213.87	54	57	56	59	57	47
3	Other	138	172.46	159.42	146.01	143.84	33	54	69	63	87	34
5	Other	122	175.41	160.66	154.30	.	37	56	67	64	71	32
6	Other	134	173.13	169.40	176.12	.	20	42	51	63	71	45
7	Other	160	181.25	170.94	.	.	30	72	58	70	71	80
9	Other	152	212.83	179.93	169.74	164.47	49100	65	87	71	74	4
10	Other	145	239.83	214.48	208.28	.	48	56	51	56	53	53
11	Other	158	175.32	162.66	161.39	.	51	66	60	71	62	84
12	Other	174	227.01	211.06	202.87	205.17	29	77	70	69	65	64
13	Other	137	274.82	251.82	248.18	.	31	66	82	66	69	55
15	Other	152	168.75	156.58	154.61	156.58	42	52	58	65	67	51
16	Other	162	187.81	172.07	.	.	40	57	68	67	67	74
17	Other	123	226.63	.	219.72	.	21	58	49	59	74	70
18	Other	146	176.03	160.27	160.27	.	41	48	55	49	68	43
19	Other	166	190.96	159.04	.	.	32	53	52	51	62	71
21	Other	148	165.54	.	166.22	.	48	57	60	65	74	55
22	Other	125	193.60	184.00	.	.	28	64	67	70	69	54
24	Other	152	267.43	230.26	206.09	.	30	48	45	55	50	41
25	Other	151	193.38	185.43	.	.	33	54	99	67	75	74
26	Other	134	252.61	227.61	217.72	223.88	31	62	51	70	57	39
27	Other	140	193.93	191.43	196.43	.	25	54	65	66	55	55
29	Other	119	182.77	.	.	.	38	66	63	64	60	41
30	Other	134	189.93	172.39	175.37	.	35	70	92	73	98	39
31	Other	138	190.22	181.88	178.99	181.16	36	56	69	61	62	34
32	Other	134	182.09	169.40	163.81	163.81	34	42	65	54	55	41
34	Other	133	200.00	189.47	.	.	24	52	61	64	62	39
35	Other	149	221.81	216.11	.	.	46	66	52	69	71	61
36	Other	133	241.35	247.37	.	.	34	50	65	57	74	47
37	Other	134	223.13	217.91	.	.	33	64	63	63	60	41
38	Other	140	235.36	228.57	210.71	.	50	56	61	70	74	39
39	Other	125	178.60	178.40	.	.	39	50	73	58	58	32
40	Other	143	243.01	226.57	210.49	.	38	64	66	70	54	46
41	Other	134	282.65	239.55	.	.	26	66	65	75	74	53
44	Other	139	282.37	258.99	238.13	241.01	43	66	69	68	65	39
45	Other	134	216.04	182.09	.	.	39	50	55	59	67	43
46	Other	115	190.00	171.30	.	.	36	64	59	58	58	44
47	Other	134	175.19	167.16	.	.	43	57	48	52	71	47
48	Other	118	179.87	.	.	.	37	52	57	45	50	30
50	Other	137	173.54	166.97	164.60	.	32	54	76	63	69	25
51	Other	125	180.60	162.40	152.00	157.60	32	50	56	64	53	53
52	Other	155	235.32	225.16	210.32	208.39	35	62	64	55	60	51
53	Other	143	183.39	169.23	.	.	38	64	57	72	81	61
54	Other	131	212.60	208.40	211.45	.	27	50	67	53	74	47
63	Surgery	146	219.18	167.12	139.73	119.18	35	58	53	67	48	55
65	Surgery	123	192.68	155.28	127.64	115.45	31	52	74	54	64	32
66	Surgery	134	199.25	173.88	161.19	144.03	38	68	64	70	77	30
71	Surgery	139	209.35	172.66	156.83	138.13	39	66	56	66	71	42
77	Surgery	137	179.56	150.36	132.12	123.36	29	46	49	42	47	49
80	Surgery	170	138.24	121.76	100.00	.	31	56	57	64	64	39
81	Surgery	129	206.98	173.64	132.56	121.71	28	42	78	52	62	41
84	Surgery	143	220.28	178.32	.	.	29	58	67	54	46	59
85	Surgery	152	189.47	156.58	140.79	140.79	34	75	73	62	74	51
86	Surgery	210	157.14	138.57	121.90	109.05	30	88	75	73	69	65

```

92 Surgery 139 203.60 169.78 143.88 . 38 62 59 68 60 49
93 Surgery 151 171.52 150.33 123.18 109.27 42 72 69 59 53 47
95 Surgery 134 207.46 155.22 . . 41 51 52 56 63 57
96 Surgery 138 201.45 172.46 172.46 155.07 55 57 49 57 67 37
97 Surgery 128 209.38 182.81 162.50 153.91 34 68 88 73 74 .
101 Surgery 166 185.54 146.39 139.76 132.53 42 82 80 89 79 51
102 Surgery 127 218.11 173.23 152.76 137.80 39 76 80 70 74 45
107 Surgery 128 227.34 192.97 184.38 170.31 49 50 51 59 50 55
110 Surgery 143 251.75 207.69 183.22 165.03 42 48 84 52 76 38
111 Surgery 152 197.37 164.47 148.68 132.89 56 90 70 86 76 70
112 Surgery 140 202.14 156.43 137.86 . 31 48 51 50 41 41
116 Surgery 139 273.38 235.25 194.24 . 31 58 55 66 81 45
117 Surgery 125 192.00 156.80 140.00 127.20 42 58 44 63 53 51
120 Surgery 119 277.31 231.93 208.40 192.44 31 60 69 59 95 34
122 Surgery 138 165.22 130.43 119.57 . 44 66 67 70 62 34
125 Surgery 152 257.89 200.00 182.89 134.21 39 47 84 53 88 74
126 Surgery 138 170.29 142.75 . . 39 64 64 66 65 46
132 Surgery 149 208.05 173.83 . 126.85 34 82 57 75 65 45
133 Surgery 136 230.15 205.15 190.44 180.88 39 52 71 52 65 30
134 Surgery 168 177.98 140.48 119.64 . 45 70 77 62 67 69
137 Surgery 138 188.41 164.49 152.90 135.51 39 58 61 57 60 46
138 Surgery 141 268.09 220.57 185.82 . 32 52 59 66 74 53
158 Surgery 130 220.00 190.77 173.85 . 27 68 73 66 58 43
223 Surgery 157 220.38 185.35 152.23 138.85 43 78 76 70 57 53
266 Surgery 135 184.44 148.15 123.70 . 41 56 55 59 64 49
269 Surgery 155 201.29 151.61 140.65 . 57 82 84 86 77 73
293 Surgery 160 163.75 130.63 . . 47 52 67 47 64 76
295 Surgery 128 211.72 172.66 151.56 137.50 45 93 65 80 60 41
298 Surgery 140 243.57 195.00 . 166.43 40 63 62 62 76 52
;
run;
```

## CONTROL.WGHT

```

data control.wght (label='California Results');
 input ID TREAT $8. IBW INITWT WT3MOS WT6MOS WT9MOS AGE MMPI1 MMPI2
 MMPI3 MMPI4 MMPI5;
 datalines;
1 Other 149 166.28 146.98 138.26 . 35 62 68 67 55 67
2 Other 137 214.42 210.22 . 213.87 54 57 56 59 57 47
3 Other 138 172.46 159.42 146.01 143.84 33 54 69 63 87 34
5 Other 122 175.41 160.66 154.30 . 37 56 67 64 71 32
6 Other 134 173.13 169.40 176.12 . 20 42 51 63 71 45
7 Other 160 181.25 170.94 . . 30 72 58 70 71 80
9 Other 152 212.83 179.93 169.74 164.47 49 65 87 71 74 4
10 Other 145 239.83 214.48 208.28 . 48 56 51 56 53 53
11 Other 158 175.32 162.66 161.39 . 51 66 60 71 62 84
12 Other 174 227.01 211.06 202.87 205.17 29 77 70 69 65 64
13 Other 137 274.82 251.82 248.18 . 31 66 82 66 69 55
15 Other 152 168.75 156.58 154.61 156.58 42 52 58 65 67 51
16 Other 162 187.81 172.07 . . 40 57 68 67 67 74
17 Other 123 226.63 . 219.72 . 21 58 49 59 74 70
18 Other 146 176.03 160.27 160.27 . 41 48 55 49 68 43
```

19	Other	166	190.96	159.04	.	.	32	53	52	51	62	71
21	Other	148	165.54	.	166.22	.	48	57	60	65	74	55
22	Other	125	193.60	184.00	.	.	28	64	67	70	69	54
24	Other	152	267.43	230.26	206.09	.	30	48	45	55	50	41
25	Other	151	193.38	185.43	.	.	33	54	99	67	75	74
26	Other	134	252.61	227.61	217.72	223.88	31	62	51	70	57	39
27	Other	140	193.93	191.43	196.43	.	25	54	65	66	55	55
29	Other	119	182.77	.	.	.	38	66	63	64	60	41
30	Other	134	189.93	172.39	175.37	.	35	70	92	73	98	39
31	Other	138	190.22	181.88	178.99	181.16	36	56	69	61	62	34
32	Other	134	182.09	169.40	163.81	163.81	34	42	65	54	55	41
34	Other	133	200.00	189.47	.	.	24	52	61	64	62	39
35	Other	149	221.81	216.11	.	.	46	66	52	69	71	61
36	Other	133	241.35	247.37	.	.	34	50	65	57	74	47
37	Other	134	223.13	217.91	.	.	33	64	63	63	60	41
38	Other	140	235.36	228.57	210.71	.	50	56	61	70	74	39
39	Other	125	178.60	178.40	.	.	39	50	73	58	58	32
40	Other	143	243.01	226.57	210.49	.	38	64	66	70	54	46
41	Other	134	282.65	239.55	.	.	26	66	65	75	74	53
44	Other	139	282.37	258.99	238.13	241.01	43	66	69	68	65	39
45	Other	134	216.04	182.09	.	.	39	50	55	59	67	43
46	Other	115	190.00	171.30	.	.	36	64	59	58	58	44
47	Other	134	175.19	167.16	.	.	43	57	48	52	71	47
48	Other	118	179.87	.	.	.	37	52	57	45	50	30
50	Other	137	173.54	166.97	164.60	.	32	54	76	63	69	25
51	Other	125	180.60	162.40	152.00	157.60	32	50	56	64	53	53
52	Other	155	235.32	225.16	210.32	208.39	35	62	64	55	60	51
53	Other	143	183.39	169.23	.	.	38	64	57	72	81	61
54	Other	131	212.60	208.40	211.45	.	27	50	67	53	74	47
63	Surgery	146	219.18	167.12	139.73	119.18	35	58	53	67	48	55
65	Surgery	123	192.68	155.28	127.64	115.45	31	52	74	54	64	32
66	Surgery	134	199.25	173.88	161.19	144.03	38	68	64	70	77	30
71	Surgery	139	209.35	172.66	156.83	138.13	39	66	56	66	71	42
77	Surgery	137	179.56	150.36	132.12	123.36	29	46	49	42	47	49
80	Surgery	170	138.24	121.76	100.00	.	31	56	57	64	64	39
81	Surgery	129	206.98	173.64	132.56	121.71	28	42	78	52	62	41
84	Surgery	143	220.28	178.32	.	.	29	58	67	54	46	59
85	Surgery	152	189.47	156.58	140.79	140.79	34	75	73	62	74	51
86	Surgery	210	157.14	138.57	121.90	109.05	30	88	75	73	69	65
92	Surgery	139	203.60	169.78	143.88	.	38	62	59	68	60	49
93	Surgery	151	171.52	150.33	123.18	109.27	42	72	69	59	53	47
95	Surgery	134	207.46	155.22	.	.	41	51	52	56	63	57
96	Surgery	138	201.45	172.46	172.46	155.07	55	57	49	57	67	37
97	Surgery	128	209.38	182.81	162.50	153.91	34	68	88	73	74	.
101	Surgery	166	185.54	146.39	139.76	132.53	42	82	80	89	79	51
102	Surgery	127	218.11	173.23	152.76	137.80	39	76	80	70	74	45
107	Surgery	128	227.34	192.97	184.38	170.31	49	50	51	59	50	55
110	Surgery	143	251.75	207.69	183.22	165.03	42	48	84	52	76	38
111	Surgery	152	197.37	164.47	148.68	132.89	56	90	70	86	76	70
112	Surgery	140	202.14	156.43	137.86	.	31	48	51	50	41	41
116	Surgery	139	273.38	235.25	194.24	.	31	58	55	66	81	45
117	Surgery	125	192.00	156.80	140.00	127.20	42	58	44	63	53	51
120	Surgery	119	277.31	231.93	208.40	192.44	31	60	69	59	95	34
122	Surgery	138	165.22	130.43	119.57	.	44	66	67	70	62	34
125	Surgery	152	257.89	200.00	182.89	134.21	39	47	84	53	88	74
126	Surgery	138	170.29	142.75	.	.	39	64	64	66	65	46

```

132 Surgery 149 208.05 173.83 . 126.85 34 82 57 75 65 45
133 Surgery 136 230.15 205.15 190.44 180.88 39 52 71 52 65 30
134 Surgery 168 177.98 140.48 119.64 . 45 70 77 62 67 69
137 Surgery 138 188.41 164.49 152.90 135.51 39 58 61 57 60 46
138 Surgery 141 268.09 220.57 185.82 . 32 52 59 66 74 53
158 Surgery 130 220.00 190.77 173.85 . 27 68 73 66 58 43
223 Surgery 157 220.38 185.35 152.23 138.85 43 78 76 70 57 53
266 Surgery 135 184.44 148.15 123.70 . 41 56 55 59 64 49
269 Surgery 155 201.29 151.61 140.65 . 57 82 84 86 77 73
293 Surgery 160 163.75 130.63 . . 47 52 67 47 64 76
295 Surgery 128 211.72 172.66 151.56 137.50 45 93 65 80 60 41
298 Surgery 140 243.57 195.00 . 166.43 40 63 62 62 76 52
;
run;

```

## CUSTOMER\_RESPONSE

```

data customer_response;
 input Customer Factor1-Factor4 Source1-Source3
 Quality1-Quality3;
 datalines;
1 . . 1 1 1 1 . 1 . .
2 1 1 . 1 1 1 . 1 1 .
3 . . 1 1 1 1
4 1 1 . 1 . 1 . . . 1
5 . 1 . 1 1 1
6 . 1 . 1 1
7 . 1 . 1 1 . . 1 . .
8 1 . . 1 1 1 . 1 1 .
9 1 1 . 1 1 1
10 1 . . 1 1 1 . 1 1 .
11 1 1 1 1 . 1 . 1 1 1
12 1 1 . 1 1 1
13 1 1 . 1 . 1 . 1 1 .
14 1 1 . 1 1 1
15 1 1 . 1 . 1 . 1 1 1
16 1 . . 1 1 . . 1 . .
17 1 1 . 1 1 1 . . 1 .
18 1 1 . 1 1 1 1 . . 1
19 . 1 . 1 1 1 1 . 1 .
20 1 . . 1 1 1 . 1 1 1
21 . . . 1 1 1 . 1 . .
22 . . . 1 1 1 . 1 1 .
23 1 . . 1 1
24 . 1 . 1 1 . . 1 . 1
25 1 1 . 1 1 . . . 1 1
26 1 1 . 1 1 . . 1 . .
27 1 . . 1 1 . . . 1 .
28 1 1 . 1 . . . 1 1 1
29 1 . . 1 1 1 . 1 . 1
30 1 . 1 1 1 . . 1 1 .

```



```

31 . . . 1 1 . . 1 1 .
32 1 1 1 1 1 . . 1 1 1
33 1 . . 1 1 . . 1 . 1
34 . . 1 1 . . . 1 1 .
35 1 1 1 1 1 . 1 1 . .
36 1 1 1 1 . 1 . 1 . .
37 1 1 . 1 . . . 1 . .
38 . . . 1 1 1 . 1 . .
39 1 1 . 1 1 . . 1 . 1
40 1 . . 1 . . 1 1 . 1
41 1 . . 1 1 1 1 1 . 1
42 1 1 1 1 . . 1 1 . .
43 1 . . 1 1 1 . 1 . .
44 1 . 1 1 . 1 . 1 . 1
45 . . . 1 . . 1 . . 1
46 . . . 1 1 . . . 1 .
47 1 1 . 1 . . 1 1 . .
48 1 . 1 1 1 . 1 1 . .
49 . . 1 1 1 1 . 1 . 1
50 . 1 . 1 1 . . 1 1 .
51 1 . 1 1 1 1
52 1 1 1 1 1 1 . 1 . .
53 . 1 1 1 . 1 . 1 1 1
54 1 . . 1 1 . . 1 1 .
55 1 1 . 1 1 1 . 1 . .
56 1 . . 1 1 . . 1 1 .
57 1 1 . 1 1 . 1 . . 1
58 . 1 . 1 . 1 . . 1 1
59 1 1 1 1 . . 1 1 1 .
60 . 1 1 1 1 1 . . 1 1
61 1 1 1 1 1 1 . 1 . .
62 1 1 . 1 1 . . 1 1 .
63 . . . 1 . . . 1 1 1
64 1 . . 1 1 1 . 1 . .
65 1 . . 1 1 1 . 1 . .
66 1 . . 1 1 1 1 1 1 .
67 1 1 . 1 1 1 . 1 1 .
68 1 1 . 1 1 1 . 1 1 .
69 1 1 . 1 1 . 1 . . .
70 . . . 1 1 1 . 1 . .
71 1 . . 1 1 . 1 . . 1
72 1 . 1 1 1 1 . . 1 .
73 1 1 . 1 . 1 . 1 1 .
74 1 1 1 1 1 1 . 1 . .
75 . 1 . 1 1 1 . . 1 .
76 1 1 . 1 1 1 . 1 1 1
77 . . . 1 1 1
78 1 1 1 1 1 1 . 1 1 .
79 1 . . 1 1 1 . 1 1 .
80 1 1 1 1 1 . 1 1 . 1
81 1 1 . 1 1 1 1 1 1 .
82 . . . 1 1 1 1 . . .
83 1 1 . 1 1 1 . 1 1 .
84 1 . . 1 1 . . 1 1 .
85 . . . 1 . 1 . 1 . .
86 1 . . 1 1 1 . 1 1 1

```

```

87 1 1 . 1 1 1 . 1 . .
88 . . . 1 . 1
89 1 . . 1 . 1 . . 1 1
90 1 1 . 1 1 1 . 1 . 1
91 . . . 1 1 . . . 1 .
92 1 . . 1 1 1 . 1 1 .
93 1 . . 1 1 . . 1 1 .
94 1 . . 1 1 1 1 1 . .
95 1 . . 1 . 1 1 1 1 .
96 1 . 1 1 1 1 . . 1 .
97 1 1 . 1 1 . . . 1 .
98 1 . 1 1 1 1 1 1 . .
99 1 1 . 1 1 1 1 1 1 .
100 1 . 1 1 1 . . . 1 1
101 1 . 1 1 1 1
102 1 . . 1 1 . 1 1 . .
103 1 1 . 1 1 1 . 1 . .
104 . . . 1 1 1 . 1 1 1
105 1 . 1 1 1 . . 1 . 1
106 1 1 1 1 1 1 1 1 1 1
107 1 1 1 1 . . . 1 . 1
108 1 . . 1 . 1 1 1 . .
109 . 1 . 1 1 . . 1 1 .
110 1 . . 1
111 1 . . 1 1 1 . 1 1 .
112 1 1 . 1 1 1 . . . 1
113 1 1 . 1 1 . 1 1 1 .
114 1 1 . 1 1
115 1 1 . 1 1 . . 1 . .
116 . 1 . 1 1 1 1 1 . .
117 . 1 . 1 1 1
118 . 1 1 1 1 . . 1 1 .
119 . . . 1 . . . 1 . .
120 1 1 . 1 1 .
;

```

---

## DJIA

```

data djia;
 input Year HighDate date9. High LowDate date9. Low;
 format highdate lowdate date9.;
 datalines;
1968 03DEC1968 985.21 21MAR1968 825.13
1969 14MAY1969 968.85 17DEC1969 769.93
1970 29DEC1970 842.00 06MAY1970 631.16
1971 28APR1971 950.82 23NOV1971 797.97
1972 11DEC1972 1036.27 26JAN1972 889.15
1973 11JAN1973 1051.70 05DEC1973 788.31
1974 13MAR1974 891.66 06DEC1974 577.60
1975 15JUL1975 881.81 02JAN1975 632.04
1976 21SEP1976 1014.79 02JAN1976 858.71

```

```

1977 03JAN1977 999.75 02NOV1977 800.85
1978 08SEP1978 907.74 28FEB1978 742.12
1979 05OCT1979 897.61 07NOV1979 796.67
1980 20NOV1980 1000.17 21APR1980 759.13
1981 27APR1981 1024.05 25SEP1981 824.01
1982 27DEC1982 1070.55 12AUG1982 776.92
1983 29NOV1983 1287.20 03JAN1983 1027.04
1984 06JAN1984 1286.64 24JUL1984 1086.57
1985 16DEC1985 1553.10 04JAN1985 1184.96
1986 02DEC1986 1955.57 22JAN1986 1502.29
1987 25AUG1987 2722.42 19OCT1987 1738.74
1988 21OCT1988 2183.50 20JAN1988 1879.14
1989 09OCT1989 2791.41 03JAN1989 2144.64
1990 16JUL1990 2999.75 11OCT1990 2365.10
1991 31DEC1991 3168.83 09JAN1991 2470.30
1992 01JUN1992 3413.21 09OCT1992 3136.58
1993 29DEC1993 3794.33 20JAN1993 3241.95
1994 31JAN1994 3978.36 04APR1994 3593.35
1995 13DEC1995 5216.47 30JAN1995 3832.08
1996 27DEC1996 6560.90 10JAN1996 5032.94
1997 06AUG1997 8259.30 11APR1997 6391.69
1998 23NOV1998 9374.27 31AUG1998 7539.06
1999 31DEC1999 11497.12 22JAN1999 9120.67
2000 14JAN2000 11722.98 07MAR2000 9796.04
2001 21MAY2001 11337.92 21SEP2001 8235.81
2002 19MAR2002 10635.25 09OCT2002 7286.27
2003 31DEC2003 10453.92 11MAR2003 7524.06
2004 28DEC2004 10854.54 25OCT2004 9749.99
2005 04MAR2005 10940.55 20APR2005 10012.36
2006 27DEC2006 12510.57 20JAN2006 10667.39
2007 09OCT2007 14164.53 05MAR2007 12050.41
2008 02MAY2008 13058.20 10OCT2008 8451.19
2009 24DEC2009 10520.10 09MAR2009 6507.04
2010 31DEC2010 11577.51 02JUL2010 9686.48
2011 29APR2011 12810.54 23SEP2011 10771.48
2012 05OCT2012 13610.15 01JUN2012 12118.57
2013 31DEC2013 16576.66 04JAN2013 13435.21
2014 05DEC2014 17958.79 11APR2014 16026.75
2015 15MAY2015 18272.56 04SEP2015 16102.38
2016 23DEC2016 19993.81 12FEB2015 15973.84
2017 22DEC2017 24754.06 06JAN2017 19963.80

```

```
;
```

---

## EDUCATION

```

data education;
 input State $14. +1 Code $ DropoutRate Expenditures MathScore Region $;
 label dropoutrate='Dropout Percentage - 2017'
 expenditures='Expenditure Per Pupil - 2017'
 mathscore='8th Grade Math Exam - 2018';
 datalines;

```

```

Alabama AL 22.3 3197 252 SE
Alaska AK 35.8 7716 . W
Arizona AZ 31.2 3902 259 W
Arkansas AR 11.5 3273 256 SE
California CA 32.7 4121 256 W
Colorado CO 24.7 4408 267 W
Connecticut CT 16.8 6857 270 NE
Delaware DE 28.5 5422 261 NE
Florida FL 38.5 4563 255 SE
Georgia GA 27.9 3852 258 SE
Hawaii HI 18.3 4121 251 W
Idaho ID 21.8 2838 272 W
Illinois IL 21.5 4906 260 MW
Indiana IN 13.8 4284 267 MW
Iowa IA 13.6 4285 278 MW
Kansas KS 17.9 4443 . MW
Kentucky KY 32.7 3347 256 SE
Louisiana LA 43.1 3317 246 SE
Maine ME 22.5 4744 . NE
Maryland MD 26.0 5758 260 NE
Massachusetts MA 28.0 5979 . NE
Michigan MI 29.3 5116 264 MW
Minnesota MN 11.4 4755 276 MW
Mississippi MS 39.9 2874 . SE
Missouri MO 26.5 4263 . MW
Montana MT 15.0 4293 280 W
Nebraska NE 13.9 4360 276 MW
Nevada NV 28.1 3791 . W
New Hampshire NH 25.9 4807 273 NE
New Jersey NE 20.4 7549 269 NE
New Mexico NM 28.5 3473 256 W
New York NY 35.0 . 261 NE
North Carolina NC 31.2 3874 250 SE
North Dakota ND 12.1 3952 281 MW
Ohio OH 24.4 4649 264 MW
;

```

---

## EMPDATA

```

data empdata;
input IdNumber $ 1-4 LastName $ 8-18 FirstName $ 19-28
 City $ 29-41 State $ 42-43
 Gender $ 45 JobCode $ 49-51 Salary 55-60 @63 Birth date9.
 @73 Hired date9. HomePhone $ 85-98;
format birth hired date9.;
datalines;
1919 Adams Gerald Stamford CT M TA2 34376 12SEP1990 04JUN2007
203/781-1255
1653 Ahmad Azeem Bridgeport CT F ME2 35108 15OCT1984 09AUG2010
203/675-7715

```

1400	Alvarez	Gloria	New York	NY M	ME1	29769	05NOV1987	16OCT2010
212/586-0808								
1350	Arthur	Barbara	New York	NY F	FA3	32886	31AUG1985	29JUL2010
718/383-1549								
1401	Avery	Jerry	Paterson	NJ M	TA3	38822	13DEC1970	17NOV2005
201/732-8787								
1499	Barefoot	Joseph	Princeton	NJ M	ME3	43025	26APR1974	07JUN2010
201/812-5665								
1101	Basquez	Richardo	New York	NY M	SCP	18723	06JUN1982	01OCT2010
212/586-8060								
1333	Beaulieu	Armando	Stamford	CT M	PT2	88606	30MAR1983	10FEB2011
203/781-1777								
1402	Blalock	Ralph	New York	NY M	TA2	32615	17JAN1983	02DEC2010
718/384-2849								
1479	Bostic	Marie	New York	NY F	TA3	38785	22DEC1988	05OCT2007
718/384-8816								
1403	Bowden	Earl	Bridgeport	CT M	ME1	28072	28JAN1989	21DEC2016
203/675-3434								
1739	Boyce	Jonathan	New York	NY M	PT1	66517	25DEC1984	27JAN2011
212/587-1247								
1658	Bradley	Jeremy	New York	NY M	SCP	17943	08APR1987	29FEB2012
212/587-3622								
1428	Brady	Christine	Stamford	CT F	PT1	68767	04APR1980	16NOV2011
203/781-1212								
1782	Brown	Jason	Stamford	CT M	ME2	35345	04DEC1980	22FEB2012
203/781-0019								
1244	Bryant	Leonard	New York	NY M	ME2	36925	31AUG1983	17JAN2008
718/383-3334								
1383	Burnette	Thomas	New York	NY M	BCK	25823	25JAN1985	20OCT2009
718/384-3569								
1574	Cahill	Marshall	New York	NY M	FA2	28572	27APR1980	20DEC2012
718/383-2338								
1789	Canales	Viviana	New York	NY M	SCP	18326	23JAN1977	11APR2008
212/587-9000								
1404	Carter	Donald	New York	NY M	PT2	91376	24FEB1973	01JAN2010
718/384-2946								
1437	Carter	Dorothy	Bridgeport	CT F	A3	33104	20SEP1980	31AUG2004
203/675-4117								
1639	Carter	Karen	Stamford	CT F	A3	40260	26JUN1977	28JAN2014
203/781-8839								
1269	Caston	Franklin	Stamford	CT M	NA1	41690	03MAY1972	28NOV2012
203/781-3335								
1065	Chapman	Neil	New York	NY M	ME2	35090	26JAN1977	07JAN2007
718/384-5618								
1876	Chin	Jack	New York	NY M	TA3	39675	20MAY1978	27APR2015
212/588-5634								
1037	Chow	Jane	Stamford	CT F	TA1	28558	10APR1984	13SEP2012
203/781-8868								
1129	Cook	Brenda	New York	NY F	ME2	34929	08DEC1981	17AUG2011
718/383-2313								
1988	Cooper	Anthony	New York	NY M	FA3	32217	30NOV1979	18SEP2004
212/587-1228								
1405	Davidson	Jason	Paterson	NJ M	SCP	18056	05MAR1986	26JAN2012
201/732-2323								
1430	Dean	Sandra	Bridgeport	CT F	TA2	32925	28FEB1982	27APR2017
203/675-1647								

```

1983 Dean Sharon New York NY F FA3 33419 28FEB1962 30APR2005
718/384-1647
1134 Delgado Maria Stamford CT F TA2 33462 05MAR1989 21DEC2016
203/781-1528
1118 Dennis Roger New York NY M PT3 111379 16JAN1964 18DEC2000
718/383-1122
1438 Desai Aakash Stamford CT F TA3 39223 15MAR1985 18NOV2007
203/781-2229
1125 Dunlap Donna New York NY F FA2 28888 08NOV1988 11DEC2007
718/383-2094
1475 Eaton Alicia New York NY F FA2 27787 15DEC1981 13JUL2010
718/383-2828
1117 Edgerton Joshua New York NY M TA3 39771 05JUN1983 13AUG2012
212/588-1239
1935 Fernandez Katrina Bridgeport CT F NA2 51081 28MAR1974 16OCT2011
203/675-2962
1124 Fields Diana White Plains NY F FA1 23177 10JUL1978 01OCT2010
914/455-2998
1422 Fletcher Marie Princeton NJ F FA1 22454 04JUN1984 06APR2011
201/812-0902
1616 Flowers Annette New York NY F TA2 34137 01MAR1990 04JUN2013
718/384-3329
1406 Foster Gerald Bridgeport CT M ME2 35185 08MAR1981 17FEB2007
203/675-6363
1120 Garcia Jack New York NY M ME1 28619 11SEP1992 07OCT2013
718/384-4930
1094 Gomez Alan Bridgeport CT M FA1 22268 02APR1990 17APR2011
203/675-7181
1389 Gordon Levi New York NY M BCK 25028 15JUL1979 18AUG2010
718/384-9326
1905 Graham Alvin New York NY M PT1 65111 16APR1992 29MAY2012
212/586-8815
1407 Grant Daniel Mt. Vernon NY M PT1 68096 23MAR1989 18MAR2010
914/468-1616
1114 Green Janice New York NY F TA2 32928 18SEP1989 27JUN2007
212/588-1092
;

```

---

## ENERGY

```

data energy;
 length State $2;
 input Region Division state $ Type Expenditures;
 datalines;
1 1 ME 1 708
1 1 ME 2 379
1 1 NH 1 597
1 1 NH 2 301
1 1 VT 1 353
1 1 VT 2 188
1 1 MA 1 3264

```

```

1 1 MA 2 2498
1 1 RI 1 531
1 1 RI 2 358
1 1 CT 1 2024
1 1 CT 2 1405
1 2 NY 1 8786
1 2 NY 2 7825
1 2 NJ 1 4115
1 2 NJ 2 3558
1 2 PA 1 6478
1 2 PA 2 3695
4 3 MT 1 322
4 3 MT 2 232
4 3 ID 1 392
4 3 ID 2 298
4 3 WY 1 194
4 3 WY 2 184
4 3 CO 1 1215
4 3 CO 2 1173
4 3 NM 1 545
4 3 NM 2 578
4 3 AZ 1 1694
4 3 AZ 2 1448
4 3 UT 1 621
4 3 UT 2 438
4 3 NV 1 493
4 3 NV 2 378
4 4 WA 1 1680
4 4 WA 2 1122
4 4 OR 1 1014
4 4 OR 2 756
4 4 CA 1 10643
4 4 CA 2 10114
4 4 AK 1 349
4 4 AK 2 329
4 4 HI 1 273
4 4 HI 2 298
;

```

---

# EXP Library

---

## EXP.RESULTS

The following sections are the raw data and DATA steps for the EXP library.

```
options ps=40 ls=64 nodate pageno=1;
```

```
LIBNAME exp 'library-name';

data exp.results;
 set exp.wght(firstobs=1 obs=11 keep=id treat initwt wt3mos
 age);
 if age>100 then delete;
run;
proc print data=exp.results noobs;
 title 'The RESULTS Data Set';
run;
proc datasets library=exp;

data exp.results;
 input id treat $ initwt wt3mos age;
 datalines;
1 Other 166.28 146.98 35
2 Other 214.42 210.22 54
3 Other 172.46 159.42 33
5 Other 175.41 160.66 37
6 Other 173.13 169.40 20
7 Other 181.25 170.94 30
10 Other 239.83 214.48 48
11 Other 175.32 162.66 51
12 Other 227.01 211.06 29
13 Other 274.82 251.82 31
;
run;
```

---

## EXP.SUR

```
data exp.sur;
 input id treat $ initwt wt3mos wt6mos age;
 datalines;
14 surgery 203.60 169.78 143.88 38
17 surgery 171.52 150.33 123.18 42
18 surgery 207.46 155.22 . 41
;
run;
```

---

## EXPREV

```
data exprev;
input Country $ 1-24 Emp_ID $ 25-32 Order_Date $ Ship_Date $ Sale_Type $ 67-75
Quantity Price Cost;

datalines;
```



Antarctica	99999999	1/1/18	1/7/18	Internet	2
92.60 20.70					
Puerto Rico	99999999	1/1/18	1/5/18	Catalog	14
51.20 12.10					
Virgin Islands (U.S.)	99999999	1/1/18	1/4/18	In Store	25
31.10 15.65					
Aruba	99999999	1/1/18	1/4/18	Catalog	30
123.70 59.00					
Bahamas	99999999	1/1/18	1/4/18	Catalog	8
113.40 28.45					
Bermuda	99999999	1/1/18	1/4/18	Catalog	7
41.00 9.25					
Belize	120458	1/2/18	1/2/18	In Store	2
146.40 36.70					
British Virgin Islands	99999999	1/2/18	1/5/18	Catalog	11
40.20 20.20					
Canada	99999999	1/2/18	1/5/18	Catalog	100
11.80 5.00					
Cayman Islands	120454	1/2/18	1/2/18	In Store	20
71.00 32.30					
Costa Rica	99999999	1/2/18	1/6/18	Internet	31
53.00 26.60					
Cuba	121044	1/2/18	1/2/18	Internet	12
42.40 19.35					
Dominican Republic	121040	1/2/18	1/2/18	Internet	13
48.00 23.95					
El Salvador	99999999	1/2/18	1/6/18	Catalog	21
266.40 66.70					
Guatemala	120931	1/2/18	1/2/18	In Store	13
144.40 65.70					
Haiti	121059	1/2/18	1/2/18	Internet	5
47.90 23.45					
Honduras	120455	1/2/18	1/2/18	Internet	20
66.40 30.25					
Jamaica	99999999	1/2/18	1/4/18	In Store	23
169.80 38.70					
Mexico	120127	1/2/18	1/2/18	In Store	30
211.80 33.65					
Montserrat	120127	1/2/18	1/2/18	In Store	19
184.20 36.90					
Nicaragua	120932	1/2/18	1/2/18	Internet	16
122.00 28.75					
Panama	99999999	1/2/18	1/6/18	Internet	20
88.20 38.40					
Saint Kitts/Nevis	99999999	1/2/18	1/6/18	Internet	20
41.40 18.00					
St. Helena	120360	1/2/18	1/2/18	Internet	19
94.70 47.45					
St. Pierre/Miquelon	120842	1/2/18	1/6/18	Internet	16
103.80 47.25					
Turks/Caicos Islands	120372	1/2/18	1/2/18	Internet	10
57.70 28.95					
United States	120372	1/2/18	1/2/18	Internet	20
88.20 38.40					
Anguilla	99999999	1/2/18	1/6/18	In Store	15
233.50 22.25					

Antigua/Barbuda	120458	1/2/18	1/2/18	In Store	31
99.60 45.35					
Argentina	99999999	1/2/18	1/6/18	In Store	42
408.80 87.15					
Barbados	99999999	1/2/18	1/6/18	In Store	26
94.80 42.60					
Bolivia	120127	1/2/18	1/2/18	In Store	26
66.00 16.60					
Brazil	120127	1/2/18	1/2/18	Catalog	12
73.40 18.45					
Chile	120447	1/2/18	1/2/18	In Store	20
19.10 8.75					
Colombia	121059	1/2/18	1/2/18	Internet	28
361.40 90.45					
Dominica	121043	1/2/18	1/2/18	Internet	35
121.30 57.80					
Ecuador	121042	1/2/18	1/2/18	In Store	11
100.90 50.55					
Falkland Islands	120932	1/2/18	1/2/18	In Store	15
61.40 30.80					
French Guiana	120935	1/2/18	1/2/18	Catalog	15
96.40 43.85					
Grenada	120931	1/2/18	1/2/18	Catalog	19
56.30 25.05					
Guadeloupe	120445	1/2/18	1/2/18	Internet	21
231.60 48.70					
Guyana	120455	1/2/18	1/2/18	In Store	25
132.80 30.25					
Martinique	120841	1/2/18	1/3/18	In Store	16
56.30 31.05					
Netherlands Antilles	99999999	1/2/18	1/6/18	In Store	31
41.80 19.45					
Paraguay	120603	1/2/18	1/2/18	Catalog	17
117.60 58.90					
Peru	120845	1/2/18	1/2/18	Catalog	12
93.80 41.75					
St. Lucia	120845	1/2/18	1/2/18	Internet	19
64.30 28.65					
Suriname	120538	1/3/18	1/3/18	Internet	22
110.80 29.35					
;					
run;					

---

## GROC

```
data groc;
 input Region $9. Manager $ Department $ Sales;
 datalines;
Southeast Hayes Paper 250
Southeast Hayes Produce 100
Southeast Hayes Canned 120
```

Southeast	Hayes	Meat	80
Southeast	Michaels	Paper	40
Southeast	Michaels	Produce	300
Southeast	Michaels	Canned	220
Southeast	Michaels	Meat	70
Northwest	Jeffreys	Paper	60
Northwest	Jeffreys	Produce	600
Northwest	Jeffreys	Canned	420
Northwest	Jeffreys	Meat	30
Northwest	Duncan	Paper	45
Northwest	Duncan	Produce	250
Northwest	Duncan	Canned	230
Northwest	Duncan	Meat	73
Northwest	Aikmann	Paper	45
Northwest	Aikmann	Produce	205
Northwest	Aikmann	Canned	420
Northwest	Aikmann	Meat	76
Southwest	Royster	Paper	53
Southwest	Royster	Produce	130
Southwest	Royster	Canned	120
Southwest	Royster	Meat	50
Southwest	Patel	Paper	40
Southwest	Patel	Produce	350
Southwest	Patel	Canned	225
Southwest	Patel	Meat	80
Northeast	Rice	Paper	90
Northeast	Rice	Produce	90
Northeast	Rice	Canned	420
Northeast	Rice	Meat	86
Northeast	Fuller	Paper	200
Northeast	Fuller	Produce	300
Northeast	Fuller	Canned	420
Northeast	Fuller	Meat	125

;

---

## MATCH\_11

```
data match_11;
 input Pair Low Age Lwt Race Smoke Ptd Ht UI @@;
 select(race);
 when (1) do;
 race1=0;
 race2=0;
 end;
 when (2) do;
 race1=1;
 race2=0;
 end;
 when (3) do;
 race1=0;
 race2=1;
 end;
 end;
```

```

 end;
 end;
 datalines;
1 0 14 135 1 0 0 0 0 1 1 14 101 3 1 1 0 0
2 0 15 98 2 0 0 0 0 2 1 15 115 3 0 0 0 1
3 0 16 95 3 0 0 0 0 3 1 16 130 3 0 0 0 0
4 0 17 103 3 0 0 0 0 4 1 17 130 3 1 1 0 1
5 0 17 122 1 1 0 0 0 5 1 17 110 1 1 0 0 0
6 0 17 113 2 0 0 0 0 6 1 17 120 1 1 0 0 0
7 0 17 113 2 0 0 0 0 7 1 17 120 2 0 0 0 0
8 0 17 119 3 0 0 0 0 8 1 17 142 2 0 0 1 0
9 0 18 100 1 1 0 0 0 9 1 18 148 3 0 0 0 0
10 0 18 90 1 1 0 0 1 10 1 18 110 2 1 1 0 0
11 0 19 150 3 0 0 0 0 11 1 19 91 1 1 1 0 1
12 0 19 115 3 0 0 0 0 12 1 19 102 1 0 0 0 0
13 0 19 235 1 1 0 1 0 13 1 19 112 1 1 0 0 1
14 0 20 120 3 0 0 0 1 14 1 20 150 1 1 0 0 0
15 0 20 103 3 0 0 0 0 15 1 20 125 3 0 0 0 1
16 0 20 169 3 0 1 0 1 16 1 20 120 2 1 0 0 0
17 0 20 141 1 0 1 0 1 17 1 20 80 3 1 0 0 1
18 0 20 121 2 1 0 0 0 18 1 20 109 3 0 0 0 0
19 0 20 127 3 0 0 0 0 19 1 20 121 1 1 1 0 1
20 0 20 120 3 0 0 0 0 20 1 20 122 2 1 0 0 0
21 0 20 158 1 0 0 0 0 21 1 20 105 3 0 0 0 0
22 0 21 108 1 1 0 0 1 22 1 21 165 1 1 0 1 0
23 0 21 124 3 0 0 0 0 23 1 21 200 2 0 0 0 0
24 0 21 185 2 1 0 0 0 24 1 21 103 3 0 0 0 0
25 0 21 160 1 0 0 0 0 25 1 21 100 3 0 1 0 0
26 0 21 115 1 0 0 0 0 26 1 21 130 1 1 0 1 0
27 0 22 95 3 0 0 1 0 27 1 22 130 1 1 0 0 0
28 0 22 158 2 0 1 0 0 28 1 22 130 1 1 1 0 1
29 0 23 130 2 0 0 0 0 29 1 23 97 3 0 0 0 1
30 0 23 128 3 0 0 0 0 30 1 23 187 2 1 0 0 0
31 0 23 119 3 0 0 0 0 31 1 23 120 3 0 0 0 0
32 0 23 115 3 1 0 0 0 32 1 23 110 1 1 1 0 0
33 0 23 190 1 0 0 0 0 33 1 23 94 3 1 0 0 0
34 0 24 90 1 1 1 0 0 34 1 24 128 2 0 1 0 0
35 0 24 115 1 0 0 0 0 35 1 24 132 3 0 0 1 0
36 0 24 110 3 0 0 0 0 36 1 24 155 1 1 1 0 0
37 0 24 115 3 0 0 0 0 37 1 24 138 1 0 0 0 0
38 0 24 110 3 0 1 0 0 38 1 24 105 2 1 0 0 0
39 0 25 118 1 1 0 0 0 39 1 25 105 3 0 1 1 0
40 0 25 120 3 0 0 0 1 40 1 25 85 3 0 0 0 1
41 0 25 155 1 0 0 0 0 41 1 25 115 3 0 0 0 0
42 0 25 125 2 0 0 0 0 42 1 25 92 1 1 0 0 0
43 0 25 140 1 0 0 0 0 43 1 25 89 3 0 1 0 0
44 0 25 241 2 0 0 1 0 44 1 25 105 3 0 1 0 0
45 0 26 113 1 1 0 0 0 45 1 26 117 1 1 1 0 0
46 0 26 168 2 1 0 0 0 46 1 26 96 3 0 0 0 0
47 0 26 133 3 1 1 0 0 47 1 26 154 3 0 1 1 0
48 0 26 160 3 0 0 0 0 48 1 26 190 1 1 0 0 0
49 0 27 124 1 1 0 0 0 49 1 27 130 2 0 0 0 1
50 0 28 120 3 0 0 0 0 50 1 28 120 3 1 1 0 1
51 0 28 130 3 0 0 0 0 51 1 28 95 1 1 0 0 0
52 0 29 135 1 0 0 0 0 52 1 29 130 1 0 0 0 1
53 0 30 95 1 1 0 0 0 53 1 30 142 1 1 1 0 0

```

```

54 0 31 215 1 1 0 0 0 54 1 31 102 1 1 1 0 0
55 0 32 121 3 0 0 0 0 55 1 32 105 1 1 0 0 0
56 0 34 170 1 0 1 0 0 56 1 34 187 2 1 0 1 0
;

```

## PROCLIB.DELAY

```

data proclib.delay;
 input flight $3. +5 date date7. +2 orig $3. +3 dest $3. +3
 delaycat $15. +2 destype $15. +8 delay;
 informat date date7.;
 format date date7.;
 datalines;

```

114	01MAR18	LGA	LAX	1-10 Minutes	Domestic	8
202	01MAR18	LGA	ORD	No Delay	Domestic	-5
219	01MAR18	LGA	LON	11+ Minutes	International	18
622	01MAR18	LGA	FRA	No Delay	International	-5
132	01MAR18	LGA	YYZ	11+ Minutes	International	14
271	01MAR18	LGA	PAR	1-10 Minutes	International	5
302	01MAR18	LGA	WAS	No Delay	Domestic	-2
114	02MAR18	LGA	LAX	No Delay	Domestic	0
202	02MAR18	LGA	ORD	1-10 Minutes	Domestic	5
219	02MAR18	LGA	LON	11+ Minutes	International	18
622	02MAR18	LGA	FRA	No Delay	International	0
132	02MAR18	LGA	YYZ	1-10 Minutes	International	5
271	02MAR18	LGA	PAR	1-10 Minutes	International	4
302	02MAR18	LGA	WAS	No Delay	Domestic	0
114	03MAR18	LGA	LAX	No Delay	Domestic	-1
202	03MAR18	LGA	ORD	No Delay	Domestic	-1
219	03MAR18	LGA	LON	1-10 Minutes	International	4
622	03MAR18	LGA	FRA	No Delay	International	-2
132	03MAR18	LGA	YYZ	1-10 Minutes	International	6
271	03MAR18	LGA	PAR	1-10 Minutes	International	2
302	03MAR18	LGA	WAS	1-10 Minutes	Domestic	5
114	04MAR18	LGA	LAX	11+ Minutes	Domestic	15
202	04MAR18	LGA	ORD	No Delay	Domestic	-5
219	04MAR18	LGA	LON	1-10 Minutes	International	3
622	04MAR18	LGA	FRA	11+ Minutes	International	30
132	04MAR18	LGA	YYZ	No Delay	International	-5
271	04MAR18	LGA	PAR	1-10 Minutes	International	5
302	04MAR18	LGA	WAS	1-10 Minutes	Domestic	7
114	05MAR18	LGA	LAX	No Delay	Domestic	-2
202	05MAR18	LGA	ORD	1-10 Minutes	Domestic	2
219	05MAR18	LGA	LON	1-10 Minutes	International	3
622	05MAR18	LGA	FRA	No Delay	International	-6
132	05MAR18	LGA	YYZ	1-10 Minutes	International	3
271	05MAR18	LGA	PAR	1-10 Minutes	International	5
114	06MAR18	LGA	LAX	No Delay	Domestic	-1
202	06MAR18	LGA	ORD	No Delay	Domestic	-3
219	06MAR18	LGA	LON	11+ Minutes	International	27
132	06MAR18	LGA	YYZ	1-10 Minutes	International	7

302	06MAR18	LGA	WAS	1-10 Minutes	Domestic	1
114	07MAR18	LGA	LAX	No Delay	Domestic	-1
202	07MAR18	LGA	ORD	No Delay	Domestic	-2
219	07MAR18	LGA	LON	11+ Minutes	International	15
622	07MAR18	LGA	FRA	11+ Minutes	International	21
132	07MAR18	LGA	YYZ	No Delay	International	-2
271	07MAR18	LGA	PAR	1-10 Minutes	International	4
302	07MAR18	LGA	WAS	No Delay	Domestic	0

;

## PROCLIB.EMP95

```
data proclib.emp95;
 input #1 idnum $4. @6 name $15.
 #2 address $42.
 #3 salary 6.;
 datalines;
2388 James Schmidt
100 Apt. C Blount St. SW Raleigh NC 27693
92100
2457 Fred Williams
99 West Lane Garner NC 27509
33190
2776 Robert Jones
12988 Wellington Farms Ave. Cary NC 27512
29025
8699 Jerry Capalleti
222 West L St. Oxford NC 27587
39985
2100 Lanny Engles
293 Manning Pl. Raleigh NC 27606
30998
9857 Kathy Krupski
1000 Taft Ave. Morrisville NC 27508
38756
0987 Dolly Lunford
2344 Persimmons Branch Apex NC 27505
44010
3286 Hoa Nguyen
2818 Long St. Cary NC 27513
87734
6579 Bryan Samosky
3887 Charles Ave. Garner NC 27508
50234
3888 Kim Siu
5662 Magnolia Blvd Southeast Cary NC 27513
77558
;
```

---

# PROCLIB.EMP96

```
data proclib.emp96;
 input #1 idnum $4. @6 name $15.
 #2 address $42.
 #3 salary 6.;
 datalines;
2388 James Schmidt
100 Apt. C Blount St. SW Raleigh NC 27693
92100
2457 Fred Williams
99 West Lane Garner NC 27509
33190
2776 Robert Jones
12988 Wellington Farms Ave. Cary NC 27511
29025
8699 Jerry Capalleti
222 West L St. Oxford NC 27587
39985
3278 Mary Cravens
211 N. Cypress St. Cary NC 27512
35362
2100 Lanny Engles
293 Manning Pl. Raleigh NC 27606
30998
9857 Kathy Krupski
100 Taft Ave. Morrisville NC 27508
40456
0987 Dolly Lunford
2344 Persimmons Branch Trail Apex NC 27505
45110
3286 Hoa Nguyen
2818 Long St. Cary NC 27513
89834
6579 Bryan Samosky
3887 Charles Ave. Garner NC 27508
50234
3888 Kim Siu
5662 Magnolia Blvd Southwest Cary NC 27513
79958
6544 Roger Monday
3004 Crepe Myrtle Court Raleigh NC 27604
47007
;
```

## PROCLIB.INTERNAT

```
data proclib.internat;
 input flight $3. +5 date date7. +2 dest $3. +8 boarded;
 informat date date7.;
 format date date7.;
 datalines;
219 01MAR18 LON 198
622 01MAR18 FRA 207
132 01MAR18 YYZ 115
271 01MAR18 PAR 138
219 02MAR18 LON 147
622 02MAR18 FRA 176
132 02MAR18 YYZ 106
271 02MAR18 PAR 172
219 03MAR18 LON 197
622 03MAR18 FRA 180
132 03MAR18 YYZ 75
271 03MAR18 PAR 147
219 04MAR18 LON 232
622 04MAR18 FRA 137
132 04MAR18 YYZ 117
271 04MAR18 PAR 146
219 05MAR18 LON 160
622 05MAR18 FRA 185
132 05MAR18 YYZ 157
271 05MAR18 PAR 177
219 06MAR18 LON 163
132 06MAR18 YYZ 150
219 07MAR18 LON 241
622 07MAR18 FRA 210
132 07MAR18 YYZ 164
271 07MAR18 PAR 155
;
```

## PROCLIB.LAKES

```
data proclib.lakes;
 input region $ 1-2 lake $ 5-13 pol_a1 pol_a2 pol_b1-pol_b4;
 datalines;
NE Carr 0.24 0.99 0.95 0.36 0.44 0.67
NE Duraleigh 0.34 0.01 0.48 0.58 0.12 0.56
NE Charlie 0.40 0.48 0.29 0.56 0.52 0.95
NE Farmer 0.60 0.65 0.25 0.20 0.30 0.64
```



NW	Canyon	0.63	0.44	0.20	0.98	0.19	0.01
NW	Morris	0.85	0.95	0.80	0.67	0.32	0.81
NW	Golf	0.69	0.37	0.08	0.72	0.71	0.32
NW	Falls	0.01	0.02	0.59	0.58	0.67	0.02
SE	Pleasant	0.16	0.96	0.71	0.35	0.35	0.48
SE	Juliette	0.82	0.35	0.09	0.03	0.59	0.90
SE	Massey	1.01	0.77	0.45	0.32	0.55	0.66
SE	Delta	0.84	1.05	0.90	0.09	0.64	0.03
SW	Alumni	0.45	0.32	0.45	0.44	0.55	0.12
SW	New Dam	0.80	0.70	0.31	0.98	1.00	0.22
SW	Border	0.51	0.04	0.55	0.35	0.45	0.78
SW	Red	0.22	0.09	0.02	0.10	0.32	0.01

;

---

## PROCLIB.MARCH

```
data proclib.march;
 input flight $3. +5 date date7. +3 depart time5. +2 orig $3.
 +3 dest $3. +7 miles +6 boarded +6 capacity;
 format date date7. depart time5.;
 informat date date7. depart time5.;
 datalines;
```

114	01MAR18	7:10	LGA	LAX	2475	172	210
202	01MAR18	10:43	LGA	ORD	740	151	210
219	01MAR18	9:31	LGA	LON	3442	198	250
622	01MAR18	12:19	LGA	FRA	3857	207	250
132	01MAR18	15:35	LGA	YYZ	366	115	178
271	01MAR18	13:17	LGA	PAR	3635	138	250
302	01MAR18	20:22	LGA	WAS	229	105	180
114	02MAR18	7:10	LGA	LAX	2475	119	210
202	02MAR18	10:43	LGA	ORD	740	120	210
219	02MAR18	9:31	LGA	LON	3442	147	250
622	02MAR18	12:19	LGA	FRA	3857	176	250
132	02MAR18	15:35	LGA	YYZ	366	106	178
302	02MAR18	20:22	LGA	WAS	229	78	180
271	02MAR18	13:17	LGA	PAR	3635	104	250
114	03MAR18	7:10	LGA	LAX	2475	197	210
202	03MAR18	10:43	LGA	ORD	740	118	210
219	03MAR18	9:31	LGA	LON	3442	197	250
622	03MAR18	12:19	LGA	FRA	3857	180	250
132	03MAR18	15:35	LGA	YYZ	366	75	178
271	03MAR18	13:17	LGA	PAR	3635	147	250
302	03MAR18	20:22	LGA	WAS	229	123	180
114	04MAR18	7:10	LGA	LAX	2475	178	210
202	04MAR18	10:43	LGA	ORD	740	148	210
219	04MAR18	9:31	LGA	LON	3442	232	250
622	04MAR18	12:19	LGA	FRA	3857	137	250
132	04MAR18	15:35	LGA	YYZ	366	117	178
271	04MAR18	13:17	LGA	PAR	3635	146	250
302	04MAR18	20:22	LGA	WAS	229	115	180
114	05MAR18	7:10	LGA	LAX	2475	117	210

202	05MAR18	10:43	LGA	ORD	740	104	210
219	05MAR18	9:31	LGA	LON	3442	160	250
622	05MAR18	12:19	LGA	FRA	3857	185	250
132	05MAR18	15:35	LGA	YYZ	366	157	178
271	05MAR18	13:17	LGA	PAR	3635	177	250
114	06MAR18	7:10	LGA	LAX	2475	128	210
202	06MAR18	10:43	LGA	ORD	740	115	210
219	06MAR18	9:31	LGA	LON	3442	163	250
132	06MAR18	15:35	LGA	YYZ	366	150	178
302	06MAR18	20:22	LGA	WAS	229	66	180
114	07MAR18	7:10	LGA	LAX	2475	160	210
202	07MAR18	10:43	LGA	ORD	740	175	210
219	07MAR18	9:31	LGA	LON	3442	241	250
622	07MAR18	12:19	LGA	FRA	3857	210	250
132	07MAR18	15:35	LGA	YYZ	366	164	178
271	07MAR18	13:17	LGA	PAR	3635	155	250
302	07MAR18	20:22	LGA	WAS	229	135	180

;

---

## PROCLIB.PAYLIST2

```
proc sql;
 create table proclib.paylist2
 (IdNum char(4),
 Gender char(1),
 Jobcode char(3),
 Salary num,
 Birth num informat=date9.
 format=date9.,
 Hired num informat=date9.
 format=date9.);

insert into proclib.paylist2
values('1919','M','TA2',34376,'12SEP1990'd,'04JUN2007'd)
values('1653','F','ME2',31896,'15OCT1984'd,'09AUG2010'd)
values('1350','F','FA3',36886,'31AUG1985'd,'29JUL2010'd)
values('1401','M','TA3',38822,'13DEC1970'd,'17NOV2005'd)
values('1499','M','ME1',23025,'26APR1974'd,'07JUN2010'd);

title 'PROCLIB.PAYLIST2 Table';
select * from proclib.paylist2;
```

# PROCLIB.PAYROLL

This data set (table) is updated in “[Updating Data in a PROC SQL Table](#)” in *SAS SQL Procedure User’s Guide*, and its updated data is used in subsequent examples.

```
data proclib.payroll;
 input IdNumber $4. +3 Gender $1. +4 Jobcode $3. +8 Salary 6.
 +2 Birth date9. +2 Hired date9.;
 informat birth date9. hired date9.;
 format birth date9. hired date9.;
 datalines;
1919 M TA2 34376 12SEP1990 04JUN2007
1653 F ME2 35108 15OCT1984 09AUG2010
1400 M ME1 29769 05NOV1987 16OCT2010
1350 F FA3 32886 31AUG1985 29JUL2010
1401 M TA3 38822 13DEC1970 17NOV2005
1499 M ME3 43025 26APR1974 07JUN2010
1101 M SCP 18723 06JUN1982 01OCT2010
1333 M PT2 88606 30MAR1981 10FEB2011
1402 M TA2 32615 17JAN1983 02DEC2010
1479 F TA3 38785 22DEC1988 05OCT2007
1403 M ME1 28072 28JAN1989 21DEC2016
1739 M PT1 66517 25DEC1984 27JAN2011
1658 M SCP 17943 08APR1987 29FEB2012
1428 F PT1 68767 04APR1980 16NOV2011
1782 M ME2 35345 04DEC1980 22FEB2012
1244 M ME2 36925 31AUG1983 17JAN2008
1383 M BCK 25823 25JAN1985 20OCT2009
1574 M FA2 28572 27APR1980 20DEC2012
1789 M SCP 18326 23JAN1977 11APR2008
1404 M PT2 91376 24FEB1973 01JAN2010
1437 F FA3 33104 20SEP1980 31AUG2004
1639 F TA3 40260 26JUN1977 28JAN2014
1269 M NA1 41690 03MAY1972 28NOV2012
1065 M ME2 35090 26JAN1977 07JAN2007
1876 M TA3 39675 20MAY1978 27APR2015
1037 F TA1 28558 10APR1984 13SEP2012
1129 F ME2 34929 08DEC1981 17AUG2011
1988 M FA3 32217 30NOV1979 18SEP2004
1405 M SCP 18056 05MAR1986 26JAN2012
1430 F TA2 32925 28FEB1982 27APR2017
1983 F FA3 33419 28FEB1962 30APR2005
1134 F TA2 33462 05MAR1989 21DEC2016
1118 M PT3 111379 16JAN1964 18DEC2000
1438 F TA3 39223 15MAR1985 18NOV2007
1125 F FA2 28888 08NOV1988 11DEC2007
1475 F FA2 27787 15DEC1981 13JUL2010
1117 M TA3 39771 05JUN1983 13AUG2012
1935 F NA2 51081 28MAR1974 16OCT2011
1124 F FA1 23177 10JUL1978 01OCT2010
```

1422	F	FA1	22454	04JUN1984	06APR2011
1616	F	TA2	34137	01MAR1990	04JUN2013
1406	M	ME2	35185	08MAR1981	17FEB2007
1120	M	ME1	28619	11SEP1992	07OCT2013
1094	M	FA1	22268	02APR1990	17APR2011
1389	M	BCK	25028	15JUL1979	18AUG2010
1905	M	PT1	65111	16APR1992	29MAY2012
1407	M	PT1	68096	23MAR1989	18MAR2010
1114	F	TA2	32928	18SEP1989	27JUN2007
1410	M	PT2	84685	03MAY1987	07NOV2006
1439	F	PT1	70736	06MAR1984	10SEP2010
1409	M	ME3	41551	19APR1970	22OCT2001
1408	M	TA2	34138	29MAR1980	14OCT2007
1121	M	ME1	29112	26SEP1981	07DEC2011
1991	F	TA1	27645	07MAY1982	12DEC2012
1102	M	TA2	34542	01OCT1979	15APR2011
1356	M	ME2	36869	26SEP1977	22FEB2003
1545	M	PT1	66130	12AUG1979	29MAY2010
1292	F	ME2	36691	28OCT1984	02JUL2009
1440	F	ME2	35757	27SEP1982	09APR2011
1368	M	FA2	27808	11JUN1981	03NOV2014
1369	M	TA2	33705	28DEC1981	13MAR2017
1411	M	FA2	27265	27MAY1981	01DEC2017
1113	F	FA1	22367	15JAN1988	17OCT2011
1704	M	BCK	25465	30AUG1986	28JUN2007
1900	M	ME2	35105	25MAY1982	27OCT2007
1126	F	TA3	40899	28MAY1983	21NOV2010
1677	M	BCK	26007	05NOV1983	27MAR2009
1441	F	FA2	27158	19NOV1989	23MAR2011
1421	M	TA2	33155	08JAN1979	28FEB2010
1119	M	TA1	26924	20JUN1982	06SEP2008
1834	M	BCK	26896	08FEB1982	02JUL2012
1777	M	PT3	109630	23SEP1971	21JUN2001
1663	M	BCK	26452	11JAN1987	11AUG2011
1106	M	PT2	89632	06NOV1977	16AUG2004
1103	F	FA1	23738	16FEB1988	23JUL2012
1477	M	FA2	28566	21MAR1984	07MAR2008
1476	F	TA2	34803	30MAY1986	17MAR2017
1379	M	ME3	42264	08AUG1981	10JUN2004
1104	M	SCP	17946	25APR1983	10JUN2011
1009	M	TA1	28880	02MAR1979	26MAR2012
1412	M	ME1	27799	18JUN1976	05DEC2011
1115	F	FA3	32699	22AUG1980	28FEB2010
1128	F	TA2	32777	23MAY1985	20OCT2010
1442	F	PT2	84536	05SEP1986	12APR2008
1417	M	NA2	52270	27JUN1984	07MAR2009
1478	M	PT2	84203	09AUG1979	24OCT2010
1673	M	BCK	25477	27FEB1980	15JUL2011
1839	F	NA1	43433	29NOV1980	03JUL2013
1347	M	TA3	40079	21SEP1987	06SEP2014
1423	F	ME2	35773	14MAY1988	19AUG2010
1200	F	ME1	27816	10JAN1981	14AUG2012
1970	F	FA1	22615	25SEP1984	12MAR2011
1521	M	ME3	41526	12APR1983	13JUL2008
1354	F	SCP	18335	29MAY1981	16JUN2012
1424	F	FA2	28978	04AUG1979	11DEC2009

1132	F	FA1	22413	30MAY1972	22OCT2003
1845	M	BCK	25996	20NOV1979	22MAR2000
1556	M	PT1	71349	22JUN1984	11DEC2011
1413	M	FA2	27435	16SEP1975	02JAN2010
1123	F	TA1	28407	31OCT1982	05DEC2012
1907	M	TA2	33329	15NOV1980	06JUL2007
1436	F	TA2	34475	11JUN1984	12MAR2007
1385	M	ME3	43900	16JAN1982	01APR2016
1432	F	ME2	35327	03NOV1981	10FEB2015
1111	M	NA1	40586	14JUL1983	31OCT2012
1116	F	FA1	22862	28SEP1979	21MAR2011
1352	M	NA2	53798	02DEC1980	16OCT2006
1555	F	FA2	27499	16MAR1978	04JUL2012
1038	F	TA1	26533	09NOV1989	23NOV2011
1420	M	ME3	43071	19FEB1985	22JUL2017
1561	M	TA2	34514	30NOV1983	07OCT2007
1434	F	FA2	28622	11JUL1982	28OCT2010
1414	M	FA1	23644	24MAR1982	12APR2012
1112	M	TA1	26905	29NOV1984	07DEC2012
1390	M	FA2	27761	19FEB1985	23JUN2011
1332	M	NA1	42178	17SEP1980	04JUN2011
1890	M	PT2	91908	20JUL1981	25NOV2017
1429	F	TA1	27939	28FEB1980	07AUG2012
1107	M	PT2	89977	09JUN1974	10FEB2009
1908	F	TA2	32995	10DEC1979	23APR2010
1830	F	PT2	84471	27MAY1977	29JAN2013
1882	M	ME3	41538	10JUL1977	21NOV2008
1050	M	ME2	35167	14JUL1983	24AUG2016
1425	F	FA1	23979	28DEC1981	28FEB2013
1928	M	PT2	89858	16SEP1974	13JUL2010
1480	F	TA3	39583	03SEP1977	25MAR2001
1100	M	BCK	25004	01DEC1980	07MAY2008
1995	F	ME1	28810	24AUG1983	19SEP2013
1135	F	FA2	27321	20SEP1980	31MAR2010
1415	M	FA2	28278	09MAR1978	12FEB2008
1076	M	PT1	66558	14OCT1975	03OCT2011
1426	F	TA2	32991	05DEC1986	25JUN2010
1564	F	SCP	18833	12APR1982	01JUL2012
1221	F	FA2	27896	22SEP1987	04OCT2011
1133	M	TA1	27701	13JUL1986	12FEB2012
1435	F	TA3	38808	12MAY1979	08FEB2010
1418	M	ME1	28005	29MAR1977	06JAN2012
1017	M	TA3	40858	28DEC1977	16OCT2011
1443	F	NA1	42274	17NOV1988	29AUG2011
1131	F	TA2	32575	26DEC1981	19APR2011
1427	F	TA2	34046	31OCT1990	30JAN2010
1036	F	TA3	39392	19MAY1985	23OCT2004
1130	F	FA1	23916	16MAY1981	05JUN2012
1127	F	TA2	33011	09NOV1984	07DEC2006
1433	F	FA3	32982	08JUL1986	17JAN2007
1431	F	FA3	33230	09JUN1984	05APR2008
1122	F	FA2	27956	01MAY1983	27NOV2008
1105	M	ME2	34805	01MAR1982	13AUG2010

;

## PROCLIB.PAYROLL2

```
data proclib.payroll2;
 input idnum $4. +3 gender $1. +4 jobcode $3. +9 salary 5.
 +2 birth date9. +2 hired date9.;
 informat birth date9. hired date9.;
 format birth date9. hired date9.;
 datalines;
1639 F TA3 42260 26JUN1977 28JAN2004
1065 M ME3 38090 26JAN1964 07JAN2007
1561 M TA3 36514 30NOV1983 07OCT2007
1221 F FA3 29896 22SEP1987 04OCT2011
1447 F FA1 22123 07AUG1982 29OCT2002
1998 M SCP 23100 10SEP1970 02NOV2012
1036 F TA3 42465 19MAY1965 23OCT2004
1106 M PT3 94039 06NOV1957 16AUG2004
1129 F ME3 36758 08DEC1961 17AUG2011
1350 F FA3 36098 31AUG1985 29JUL2010
1369 M TA3 36598 28DEC1961 13MAR2007
1076 M PT1 69742 14OCT1955 03OCT2011
;
```

## PROCLIB.SCHEDULE

```
data proclib.schedule;
 input flight $3. +5 date date7. +2 dest $3. +3 idnum $4.;
 format date date7.;
 informat date date7.;
 datalines;
132 01MAR18 YYZ 1739
132 01MAR18 YYZ 1478
132 01MAR18 YYZ 1130
132 01MAR18 YYZ 1390
132 01MAR18 YYZ 1983
132 01MAR18 YYZ 1111
219 01MAR18 LON 1407
219 01MAR18 LON 1777
219 01MAR18 LON 1103
219 01MAR18 LON 1125
219 01MAR18 LON 1350
219 01MAR18 LON 1332
271 01MAR18 PAR 1439
271 01MAR18 PAR 1442
271 01MAR18 PAR 1132
```

271	01MAR18	PAR	1411
271	01MAR18	PAR	1988
271	01MAR18	PAR	1443
622	01MAR18	FRA	1545
622	01MAR18	FRA	1890
622	01MAR18	FRA	1116
622	01MAR18	FRA	1221
622	01MAR18	FRA	1433
622	01MAR18	FRA	1352
132	02MAR18	YYZ	1556
132	02MAR18	YYZ	1478
132	02MAR18	YYZ	1113
132	02MAR18	YYZ	1411
132	02MAR18	YYZ	1574
132	02MAR18	YYZ	1111
219	02MAR18	LON	1407
219	02MAR18	LON	1118
219	02MAR18	LON	1132
219	02MAR18	LON	1135
219	02MAR18	LON	1441
219	02MAR18	LON	1332
271	02MAR18	PAR	1739
271	02MAR18	PAR	1442
271	02MAR18	PAR	1103
271	02MAR18	PAR	1413
271	02MAR18	PAR	1115
271	02MAR18	PAR	1443
622	02MAR18	FRA	1439
622	02MAR18	FRA	1890
622	02MAR18	FRA	1124
622	02MAR18	FRA	1368
622	02MAR18	FRA	1477
622	02MAR18	FRA	1352
132	03MAR18	YYZ	1739
132	03MAR18	YYZ	1928
132	03MAR18	YYZ	1425
132	03MAR18	YYZ	1135
132	03MAR18	YYZ	1437
132	03MAR18	YYZ	1111
219	03MAR18	LON	1428
219	03MAR18	LON	1442
219	03MAR18	LON	1130
219	03MAR18	LON	1411
219	03MAR18	LON	1115
219	03MAR18	LON	1332
271	03MAR18	PAR	1905
271	03MAR18	PAR	1118
271	03MAR18	PAR	1970
271	03MAR18	PAR	1125
271	03MAR18	PAR	1983
271	03MAR18	PAR	1443
622	03MAR18	FRA	1545
622	03MAR18	FRA	1830
622	03MAR18	FRA	1414
622	03MAR18	FRA	1368
622	03MAR18	FRA	1431

622	03MAR18	FRA	1352
132	04MAR18	YYZ	1428
132	04MAR18	YYZ	1118
132	04MAR18	YYZ	1103
132	04MAR18	YYZ	1390
132	04MAR18	YYZ	1350
132	04MAR18	YYZ	1111
219	04MAR18	LON	1739
219	04MAR18	LON	1478
219	04MAR18	LON	1130
219	04MAR18	LON	1125
219	04MAR18	LON	1983
219	04MAR18	LON	1332
271	04MAR18	PAR	1407
271	04MAR18	PAR	1410
271	04MAR18	PAR	1094
271	04MAR18	PAR	1411
271	04MAR18	PAR	1115
271	04MAR18	PAR	1443
622	04MAR18	FRA	1545
622	04MAR18	FRA	1890
622	04MAR18	FRA	1116
622	04MAR18	FRA	1221
622	04MAR18	FRA	1433
622	04MAR18	FRA	1352
132	05MAR18	YYZ	1556
132	05MAR18	YYZ	1890
132	05MAR18	YYZ	1113
132	05MAR18	YYZ	1475
132	05MAR18	YYZ	1431
132	05MAR18	YYZ	1111
219	05MAR18	LON	1428
219	05MAR18	LON	1442
219	05MAR18	LON	1422
219	05MAR18	LON	1413
219	05MAR18	LON	1574
219	05MAR18	LON	1332
271	05MAR18	PAR	1739
271	05MAR18	PAR	1928
271	05MAR18	PAR	1103
271	05MAR18	PAR	1477
271	05MAR18	PAR	1433
271	05MAR18	PAR	1443
622	05MAR18	FRA	1545
622	05MAR18	FRA	1830
622	05MAR18	FRA	1970
622	05MAR18	FRA	1441
622	05MAR18	FRA	1350
622	05MAR18	FRA	1352
132	06MAR18	YYZ	1333
132	06MAR18	YYZ	1890
132	06MAR18	YYZ	1414
132	06MAR18	YYZ	1475
132	06MAR18	YYZ	1437
132	06MAR18	YYZ	1111
219	06MAR18	LON	1106



```

219 06MAR18 LON 1118
219 06MAR18 LON 1425
219 06MAR18 LON 1434
219 06MAR18 LON 1555
219 06MAR18 LON 1332
132 07MAR18 YYZ 1407
132 07MAR18 YYZ 1118
132 07MAR18 YYZ 1094
132 07MAR18 YYZ 1555
132 07MAR18 YYZ 1350
132 07MAR18 YYZ 1111
219 07MAR18 LON 1905
219 07MAR18 LON 1478
219 07MAR18 LON 1124
219 07MAR18 LON 1434
219 07MAR18 LON 1983
219 07MAR18 LON 1332
271 07MAR18 PAR 1410
271 07MAR18 PAR 1777
271 07MAR18 PAR 1103
271 07MAR18 PAR 1574
271 07MAR18 PAR 1115
271 07MAR18 PAR 1443
622 07MAR18 FRA 1107
622 07MAR18 FRA 1890
622 07MAR18 FRA 1425
622 07MAR18 FRA 1475
622 07MAR18 FRA 1433
622 07MAR18 FRA 1352
;

```

---

## PROCLIB.STAFF

```

data proclib.staff;
 input idnum $4. +3 lname $15. +2 fname $15. +2 city $15. +2
 state $2. +5 hphone $12.;
 datalines;
1919 ADAMS GERALD STAMFORD CT 203/781-1255
1653 ALIBRANDI MARIA BRIDGEPORT CT 203/675-7715
1400 ALHERTANI ABDULLAH NEW YORK NY 212/586-0808
1350 ALVAREZ MERCEDES NEW YORK NY 718/383-1549
1401 ALVAREZ CARLOS PATERSON NJ 201/732-8787
1499 BAREFOOT JOSEPH PRINCETON NJ 201/812-5665
1101 BAUCOM WALTER NEW YORK NY 212/586-8060
1333 BANADYGA JUSTIN STAMFORD CT 203/781-1777
1402 BLALOCK RALPH NEW YORK NY 718/384-2849
1479 BALLETTI MARIE NEW YORK NY 718/384-8816
1403 BOWDEN EARL BRIDGEPORT CT 203/675-3434
1739 BRANCACCIO JOSEPH NEW YORK NY 212/587-1247
1658 BREUHAUS JEREMY NEW YORK NY 212/587-3622
1428 BRADY CHRISTINE STAMFORD CT 203/781-1212

```

1782	BREWCZAK	JAKOB	STAMFORD	CT	203/781-0019
1244	BUCCI	ANTHONY	NEW YORK	NY	718/383-3334
1383	BURNETTE	THOMAS	NEW YORK	NY	718/384-3569
1574	CAHILL	MARSHALL	NEW YORK	NY	718/383-2338
1789	CARAWAY	DAVIS	NEW YORK	NY	212/587-9000
1404	COHEN	LEE	NEW YORK	NY	718/384-2946
1437	CARTER	DOROTHY	BRIDGEPORT	CT	203/675-4117
1639	CARTER-COHEN	KAREN	STAMFORD	CT	203/781-8839
1269	CASTON	FRANKLIN	STAMFORD	CT	203/781-3335
1065	COPAS	FREDERICO	NEW YORK	NY	718/384-5618
1876	CHIN	JACK	NEW YORK	NY	212/588-5634
1037	CHOW	JANE	STAMFORD	CT	203/781-8868
1129	COUNIHAN	BRENDA	NEW YORK	NY	718/383-2313
1988	COOPER	ANTHONY	NEW YORK	NY	212/587-1228
1405	DACKO	JASON	PATERSON	NJ	201/732-2323
1430	DABROWSKI	SANDRA	BRIDGEPORT	CT	203/675-1647
1983	DEAN	SHARON	NEW YORK	NY	718/384-1647
1134	DELGADO	MARIA	STAMFORD	CT	203/781-1528
1118	DENNIS	ROGER	NEW YORK	NY	718/383-1122
1438	DABBOUSSI	KAMILLA	STAMFORD	CT	203/781-2229
1125	DUNLAP	DONNA	NEW YORK	NY	718/383-2094
1475	ELGES	MARGARETE	NEW YORK	NY	718/383-2828
1117	EDGERTON	JOSHUA	NEW YORK	NY	212/588-1239
1935	FERNANDEZ	KATRINA	BRIDGEPORT	CT	203/675-2962
1124	FIELDS	DIANA	WHITE PLAINS	NY	914/455-2998
1422	FUJIHARA	KYOKO	PRINCETON	NJ	201/812-0902
1616	FUENTAS	CARLA	NEW YORK	NY	718/384-3329
1406	FOSTER	GERALD	BRIDGEPORT	CT	203/675-6363
1120	GARCIA	JACK	NEW YORK	NY	718/384-4930
1094	GOMEZ	ALAN	BRIDGEPORT	CT	203/675-7181
1389	GOLDSTEIN	LEVI	NEW YORK	NY	718/384-9326
1905	GRAHAM	ALVIN	NEW YORK	NY	212/586-8815
1407	GREGORSKI	DANIEL	MT. VERNON	NY	914/468-1616
1114	GREENWALD	JANICE	NEW YORK	NY	212/588-1092
1410	HARRIS	CHARLES	STAMFORD	CT	203/781-0937
1439	HASENHAUER	CHRISTINA	BRIDGEPORT	CT	203/675-4987
1409	HAVELKA	RAYMOND	STAMFORD	CT	203/781-9697
1408	HENDERSON	WILLIAM	PRINCETON	NJ	201/812-4789
1121	HERNANDEZ	ROBERTO	NEW YORK	NY	718/384-3313
1991	HOWARD	GRETCHEN	BRIDGEPORT	CT	203/675-0007
1102	HERMANN	JOACHIM	WHITE PLAINS	NY	914/455-0976
1356	HOWARD	MICHAEL	NEW YORK	NY	212/586-8411
1545	HERRERO	CLYDE	STAMFORD	CT	203/781-1119
1292	HUNTER	HELEN	BRIDGEPORT	CT	203/675-4830
1440	JACKSON	LAURA	STAMFORD	CT	203/781-0088
1368	JEPSSEN	RONALD	STAMFORD	CT	203/781-8413
1369	JONSON	ANTHONY	NEW YORK	NY	212/587-5385
1411	JOHNSEN	JACK	PATERSON	NJ	201/732-3678
1113	JOHNSON	LESLIE	NEW YORK	NY	718/383-3003
1704	JONES	NATHAN	NEW YORK	NY	718/384-0049
1900	KING	WILLIAM	NEW YORK	NY	718/383-3698
1126	KIMANI	ANNE	NEW YORK	NY	212/586-1229
1677	KRAMER	JACKSON	BRIDGEPORT	CT	203/675-7432
1441	LAWRENCE	KATHY	PRINCETON	NJ	201/812-3337
1421	LEE	RUSSELL	MT. VERNON	NY	914/468-9143
1119	LI	JEFF	NEW YORK	NY	212/586-2344

1834	LEBLANC	RUSSELL	NEW YORK	NY	718/384-0040
1777	LUFKIN	ROY	NEW YORK	NY	718/383-4413
1663	MARKS	JOHN	NEW YORK	NY	212/587-7742
1106	MARSHBURN	JASPER	STAMFORD	CT	203/781-1457
1103	MCDANIEL	RONDA	NEW YORK	NY	212/586-0013
1477	MEYERS	PRESTON	BRIDGEPORT	CT	203/675-8125
1476	MONROE	JOYCE	STAMFORD	CT	203/781-2837
1379	MORGAN	ALFRED	STAMFORD	CT	203/781-2216
1104	MORGAN	CHRISTOPHER	NEW YORK	NY	718/383-9740
1009	MORGAN	GEORGE	NEW YORK	NY	212/586-7753
1412	MURPHEY	JOHN	PRINCETON	NJ	201/812-4414
1115	MURPHY	ALICE	NEW YORK	NY	718/384-1982
1128	NELSON	FELICIA	BRIDGEPORT	CT	203/675-1166
1442	NEWKIRK	SANDRA	PRINCETON	NJ	201/812-3331
1417	NEWKIRK	WILLIAM	PATERSON	NJ	201/732-6611
1478	NEWTON	JAMES	NEW YORK	NY	212/587-5549
1673	NICHOLLS	HENRY	STAMFORD	CT	203/781-7770
1839	NORRIS	DIANE	NEW YORK	NY	718/384-1767
1347	O'NEAL	BRYAN	NEW YORK	NY	718/384-0230
1423	OSWALD	LESLIE	MT. VERNON	NY	914/468-9171
1200	OVERMAN	MICHELLE	STAMFORD	CT	203/781-1835
1970	PARKER	ANNE	NEW YORK	NY	718/383-3895
1521	PARKER	JAY	NEW YORK	NY	212/587-7603
1354	PARKER	MARY	WHITE PLAINS	NY	914/455-2337
1424	PATTERSON	RENEE	NEW YORK	NY	212/587-8991
1132	PEARCE	CAROL	NEW YORK	NY	718/384-1986
1845	PEARSON	JAMES	NEW YORK	NY	718/384-2311
1556	PENNINGTON	MICHAEL	NEW YORK	NY	718/383-5681
1413	PETERS	RANDALL	PRINCETON	NJ	201/812-2478
1123	PETERSON	SUZANNE	NEW YORK	NY	718/383-0077
1907	PHELPS	WILLIAM	STAMFORD	CT	203/781-1118
1436	PORTER	SUSAN	NEW YORK	NY	718/383-5777
1385	RAYNOR	MILTON	BRIDGEPORT	CT	203/675-2846
1432	REED	MARILYN	MT. VERNON	NY	914/468-5454
1111	RHODES	JEREMY	PRINCETON	NJ	201/812-1837
1116	RICHARDS	CASEY	NEW YORK	NY	212/587-1224
1352	RIVERS	SIMON	NEW YORK	NY	718/383-3345
1555	RODRIGUEZ	JULIA	BRIDGEPORT	CT	203/675-2401
1038	RODRIGUEZ	MARIA	BRIDGEPORT	CT	203/675-2048
1420	ROUSE	JEREMY	PATERSON	NJ	201/732-9834
1561	SANDERS	RAYMOND	NEW YORK	NY	212/588-6615
1434	SANDERSON	EDITH	STAMFORD	CT	203/781-1333
1414	SANDERSON	NATHAN	BRIDGEPORT	CT	203/675-1715
1112	SANYERS	RANDY	NEW YORK	NY	718/384-4895
1390	SMART	JONATHAN	NEW YORK	NY	718/383-1141
1332	STEPHENSON	ADAM	BRIDGEPORT	CT	203/675-1497
1890	STEPHENSON	ROBERT	NEW YORK	NY	718/384-9874
1429	THOMPSON	ALICE	STAMFORD	CT	203/781-3857
1107	THOMPSON	WAYNE	NEW YORK	NY	718/384-3785
1908	TRENTON	MELISSA	NEW YORK	NY	212/586-6262
1830	TRIPP	KATHY	BRIDGEPORT	CT	203/675-2479
1882	TUCKER	ALAN	NEW YORK	NY	718/384-0216
1050	TUTTLE	THOMAS	WHITE PLAINS	NY	914/455-2119
1425	UNDERWOOD	JENNY	STAMFORD	CT	203/781-0978
1928	UPCHURCH	LARRY	WHITE PLAINS	NY	914/455-5009
1480	UPDIKE	THERESA	NEW YORK	NY	212/587-8729

1100	VANDEUSEN	RICHARD	NEW YORK	NY	212/586-2531
1995	VARNER	ELIZABETH	NEW YORK	NY	718/384-7113
1135	VEGA	ANNA	NEW YORK	NY	718/384-5913
1415	VEGA	FRANKLIN	NEW YORK	NY	718/384-2823
1076	VENTER	RANDALL	NEW YORK	NY	718/383-2321
1426	VICK	THERESA	PRINCETON	NJ	201/812-2424
1564	WALTERS	ANNE	NEW YORK	NY	212/587-3257
1221	WALTERS	DIANE	NEW YORK	NY	718/384-1918
1133	WANG	CHIN	NEW YORK	NY	212/587-1956
1435	WARD	ELAINE	NEW YORK	NY	718/383-4987
1418	WATSON	BERNARD	NEW YORK	NY	718/383-1298
1017	WELCH	DARIUS	NEW YORK	NY	212/586-5535
1443	WELLS	AGNES	STAMFORD	CT	203/781-5546
1131	WELLS	NADINE	NEW YORK	NY	718/383-1045
1427	WHALEY	CAROLYN	MT. VERNON	NY	914/468-4528
1036	WONG	LESLIE	NEW YORK	NY	212/587-2570
1130	WOOD	DEBORAH	NEW YORK	NY	212/587-0013
1127	WOOD	SANDRA	NEW YORK	NY	212/587-2881
1433	YANCEY	ROBIN	PRINCETON	NJ	201/812-1874
1431	YOUNG	DEBORAH	STAMFORD	CT	203/781-2987
1122	YOUNG	JOANN	NEW YORK	NY	718/384-2021
1105	YOUNG	LAWRENCE	NEW YORK	NY	718/384-0008

;

---

## PROCLIB.STAFF2

```
data proclib.staff;
 input Name & $16. IdNumber $ Salary
 Site $ HireDate date9.;
 format hiredate date9.;
 datalines;
Capalleti, Jimmy 2355 21163 BR1 30JAN2009
Chen, Len 5889 20976 BR1 18JUN2006
Davis, Brad 3878 19571 BR2 20MAR2004
Leung, Brenda 4409 34321 BR2 18SEP2014
Martinez, Maria 3985 49056 US2 10JAN2013
Orfali, Philip 0740 50092 US2 16FEB2003
Patel, Mary 2398 35182 BR3 02FEB2010
Smith, Robert 5162 40100 BR5 15APR2006
Sorrell, Joseph 4421 38760 US1 19JUN2011
Zook, Carla 7385 22988 BR3 18DEC2010
;
```

---

## PROCLIB.SUPERV

```
data proclib.superv;
 input supid $4. +8 state $2. +5 jobcat $2.;
 label supid='Supervisor Id' jobcat='Job Category';
 datalines;
1677 CT BC
1834 NY BC
1431 CT FA
1433 NJ FA
1983 NY FA
1385 CT ME
1420 NJ ME
1882 NY ME
1935 CT NA
1417 NJ NA
1352 NY NA
1106 CT PT
1442 NJ PT
1118 NY PT
1405 NJ SC
1564 NY SC
1639 CT TA
1401 NJ TA
1126 NY TA
;
```

---

## RADIO

This DATA step uses an INFILE statement to read data that is stored in an external file.

```
data radio;
 infile 'input-file' missover;
 input /(time1-time7) ($1. +1);
 listener=_n_;
run;
```

Here is the data that is stored in the external file:

```
967 32 f 5 3 5
7 5 5 5 7 0 0 0 8 7 0 0 8 0
781 30 f 2 3 5
5 0 0 0 5 0 0 0 4 7 5 0 0 0
859 39 f 1 0 5
```

```

1 0 0 0 1 0 0 0 0 0 0 0 0 0
859 40 f 6 1 5
7 5 0 5 7 0 0 0 0 0 0 5 0 0
467 37 m 2 3 1
1 5 5 5 5 4 4 8 8 0 0 0 0 0
220 35 f 3 1 7
7 0 0 0 7 0 0 0 7 0 0 0 0 0
833 42 m 2 2 4
7 0 0 0 7 5 4 7 4 0 1 4 4 0
967 39 f .5 1 7
7 0 0 0 7 7 0 0 0 0 0 0 8 0
677 28 m .5 .5 7
7 0 0 0 0 0 0 0 0 0 0 0 0 0
833 28 f 3 4 1
1 0 0 0 0 1 1 1 1 0 0 0 1 1
677 24 f 3 1 2
2 0 0 0 0 0 0 2 0 8 8 0 0 0
688 32 m 5 2 4
5 5 0 4 8 0 0 5 0 8 0 0 0 0
542 38 f 6 8 5
5 0 0 5 5 5 0 5 5 5 5 5 5 0
677 27 m 6 1 1
1 1 0 4 4 0 0 1 4 0 0 0 0 0
779 37 f 2.5 4 7
7 0 0 0 7 7 0 7 7 4 4 7 8 0
362 31 f 1 2 2
8 0 0 0 8 0 0 0 0 0 8 8 0 0
859 29 m 10 3 4
4 4 0 2 2 0 0 4 0 0 0 4 4 0
467 24 m 5 8 1
7 1 1 1 7 1 1 0 1 7 1 1 1 1
851 34 m 1 2 8
0 0 0 0 8 0 0 0 4 0 0 0 8 0
859 23 f 1 1 8
8 0 0 0 8 0 0 0 0 0 0 0 0 8
781 34 f 9 3 1
2 1 0 1 4 4 4 0 1 1 1 1 4 4
851 40 f 2 4 5
5 0 0 0 5 0 0 5 0 0 5 5 0 0
783 34 m 3 2 4
7 0 0 0 7 4 4 0 0 4 4 0 0 0
848 29 f 4 1.5 7
7 4 4 1 7 0 0 0 7 0 0 7 0 0
851 28 f 1 2 2
2 0 2 0 2 0 0 0 0 2 2 2 0 0
856 42 f 1.5 1 2
2 0 0 0 0 0 0 2 0 0 0 0 0 0
859 29 m .5 .5 5
5 0 0 0 1 0 0 0 0 0 8 8 5 0
833 29 m 1 3 2
2 0 0 0 2 2 0 0 4 2 0 2 0 0
859 23 f 10 3 1
1 5 0 8 8 1 4 0 1 1 1 1 1 4
781 37 f .5 2 7
7 0 0 0 1 0 0 0 1 7 0 1 0 0
833 31 f 5 4 1

```

1 0 0 0 1 0 0 0 4 0 4 0 0 0  
 942 23 f 4 2 1  
 1 0 0 0 1 0 1 0 1 1 0 0 0 0  
 848 33 f 5 4 1  
 1 1 0 1 1 0 0 0 1 1 1 0 0 0  
 222 33 f 2 0 1  
 1 0 0 0 1 0 0 0 0 0 0 0 0 0  
 851 45 f .5 1 8  
 8 0 0 0 8 0 0 0 0 0 8 0 0 0  
 848 27 f 2 4 1  
 1 0 0 0 1 1 0 0 4 1 1 1 1 1  
 781 38 m 2 2 1  
 5 0 0 0 1 0 0 0 0 0 1 1 0 0  
 222 27 f 3 1 2  
 2 0 2 0 2 2 0 0 2 0 0 0 0 0  
 467 34 f 2 2 1  
 1 0 0 0 0 1 0 1 0 0 0 0 1 0  
 833 27 f 8 8 1  
 7 0 1 0 7 4 0 0 1 1 1 4 1 0  
 677 49 f 1.5 0 8  
 8 0 8 0 8 0 0 0 0 0 0 0 0 0  
 849 43 m 1 4 1  
 1 0 0 0 4 0 0 0 4 0 1 0 0 0  
 467 28 m 2 1 7  
 7 0 0 0 7 0 0 7 0 0 1 0 0 0  
 732 29 f 1 0 2  
 2 0 0 0 2 0 0 0 0 0 0 0 0 0  
 851 31 m 2 2 2  
 2 5 0 6 0 0 8 0 2 2 8 2 0 0  
 779 42 f 8 2 2  
 7 2 0 2 7 0 0 0 0 0 0 0 2 0  
 493 40 m 1 3 3  
 3 0 0 0 5 3 0 5 5 0 0 0 1 1  
 859 30 m 1 0 7  
 7 0 0 0 7 0 0 0 0 0 0 0 0 0  
 833 36 m 4 2 5  
 7 5 0 5 0 5 0 0 7 0 0 0 5 0  
 467 30 f 1 4 1  
 0 0 0 0 1 0 6 0 0 1 1 1 0 6  
 859 32 f 3 5 2  
 2 2 2 2 2 2 6 6 2 2 2 2 2 6  
 851 43 f 8 1 5  
 7 5 5 5 0 0 0 4 0 0 0 0 0 0  
 848 29 f 3 5 1  
 7 0 0 0 7 1 0 0 1 1 1 1 1 0  
 833 25 f 2 4 5  
 7 0 0 0 5 7 0 0 7 5 0 0 5 0  
 783 33 f 8 3 8  
 8 0 8 0 7 0 0 0 8 0 5 4 0 5  
 222 26 f 10 2 1  
 1 1 0 1 1 0 0 0 3 1 1 0 0 0  
 222 23 f 3 2 2  
 2 2 2 2 7 0 0 2 2 0 0 0 0 0  
 859 50 f 1 5 4  
 7 0 0 0 7 0 0 5 4 4 4 7 0 0  
 833 26 f 3 2 1

```

1 0 0 1 1 0 0 5 5 0 1 0 0 0
467 29 m 7 2 1
1 1 1 1 1 0 0 1 1 1 0 0 0 0
859 35 m .5 2 2
7 0 0 0 2 0 0 7 5 0 0 4 0 0
833 33 f 3 3 6
7 0 0 0 6 8 0 8 0 0 0 8 6 0
221 36 f .5 1 5
0 7 0 0 0 7 0 0 7 0 0 7 7 0
220 32 f 2 4 5
5 0 5 0 5 5 5 0 5 5 5 5 5 5
684 19 f 2 4 2
0 2 0 2 0 0 0 0 0 2 2 0 0 0
493 55 f 1 0 5
5 0 0 5 0 0 0 0 7 0 0 0 0 0
221 27 m 1 1 7
7 0 0 0 0 0 0 0 5 0 0 0 5 0
684 19 f 0 .5 1
7 0 0 0 0 1 1 0 0 0 0 0 1 1
493 38 f .5 .5 5
0 8 0 0 5 0 0 0 5 0 0 0 0 0
221 26 f .5 2 1
0 1 0 0 0 1 0 0 5 5 5 1 0 0
684 18 m 1 .5 1
0 2 0 0 0 0 1 0 0 0 0 1 1 0
684 19 m 1 1 1
0 0 0 1 1 0 0 0 0 0 1 0 0 0
221 29 m .5 .5 5
0 0 0 0 0 5 5 0 0 0 0 0 5 5
683 18 f 2 4 8
0 0 0 0 8 0 0 0 8 8 8 0 0 0
966 23 f 1 2 1
1 5 5 5 1 0 0 0 0 1 0 0 1 0
493 25 f 3 5 7
7 0 0 0 7 2 0 0 7 0 2 7 7 0
683 18 f .5 .5 2
1 0 0 0 0 0 5 0 0 1 0 0 0 1
382 21 f 3 1 8
0 8 0 0 5 8 8 0 0 8 8 0 0 0
683 18 f 4 6 2
2 0 0 0 2 2 2 0 2 0 2 2 2 0
684 19 m .5 2 1
0 0 0 0 1 1 0 0 0 1 1 1 1 5
684 19 m 1.5 3.5 2
2 0 0 0 2 0 0 0 0 0 2 5 0 0
221 23 f 1 5 1
7 5 1 5 1 3 1 7 5 1 5 1 3 1
684 18 f 2 3 1
2 0 0 1 1 1 1 7 2 0 1 1 1 1
683 19 f 3 5 2
2 0 0 2 0 6 1 0 1 1 2 2 6 1
683 19 f 3 5 1
2 0 0 2 0 6 1 0 1 1 2 0 2 1
221 35 m 3 5 5
7 5 0 1 7 0 0 5 5 5 0 0 0 0
221 43 f 1 4 5

```



1 0 0 0 5 0 0 5 5 0 0 0 0 0  
 493 32 f 2 1 6  
 0 0 0 6 0 0 0 0 0 0 0 0 4 0  
 221 24 f 4 5 2  
 2 0 5 0 0 2 4 4 4 5 0 0 2 2  
 684 19 f 2 3 2  
 0 5 5 2 5 0 1 0 5 5 2 2 2 2  
 221 19 f 3 3 8  
 0 1 1 8 8 8 4 0 5 4 1 8 8 4  
 221 29 m 1 1 5  
 5 5 5 5 5 5 5 5 5 5 5 5 5 5  
 221 21 m 1 1 1  
 1 0 0 0 0 0 5 1 0 0 0 0 0 5  
 683 20 f 1 2 2  
 0 0 0 0 2 0 0 0 2 0 0 0 0 0  
 493 54 f 1 1 5  
 7 0 0 5 0 0 0 0 0 0 5 0 0 0  
 493 45 m 4 6 5  
 7 0 0 0 7 5 0 0 5 5 5 5 5 5  
 850 44 m 2.5 1.5 7  
 7 0 7 0 4 7 5 0 5 4 3 0 0 4  
 220 33 m 5 3 5  
 1 5 0 5 1 0 0 0 0 0 0 0 5 5  
 684 20 f 1.5 3 1  
 1 0 0 0 1 0 1 0 1 0 0 1 1 0  
 966 63 m 3 5 3  
 5 4 7 5 4 5 0 5 0 0 5 5 4 0  
 683 21 f 4 6 1  
 0 1 0 1 1 1 1 0 1 1 1 1 1 1  
 493 23 f 5 2 5  
 7 5 0 4 0 0 0 0 1 1 1 1 1 0  
 493 32 f 8 8 5  
 7 5 0 0 7 0 5 5 5 0 0 7 5 5  
 942 33 f 7 2 5  
 0 5 5 4 7 0 0 0 0 0 0 7 8 0  
 493 34 f .5 1 5  
 5 0 0 0 5 0 0 0 0 0 6 0 0 0  
 382 40 f 2 2 5  
 5 0 0 0 5 0 0 5 0 0 5 0 0 0  
 362 27 f 0 3 8  
 0 0 0 0 0 0 0 0 0 0 0 0 8 0  
 542 36 f 3 3 7  
 7 0 0 0 7 1 0 0 0 7 1 1 0 0  
 966 39 f 3 6 5  
 7 0 0 0 7 5 0 0 7 0 5 0 5 0  
 849 32 m 1 .5 7  
 7 0 0 0 5 0 0 0 7 4 4 5 7 0  
 677 52 f 3 2 3  
 7 0 0 0 0 7 0 0 0 7 0 0 3 0  
 222 25 m 2 4 1  
 1 0 0 0 1 0 0 0 1 0 1 0 0 0  
 732 42 f 3 2 7  
 7 0 0 0 1 7 5 5 7 0 0 3 4 0  
 467 26 f 4 4 1  
 7 0 1 0 7 1 0 0 7 7 4 7 0 0  
 467 38 m 2.5 0 1

```

1 0 0 0 1 0 0 0 0 0 0 0 0 0
382 37 f 1.5 .5 7
7 0 0 0 7 0 0 0 3 0 0 0 3 0
856 45 f 3 3 7
7 0 0 0 7 5 0 0 7 7 4 0 0 0
677 33 m 3 2 7
7 0 0 4 7 0 0 0 7 0 0 0 0 0
490 27 f .5 1 2
2 0 0 0 2 0 0 0 2 0 2 0 0 0
362 27 f 1.5 2 2
2 0 0 0 1 0 4 0 1 0 0 0 4 4
783 25 f 2 1 1
1 0 0 0 1 7 0 0 0 0 1 1 1 0
546 30 f 8 3 1
1 1 1 1 1 0 0 1 0 5 5 0 0 0
677 30 f 2 0 1
1 0 0 0 0 1 0 0 0 0 0 0 0 1
221 35 f 2 2 1
1 0 0 0 1 0 1 0 1 1 1 0 0 0
966 32 f 6 1 7
7 1 1 1 7 4 0 1 7 1 8 8 4 0
222 28 f 1 5 4
7 0 0 0 4 0 0 4 4 4 4 0 0 0
467 29 f 5 3 4
4 5 5 5 1 4 4 5 1 1 1 1 4 4
467 32 m 3 4 1
1 0 1 0 4 0 0 0 4 0 0 0 1 0
966 30 m 1.5 1 7
7 0 0 0 7 5 0 7 0 0 0 0 5 0
967 38 m 14 4 7
7 7 7 7 7 0 4 8 0 0 0 0 4 0
490 28 m 8 1 1
7 1 1 1 1 0 0 7 0 0 8 0 0 0
833 30 f .5 1 6
6 0 0 0 6 0 0 0 0 6 0 0 6 0
851 40 m 1 0 7
7 5 5 5 7 0 0 0 0 0 0 0 0 0
859 27 f 2 5 2
6 0 0 0 2 0 0 0 0 0 0 2 2 2
851 22 f 3 5 2
7 0 2 0 2 2 0 0 2 0 8 0 2 0
967 38 f 1 1.5 7
7 0 0 0 7 5 0 7 4 0 0 7 5 0
856 34 f 1.5 1 1
0 1 0 0 0 1 0 0 4 0 0 0 0 0
222 33 m .1 .1 7
7 0 0 0 7 0 0 0 0 0 7 0 0 0
856 22 m .50 .25 1
0 1 0 0 1 0 0 0 0 0 0 0 0 0
677 30 f 2 2 4
1 0 4 0 4 0 0 0 4 0 0 0 0 0
859 25 m 2 3 7
0 0 0 0 0 7 0 0 7 0 2 0 0 1
833 35 m 2 6 7
7 0 0 0 7 1 1 0 4 7 4 7 1 1
677 35 m 10 4 1

```

1 1 1 1 1 8 6 8 1 0 0 8 8 8  
848 29 f 5 3 8  
8 0 0 0 8 8 0 0 0 8 8 8 0 0  
688 26 m 3 1 1  
1 1 7 1 1 7 0 0 0 8 8 0 0 0  
490 41 m 2 2 5  
5 0 0 0 0 0 5 5 0 0 0 0 0 5  
493 35 m 4 4 7  
7 5 0 5 7 0 0 7 7 7 7 0 0 0  
677 27 m 15 11 1  
1 1 1 1 1 1 1 1 1 1 1 1 1 1  
848 27 f 3 5 1  
1 1 0 0 1 1 0 0 1 1 1 1 0 0  
362 30 f 1 0 1  
1 0 0 0 7 5 0 0 0 0 0 0 0 0  
783 29 f 1 1 4  
4 0 0 0 4 0 0 0 4 0 0 0 4 0  
467 39 f .5 2 4  
7 0 4 0 4 4 0 0 4 4 4 4 4 4  
677 27 m 2 2 7  
7 0 0 0 7 0 0 7 7 0 0 7 0 0  
221 23 f 2.5 1 1  
1 0 0 0 1 0 0 0 0 0 0 0 0 0  
677 29 f 1 1 7  
0 0 0 0 7 0 0 0 7 0 0 0 0 0  
783 32 m 1 2 5  
4 5 5 5 4 2 0 0 0 0 3 2 2 0  
833 25 f 1 0 1  
1 1 0 0 0 0 0 0 0 0 0 0 0 0  
859 24 f 7 3 7  
1 0 0 0 1 0 0 0 0 1 0 0 1 0  
677 29 m 2 2 8  
0 8 8 0 8 0 0 0 8 8 8 0 0 0  
688 31 m 8 2 5  
7 5 5 5 5 7 0 0 7 7 0 0 0 0  
856 31 m 9 4 1  
1 1 1 1 1 0 0 0 0 0 0 0 1 0  
856 44 f 1 0 6  
6 0 0 0 6 0 0 0 0 0 0 0 0 0  
677 37 f 3 3 1  
0 0 1 0 0 0 0 0 4 4 0 0 0 0  
859 27 m 2 .5 2  
2 2 2 2 2 2 2 2 0 0 0 0 0 2  
781 30 f 10 4 2  
2 0 0 0 2 0 2 0 0 0 0 0 0 2  
362 27 m 12 4 3  
3 1 1 1 1 3 3 3 0 0 0 0 3 0  
362 33 f 2 4 1  
1 0 0 0 7 0 0 7 1 1 1 1 1 0  
222 26 f 8 1 1  
1 1 1 1 0 0 0 1 0 0 0 0 0 0  
779 37 f 6 3 1  
1 1 1 1 1 0 0 1 1 0 0 0 1 0  
467 32 f 1 1 2  
2 0 0 0 0 0 0 0 2 0 0 2 0 0  
859 23 m 1 1 1

```

1 0 0 0 1 1 0 1 0 0 0 0 1 1
781 33 f 1 .5 6
6 0 0 0 6 0 0 0 0 0 0 0 0 0
779 28 m 5 2 1
1 1 1 1 1 0 0 0 0 7 7 1 1 0
677 28 m 3 1 5
7 5 5 5 5 6 0 0 6 6 6 6 6 0
677 25 f 9 2 5
1 5 5 5 5 1 1 0 1 1 1 1 1 1
848 30 f 6 2 8
8 0 0 0 2 7 0 0 0 0 2 0 2 0
546 36 f 4 6 4
7 0 0 0 4 4 0 5 5 5 5 2 4 4
222 30 f 2 3 2
2 2 0 0 2 0 0 0 2 0 2 2 0 0
383 32 m 4 1 2
2 0 0 0 2 0 0 2 0 0 0 0 0 0
851 43 f 8 1 6
4 6 0 6 4 0 0 0 0 0 0 0 0 0
222 27 f 1 3 1
1 1 0 1 1 1 0 0 1 0 0 0 4 0
833 22 f 1.5 2 1
1 0 0 0 1 1 0 0 1 1 1 0 0 0
467 29 f 2 1 8
8 0 8 0 8 0 0 0 0 0 8 0 0 0
856 28 f 2 3 1
1 0 0 0 1 0 0 0 1 0 0 1 0 0
580 31 f 2.5 2.5 6
6 6 6 6 6 6 6 6 1 1 1 1 6 6
688 39 f 8 8 3
3 3 3 3 3 3 3 3 3 3 3 3 3 3
677 37 f 1.5 .5 1
6 1 1 1 6 6 0 0 1 1 6 6 6 0
859 38 m 3 6 3
7 0 0 0 7 3 0 0 3 0 3 0 0 0
677 25 f 7 1 1
0 1 1 1 2 0 0 0 1 2 1 1 1 0
848 36 f 7 1 1
0 1 0 1 1 0 0 0 0 0 0 1 1 0
781 31 f 2 4 1
1 0 0 0 1 1 0 1 1 1 1 1 0 0
781 40 f 2 2 8
8 0 0 8 8 0 0 0 0 0 8 8 0 0
677 25 f 3 5 1
1 6 1 6 6 3 0 0 2 2 1 1 1 1
779 33 f 3 2 1
1 0 1 0 0 0 1 0 1 0 0 0 1 0
677 25 m 7 1.5 1
1 1 0 1 1 0 0 0 0 0 1 0 0 0
362 35 f .5 0 1
1 0 0 0 1 0 0 0 0 0 0 0 0 0
677 41 f 6 2 7
7 7 0 7 7 0 0 0 0 0 8 0 0 0
677 24 m 5 1 5
1 5 0 5 0 0 0 0 1 0 0 0 0 0
833 29 f .5 0 6

```

6 0 0 0 6 0 0 0 0 0 0 0 0 0  
 362 30 f 1 1 1  
 1 0 0 0 1 0 0 0 1 0 0 0 0 0  
 850 26 f 6 12 6  
 6 0 0 0 2 2 2 6 6 6 0 0 6 6  
 467 25 f 2 3 1  
 1 0 0 6 1 1 0 0 0 0 1 1 1 1  
 967 29 f 1 2 7  
 7 0 0 0 7 0 0 7 7 0 0 0 0 0  
 833 31 f 1 1 7  
 7 0 7 0 7 3 0 0 3 3 0 0 0 0  
 859 40 f 7 1 5  
 1 5 0 5 5 1 0 0 1 0 0 0 0 0  
 848 31 m 1 2 1  
 1 0 0 0 1 1 0 0 4 4 1 4 0 0  
 222 32 f 2 3 3  
 3 0 0 0 0 7 0 0 3 0 8 0 0 0  
 783 33 f 2 0 4  
 7 0 0 0 7 0 0 0 4 0 4 0 0 0  
 856 28 f 8 4 2  
 0 2 0 2 2 0 0 0 2 0 2 0 4 0  
 781 30 f 3 5 1  
 1 1 1 1 1 1 0 0 1 1 1 1 1 0  
 850 25 f 6 3 1  
 7 5 0 5 7 1 0 0 7 0 1 0 1 0  
 580 33 f 2.5 4 2  
 2 0 0 0 2 0 0 0 0 0 8 8 0 0  
 677 38 f 3 3 1  
 1 0 0 0 1 0 1 1 1 0 1 0 0 4  
 677 26 f 2 2 1  
 1 0 1 0 1 0 0 0 1 1 1 0 0 0  
 467 52 f 3 2 2  
 2 6 6 6 6 2 0 0 2 2 2 2 0 0  
 542 31 f 1 3 1  
 1 0 1 0 1 0 0 0 1 1 1 1 1 0  
 859 50 f 9 3 6  
 6 6 6 6 6 6 6 6 6 3 3 3 6 6  
 779 26 f 1 2 1  
 7 0 1 0 1 1 4 1 4 1 1 1 4 4  
 779 36 m 1.5 2 4  
 1 4 0 4 4 0 0 4 4 4 4 0 0 0  
 222 31 f 0 3 7  
 1 0 0 0 7 0 0 0 0 0 0 0 0 0  
 362 27 f 1 1 1  
 1 0 1 0 1 4 0 4 4 1 0 4 4 0  
 967 32 f 3 2 7  
 7 0 0 0 7 0 0 0 1 0 0 1 0 0  
 362 29 f 10 2 2  
 2 2 2 2 2 2 2 2 2 2 7 0 0  
 677 27 f 3 4 1  
 0 5 1 1 0 5 0 0 0 1 1 1 0 0  
 546 32 m 5 .5 8  
 8 0 0 0 8 0 0 0 8 0 0 0 0 0  
 688 38 m 2 3 2  
 2 0 0 0 2 0 0 0 2 0 0 0 1 0  
 362 28 f 1 1 1

```

1 0 0 0 1 1 0 4 0 0 0 0 4 0
851 32 f .5 2 4
5 0 0 0 4 0 0 0 0 0 0 0 2 0
967 43 f 2 2 1
1 0 0 0 1 0 0 1 7 0 0 0 1 0
467 44 f 10 4 6
7 6 0 6 6 0 6 0 0 0 0 0 0 6
467 23 f 5 3 1
0 2 1 2 1 0 0 0 1 1 1 1 1 1
783 30 f 1 .5 1
1 0 0 0 1 0 0 0 0 0 0 7 0 0
677 29 f 3 1 2
2 2 2 2 2 0 0 0 0 0 0 0 0 0
859 26 f 9.5 1.5 2
2 2 2 2 2 0 0 2 2 0 0 0 0 0
222 28 f 3 0 2
2 0 0 0 2 0 0 0 0 0 2 0 0 0
966 37 m 2 1 1
7 1 1 1 7 0 0 0 7 0 0 0 0 0
859 31 f 10 10 1
0 1 1 1 1 0 0 0 1 1 0 0 1 0
781 27 f 2 1 2
2 0 0 0 1 0 0 0 4 0 0 0 0 0
677 31 f .5 .5 6
7 0 0 0 0 0 0 0 6 0 0 0 0 0
848 28 f 5 1 2
2 2 0 2 0 0 0 0 2 0 0 0 0 0
781 24 f 3 3 6
1 6 6 6 1 6 0 0 0 0 1 0 1 1
856 27 f 1.5 1 6
2 6 6 6 2 5 0 2 0 0 5 2 0 0
382 30 m 1 2 7
7 0 0 0 7 0 4 7 0 0 0 7 4 4
848 25 f 9 3 1
7 1 1 5 1 0 0 0 1 1 1 1 1 0
382 30 m 1 2 4
7 0 0 0 7 0 4 7 0 0 0 7 4 4
688 40 m 2 3 1
1 0 0 0 1 3 1 0 5 0 4 4 7 1
856 40 f .5 5 5
3 0 0 0 3 0 0 0 0 0 5 5 0 0
966 25 f 2 .5 2
1 0 0 0 2 6 0 0 4 0 0 0 0 0
859 30 f 2 4 2
2 0 0 0 0 2 0 0 0 0 2 0 0 0
849 29 m 10 1 5
7 5 5 5 7 5 5 0 0 0 0 0 7 0
781 28 m 1.5 3 4
1 0 0 0 1 4 4 0 4 4 1 1 4 0
467 35 f 4 2 6
7 6 7 6 6 7 6 7 7 7 7 7 6
222 32 f 10 5 1
1 1 0 1 1 0 0 1 1 1 0 0 1 0
677 32 f 1 0 1
1 0 1 0 0 0 0 0 0 0 0 0 0 0
222 54 f 21 4 3

```

5 0 0 0 7 0 0 7 0 0 0 0 0 0  
 677 30 m 4 6 1  
 7 0 0 0 0 1 1 1 7 1 1 0 8 1  
 683 29 f 1 2 8  
 8 0 0 0 8 0 0 0 0 8 8 0 0 0  
 467 38 m 3 5 1  
 1 0 0 0 1 0 0 1 1 0 0 0 0 0  
 781 29 f 2 3 8  
 8 0 0 0 8 8 0 0 8 8 0 8 8 0  
 781 30 f 1 0 5  
 5 0 0 0 0 5 0 0 0 0 0 0 0 0  
 783 40 f 1.5 3 1  
 1 0 0 0 1 4 0 0 1 1 1 0 0 0  
 851 30 f 1 1 6  
 6 0 0 0 6 0 0 0 6 0 0 6 0 0  
 851 40 f 1 1 5  
 5 0 0 0 5 0 0 0 0 1 0 0 0 0  
 779 40 f 1 0 2  
 2 0 0 0 2 0 0 0 0 0 0 0 0 0  
 467 37 f 4 8 1  
 1 0 0 0 1 0 3 0 3 1 1 1 0 0  
 859 37 f 4 3 3  
 0 3 7 0 0 7 0 0 0 7 8 3 7 0  
 781 26 f 4 1 2  
 2 2 0 2 1 0 0 0 2 0 0 0 0 0  
 859 23 f 8 3 3  
 3 2 0 2 3 0 0 0 1 0 0 3 0 0  
 967 31 f .5 0 1  
 1 0 0 0 0 0 0 0 0 0 0 0 0 0  
 851 38 m 4 2 5  
 7 5 0 5 4 0 4 7 7 0 4 0 8 0  
 467 30 m 2 1 2  
 2 2 0 2 0 0 0 0 2 0 2 0 0 0  
 848 33 f 2 2 7  
 7 0 0 0 0 7 0 7 7 0 0 0 7 0  
 688 35 f 5 8 3  
 2 2 2 2 2 0 0 3 3 3 3 3 0 0  
 467 27 f 2 3 1  
 1 0 1 0 0 1 0 0 1 1 1 0 0 0  
 783 42 f 3 1 1  
 1 0 0 0 1 0 0 0 1 0 1 1 0 0  
 687 40 m 1.5 2 1  
 7 0 0 0 1 1 0 0 1 0 7 0 1 0  
 779 30 f 4 8 7  
 7 0 0 0 7 0 6 7 4 2 2 0 0 6  
 222 34 f 9 0 8  
 8 2 0 2 8 0 0 0 0 0 0 0 0 0  
 467 28 m 3 1 2  
 2 0 0 0 2 2 0 0 0 2 2 0 0 0  
 222 28 f 8 4 2  
 1 2 1 2 2 0 0 1 2 2 0 0 2 0  
 542 35 m 2 3 2  
 6 0 7 0 7 0 7 0 0 0 2 2 0 0  
 677 31 m 12 4 3  
 7 3 0 3 3 4 0 0 4 4 4 0 0 0  
 783 45 f 1.5 2 6

```

6 0 0 0 6 0 0 6 6 0 0 0 0 0
942 34 f 1 .5 4
4 0 0 0 1 0 0 0 0 0 2 0 0 0
222 30 f 8 4 1
1 1 1 1 1 0 0 0 1 1 0 0 0 0
967 38 f 1.5 2 7
7 0 0 0 7 0 0 7 1 1 1 1 0 0
783 37 f 2 1 1
6 6 1 1 6 6 0 0 6 1 1 1 6 0
467 31 f 1.5 2 2
2 0 7 0 7 0 0 7 7 0 0 0 7 0
859 48 f 3 0 7
7 0 0 0 0 0 0 0 0 7 0 0 0 0
490 35 f 1 1 7
7 0 0 0 7 0 0 0 0 0 0 0 8 0
222 27 f 3 2 3
8 0 0 0 3 8 0 3 3 0 0 0 0 0
382 36 m 3 2 4
7 0 5 4 7 4 4 0 7 7 4 7 0 4
859 37 f 1 1 2
7 0 0 0 0 2 0 2 2 0 0 0 0 2
856 29 f 3 1 1
1 0 0 0 1 1 1 1 0 0 1 1 0 1
542 32 m 3 3 7
7 0 0 0 0 7 7 7 0 0 0 0 7 7
783 31 m 1 1 1
1 0 0 0 1 0 0 0 1 1 1 0 0 0
833 35 m 1 1 1
5 4 1 5 1 0 0 1 1 0 0 0 0 0
782 38 m 30 8 5
7 5 5 5 5 0 0 4 4 4 4 4 0 0
222 33 m 3 3 1
1 1 1 1 1 1 1 1 4 1 1 1 1 1
467 24 f 2 4 1
0 0 1 0 1 0 0 0 1 1 1 0 0 0
467 34 f 1 1 1
1 0 0 0 1 0 0 1 1 0 0 0 0 0
781 53 f 2 1 5
5 0 0 0 5 5 0 0 0 0 5 5 5 0
222 30 m 2 5 3
6 3 3 3 6 0 0 0 3 3 3 3 0 0
688 26 f 2 2 1
1 0 0 0 1 0 0 0 1 0 1 1 0 0
222 29 m 8 5 1
1 6 0 6 1 0 0 1 1 1 1 0 0 0
783 33 m 1 2 7
7 0 0 0 7 0 0 0 7 0 0 0 7 0
781 39 m 1.5 2.5 2
2 0 2 0 2 0 0 0 2 2 2 0 0 0
850 22 f 2 1 1
1 0 0 0 1 1 1 0 5 0 0 1 0 0
493 36 f 1 0 5
0 0 0 0 7 0 0 0 0 0 0 0 0 0
967 46 f 2 4 7
7 5 0 5 7 0 0 0 4 7 4 0 0 0
856 41 m 2 2 4

```



7 4 0 0 7 4 0 4 0 0 0 7 0 0  
 546 25 m 5 5 8  
 8 8 0 0 0 0 0 0 0 0 0 0 0 0  
 222 27 f 4 4 3  
 2 2 2 3 7 7 0 2 2 2 3 3 3 0  
 688 23 m 9 3 3  
 3 3 3 3 3 7 0 0 3 0 0 0 0 0  
 849 26 m .5 .5 8  
 8 0 0 0 8 0 0 0 0 8 0 0 0 0  
 783 29 f 3 3 1  
 1 0 0 0 4 0 0 4 1 0 1 0 0 0  
 856 34 f 1.5 2 1  
 7 0 0 0 7 0 0 7 4 0 0 7 0 0  
 966 33 m 3 5 4  
 7 0 0 0 7 4 5 0 7 0 0 7 4 4  
 493 34 f 2 5 1  
 1 0 0 0 1 0 0 0 7 0 1 1 8 0  
 467 29 m 2 4 2  
 2 0 0 0 2 0 0 2 2 2 2 2 2 2  
 677 28 f 1 4 1  
 1 1 1 1 1 0 0 0 1 0 1 0 0 0  
 781 27 m 2 2 1  
 1 0 1 0 4 2 4 0 2 2 1 0 1 4  
 467 24 m 4 4 1  
 7 1 0 1 1 1 0 7 1 0 0 0 0 0  
 859 26 m 5 5 1  
 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
 848 27 m 7 2 5  
 7 5 0 5 4 5 0 0 0 7 4 4 0 4  
 677 25 f 1 2 8  
 8 0 0 0 0 5 0 0 8 0 0 0 2 0  
 222 26 f 3.5 0 2  
 2 0 0 0 2 0 0 0 0 0 0 0 0 0  
 833 32 m 1 2 1  
 1 0 0 0 1 0 0 0 5 0 1 0 0 0  
 781 28 m 2 .5 7  
 7 0 0 0 7 0 0 0 4 0 0 0 0 0  
 783 28 f 1 1 1  
 1 0 0 0 1 0 0 0 0 0 1 1 0 0  
 222 28 f 5 5 2  
 2 6 6 2 2 0 0 0 2 2 0 0 2 2  
 851 33 m 4 5 3  
 1 0 0 0 7 3 0 3 3 3 3 3 7 5  
 859 39 m 2 1 1  
 1 0 0 0 1 0 0 0 0 0 0 1 0 0  
 848 45 m 2 2 7  
 7 0 0 0 7 0 0 0 7 0 0 0 0 0  
 467 37 m 2 2 7  
 7 0 0 0 0 7 0 0 0 7 0 0 7 0  
 859 32 m .25 .25 1  
 1 0 0 0 0 0 0 0 1 0 0 0 0 0

---

# SALES

```
data sales;
 input Region $ CitySize $ Population Product $ SaleType $ Units NetSales;
 cards;
NC S 25000 A100 R 150 3750.00
NC M 125000 A100 R 350 8650.00
NC L 837000 A100 R 800 20000.00
NC S 25000 A100 W 150 3000.00
NC M 125000 A100 W 350 7000.00
NC M 625000 A100 W 750 15000.00
TX M 227000 A100 W 350 7250.00
TX L 5000 A100 W 750 5000.00
;
```

# Appendix 3

## UNICODE License

---

UNICODE LICENSE V3 ..... 2549

---

## UNICODE LICENSE V3

Copyright © 1991-2024 Unicode, Inc.

Copyright (c) 1995-2015 International Business Machines Corporation and others

NOTICE TO USER: Carefully read the following legal agreement. BY DOWNLOADING, INSTALLING, COPYING OR OTHERWISE USING DATA FILES, AND/OR SOFTWARE, YOU UNEQUIVOCALLY ACCEPT, AND AGREE TO BE BOUND BY, ALL OF THE TERMS AND CONDITIONS OF THIS AGREEMENT. IF YOU DO NOT AGREE, DO NOT DOWNLOAD, INSTALL, COPY, DISTRIBUTE OR USE THE DATA FILES OR SOFTWARE.

Permission is hereby granted, free of charge, to any person obtaining a copy of data files and any associated documentation (the "Data Files") or software and any associated documentation (the "Software") to deal in the Data Files or Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Data Files or Software, and to permit persons to whom the Data Files or Software are furnished to do so, provided that either (a) this copyright and permission notice appear with all copies of the Data Files or Software, or (b) this copyright and permission notice appear in associated Documentation.

THE DATA FILES AND SOFTWARE ARE PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS.

IN NO EVENT SHALL THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING

FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THE DATA FILES OR SOFTWARE.

Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in these Data Files or Software without prior written authorization of the copyright holder.

---