THE POWER TO KNOW.

Base SAS® 9.3 Procedures Guide

Second Edition

SAS® Documentation

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2012. *Base SAS® 9.3 Procedures Guide, Second Edition*. Cary, NC: SAS Institute Inc.

Base SAS® 9.3 Procedures Guide, Second Edition

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st electronic book, August 2012
2nd electronic book, March 2014

SAS® Publishing provides a complete selection of books and electronic products to help customers use SAS software to its fullest potential. For more information about our e-books, e-learning products, CDs, and hard-copy books, visit the SAS Publishing Web site at **support.sas.com/publishing** or call 1-800-727-3228.

Contents

About This Book

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 (=).

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 the following examples of SAS syntax, the keywords are the first words in the syntax:

CHAR (*string, position*)

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

ALTER (*alter-password*)

BEST *w*.

REMOVE <*data-set-name*>

In the following 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**

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 between angle brackets.

In the following 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 the following 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 the following example, *string* and *substring* are required arguments, while *modifiers* and *startpos* are optional.

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

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 the following example, the keyword ERROR is written in uppercase bold:

ERROR<*message*>;

UPPERCASE
identifies arguments that are literals.

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

CMPMODEL = BOTH | CATALOG | XML

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

- nonliteral arguments In the following example of the LINK statement, the argument *label* is a user-supplied value and is therefore written in italics:

  LINK *label*;

- nonliteral values that are assigned to an argument

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

  FORMAT = *variable-1* <, ..., *variable-nformat*><DEFAULT = *default-format*>;

Items in italics can also be the generic name for a list of arguments from which you can choose (for example, *attribute-list*). If more than one of an item in italics can be used, the items are expressed as *item-1, ..., item-n*.

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

  MAPS = *location-of-maps*

< >
: angle brackets identify optional arguments. Any argument that is not enclosed in angle brackets is required.

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

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

|
: 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 the following example of the CMPMODEL= system option, you can choose only one of the arguments:

  CMPMODEL = BOTH | CATALOG | XML

...
: an ellipsis indicates that the argument or group of arguments following the ellipsis can be repeated. If the ellipsis and the following argument are enclosed in angle brackets, then the argument is optional.

  In the following example of the CAT function, the ellipsis indicates that you can have multiple optional items:

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

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

  In the following 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 the following 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 usually have a choice of using a SAS statement (LIBNAME or FILENAME) or the operating environment's control language to make the association. 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*. Note that *SAS-library* is enclosed in quotation marks:

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

What's New in Base SAS 9.3 Procedures

Overview

The following procedures are new:

- PROC GROOVY
- PROC HADOOP
- PROC QDEVICE

The following Base SAS procedures have been enhanced:

PROC CIMPORT	PROC PWENCODE
PROC COPY	PROC RANK
PROC CPORT	PROC REGISTRY
PROC DATASETS	PROC SCAPROC
PROC FCMP	PROC SOAP
PROC FORMAT	PROC SORT
PROC OPTIONS	PROC REPORT
PROC PRINT	PROC TABULATE
PROC PRINTTO	PROC XSL

New Base SAS Procedures

The GROOVY Procedure

The GROOVY procedure 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. For more information, see Chapter 25, "GROOVY Procedure," on page 717.

The HADOOP Procedure

The HADOOP procedure is new for the second maintenance release for SAS 9.3. PROC HADOOP enables SAS to run Apache Hadoop code against Hadoop data. Using PROC HADOOP, you can submit Hadoop Distributed File System (HDFS) commands,

MapReduce programs, and Pig language code against Hadoop data. For more information, see Chapter 26, "HADOOP Procedure," on page 727.

The QDEVICE Procedure

The QDEVICE procedure creates reports about graphics devices and universal printers that summarizes information, including color support, default output sizes, margin sizes, resolution, supported fonts, hardware symbols, hardware fill types, hardware line styles, and device options. Output from this procedure can be sent to the SAS log or to an output SAS data set. For more information, see Chapter 43, "QDEVICE Procedure," on page 1141.

Enhanced Base SAS Procedures

The CIMPORT Procedure

The following enhancements have been made to the CIMPORT procedure:

- New option UPCASE is added to the CIMPORT procedure. This option is supported only for Double-Byte Character Sets (DBCS). For more information, see "UPCASE" on page 256.
- The CIMPORT SELECT and EXCLUDE statements will now support case sensitive names for files and catalogs in SAS/ACCESS engine libraries.
- The CIMPORT procedure now supports SAS name literals that include embedded blanks. For more information, see "SAS Name Literals" in Chapter 3 of *SAS Language Reference: Concepts*.
- When VALIDVARNAME=ANY or VALIDMEMNAME=EXTEND are specified, the data set names or member names used with the CIMPORT procedure can now be up to 32 bytes in length. Names and member names can also be mixed case. For more information, see VALIDMEMNAME= and VALIDVARNAME= .

For more information, see CIMPORT Procedure on page 251.

The COPY Procedure

The OVERRIDE option was added to the COPY procedure. The OVERRIDE option overrides specified output data set options copied from the input data set. For more information, see "OVERRIDE=(ds_option-1=value-1 <...ds_option-n=value-n>)" on page 410.

The CPORT Procedure

The following enhancements have been made to the CPORT procedure:

- The CPORT SELECT and EXCLUDE statements will now support case sensitive names for files and catalogs in SAS/ACCESS engine libraries.
- The CPORT procedure now supports SAS name literals that include embedded blanks. For more information, see "SAS Name Literals" in Chapter 3 of *SAS Language Reference: Concepts* for details.

- When VALIDVARNAME=ANY or VALIDMEMNAME=EXTEND are specified, the data set names or member names used with the CPORT procedure can now be up to 32 bytes in length. Names and member names can also be mixed case. For more information, see VALIDMEMNAME= and VALIDVARNAME= for details.

For more information, see CPORT Procedure on page 349.

The DATASETS Procedure

The OVERRIDE option for the COPY statement was added to the DATASETS procedure. The OVERRIDE option overrides specified output data set options copied from the input data set. For more information, see "OVERRIDE=(ds_option-1=value-1 <...ds_option-n=value-n>)" on page 410.

The FCMP Procedure

The following functions have been added to the FCMP procedure:

INVCDF
: This function computes the quantile from any distribution for which you have defined a cumulative distribution function (CDF).

LIMMOMENT
: This function computes the limited moment of any distribution for which you have defined a cumulative distribution function (CDF).

For more information, see the "INVCDF Function" on page 576 and the "LIMMOMENT Function" on page 580.

The FORMAT Procedure

The following enhancements have been made to the FORMAT procedure:

- You can create a format catalog that corresponds to the current SAS locale by specifying the LOCALE= option in the PROC FORMAT statement.
- A user-defined format or informat that defines a missing value supersedes a value specified by the MISSING system option.
- The maximum number of labels that can be used for the MULTILABEL option is 255.
- The PICTURE statement directive %n formats the number of days in a duration.
- The PICTURE statement directive %s formats fractional seconds.
- The PICTURE statement directive %z formats a UTC time-zone offset.
- The PICTURE statement directive %Z formats a time-zone name.
- Use the VALUE= statement to create a format that performs a function on a value.
- Use can use SAS Explorer to view format and informat definitions.

For more information, see Chapter 23, "FORMAT Procedure," on page 625.

The OPTIONS Procedure

The following enhancements have been made to the OPTIONS procedure

- These PROC OPTIONS statement options are new:

LISTINSERTAPPEND
This option lists the system options whose value can be modified by the INSERT and APPEND system options.

LISTRESTRICT
This option lists the system options that can be restricted by your site administrator.

- These PROC OPTIONS statement options have been enhanced:

  DEFINE
  Valid values for an option now display in the SAS log when you specify the DEFINE option.

  OPTION=
  The OPTION= option now accepts one or more options.

  VALUE
  If the option was set by a configuration file, the name of the configuration file that set the option now displays in the SAS log when you specify the VALUE option.

For more information, see Chapter 32, "OPTIONS Procedure," on page 863.

The PRINT Procedure

The following enhancements have been made to the PRINT procedure:

- The PRINT procedure is now completely integrated with the Output Delivery System.
- When you specify the BLANKLINE option, each BY group is a separate table. The count of the number of observations to output between each blank line is reset to zero at the beginning of each BY group.
- Bylines can be up to 512 characters.
- For all destinations other than the LISTING destination, if HEADING=V, the size of the column label is no longer restricted by the page size specified for the LISTING destination.
- For the LISTING destination, if HEADING=V, the variable name is used in place of a label if the column heading is too long for the page.
- ROWS= is valid only for the LISTING destination.
- If you specify a BY variable whose values are not sorted, SAS stops printing at the observation for the unsorted value and writes a message to the log. In previous releases of SAS, SAS would stop printing at the observation preceding the unsorted value.
- If the PRINT procedure errors or terminates, output might be produced where previously there was none.

For more information, see Chapter 37, "PRINT Procedure," on page 995.

The PRINTTO Procedure

The following enhancements have been made to the PRINTTO procedure:

- If you use the PRINTTO procedure to write to a file or a catalog entry, you must open the LISTING destination.

- If SAS is started in objectserver mode, the PRINTTO procedure does not route log messages to the log specified by the ALTLOG= system option.

For more information, see Chapter 38, "PRINTTO Procedure," on page 1073.

The PWENCODE Procedure

The following enhancements have been made to the PWENCODE procedure:

- 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.

- If the METHOD= option is omitted, the default encoding method is used. When the FIPS 140-2 compliance option, -encryptfips, is specified, the encoding default method is **`sas003`**. For all other cases, encoding method **`sas002`** is the default method used.

For more information, see Chapter 42, "PWENCODE Procedure," on page 1133.

The RANK Procedure

The following enhancements have been made to the RANK procedure:

- There is now in-database support for the Netezza database management system.
- The SQL_IP_TRACE option shows the generated SQL that PROC RANK generates.
- The PRESERVERAWBYVALUES option preserves the raw values of the BY variable.

For more information, see "In-Database Processing for PROC RANK" on page 1185.

The REGISTRY Procedure

The following two options are new:

FOLLOWLINKS
: The FOLLOWLINKS option follows links that are found when processing the LIST command.

KEYSONLY
: The KEYSONLY option limits the LIST, LISTUSER, LISTHELP, and LISTREG options output to display keys only.

For more information, see Chapter 45, "REGISTRY Procedure," on page 1203.

The REPORT Procedure

The MLF option has been added to the DEFINE statement in PROC REPORT. For more information, see the MLF option on page 1284 in Chapter 46, "REPORT Procedure," on page 1220.

The SCAPROC Procedure

The RECORD statement has added support for the EXPANDMACROS, INHERITLIB, and NOOPTIMZE arguments. For more information, see Chapter 48, "SCAPROC Procedure," on page 1401.

The SOAP Procedure

You can now call a SAS secured service without providing credentials. However, a connection to the metadata server is required. For more information, see Chapter 49, "SOAP Procedure," on page 1411.

The SORT Procedure

The following enhancements have been made to the SORT procedure:

- There is now in-database support for the NETEZZA database management system.
- The new NOUNIQUEKEY and UNIQUEOUT= options have been added. For more information, see "NOUNIQUEKEY" on page 1440 and UNIQUEOUT= on page 1443.

For more information, see Chapter 50, "SORT Procedure," on page 1425.

The TABULATE Procedure

The NOCELLMERGE option has been added to the TABLE statement in PROC TABULATE. For more information, see the NOCELLMERGE on page 1519 option in the TABLE statement of PROC TABULATE on page 1516.

The XSL Procedure

For the second maintenance release for SAS 9.3, the XSL procedure uses the Saxon-EE version 9.3 software package from Saxonica to transform the XML document. PROC XSL functionality is now production. For more information, see Chapter 56, "XSL Procedure," on page 1659.

Documentation Enhancements

The following changes have been made to the *Base SAS Procedures Guide*:

- The section titled "Base SAS Procedures Documented in Other Publications" contains links to Base SAS procedures that are documented in publications other than the *Base SAS Procedures Guide*. These procedures previously appeared as separate entries in the Table of Contents of the *Base SAS(R) 9.2 Procedures Guide*. For more information, see Chapter 5, "Base SAS Procedures Documented in Other Publications," on page 53.
- The SQL procedure documentation has been moved to the *SAS SQL Procedure User's Guide*.

Recommended Reading

- *The Little SAS(R) Book: A Primer*
- *Learning SAS(R) by Example: A Programmer's Guide*
- *Output Delivery System: The Basics and Beyond*
- *SAS(R) 9.3 Output Delivery System: User's Guide*
- *PROC TABULATE by Example*
- *SAS(R) Guide to Report Writing: Examples*
- *Carpenter's Complete Guide to the SAS(R) REPORT Procedure*
- *PROC SQL: Beyond the Basics Using SAS*
- *SAS(R) 9.3 SQL Procedure User's Guide*
- *SAS(R) Language Reference: Concepts*
- *Base SAS(R) 9.3 Utilities: Reference*
- *SAS(R) 9.3 Component Objects: Reference*
- *SAS(R) 9.3 Data Set Options: Reference*
- *SAS(R) 9.3 Formats and Informats: Reference*
- *SAS(R) 9.3 Functions: Reference*
- *SAS(R) 9.3 Statements: Reference*
- *SAS(R) 9.3 System Options: Reference*
- *SAS(R) 9.3 Guide to Metadata-Bound Libraries*

For a complete list of SAS publications, go to support.sas.com/bookstore. If you have questions about which titles you need, please contact a SAS Publishing Sales Representative:

SAS Publishing Sales
SAS Campus Drive
Cary, NC 27513-2414
Phone: 1-800-727-3228
Fax: 1-919-677-8166
E-mail: sasbook@sas.com
Web address: support.sas.com/bookstore

Part 1

Concepts

Chapter 1

Choosing the Right Procedure

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.

CALENDAR	PLOT	SQL*
CHART*	PRINT	SUMMARY*
FREQ*	QDEVICE*	TABULATE *
MEANS*	REPORT*	TIMEPLOT

* 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.

CHART	RANK	SUMMARY
CORR	REPORT	TABULATE
FREQ	SQL	UNIVARIATE
MEANS	STANDARD	

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

APPEND	FONTREG	PMENU
AUTHLIB	FORMAT	PRINTTO
BMDP*	FSLIST	PROTO
CATALOG	GROOVY	PRTDEF
CIMPORT	HADOOP	PRTEXP
COMPARE	IMPORT††	PWENCODE
CONTENTS	INFOMAPS †††	REGISTRY
CONVERT*	JAVAINFO	RELEASE*
COPY	METADATA‡	SCAPROC
CPORT	METALIB‡	SORT
CV2VIEW***	METAOPERATE‡	SOURCE*
DATASETS	MIGRATE	SQL
DBCSTAB†	OPTIONS	TAPECOPY *
DISPLAY	OPTLOAD	TAPELABEL *
DOCUMENT **	OPTSAVE	TEMPLATE **
EXPORT††	PDS*	TRANSPOSE
FCMP	PDSCOPY*	TRANTAB†

		XSL

* See the SAS documentation for your operating environment for a description of this procedure.
** For a description of this procedure, see the *SAS Output Delivery System: User's Guide*.
*** For a description of this procedure, see the *SAS/ACCESS for Relational Databases: Reference*.
† For a description of this procedure, see the *SAS National Language Support (NLS): Reference Guide*.
†† For a description of this procedure, see the *SAS/ACCESS Interface to PC Files: Reference*.
††† For a description of this procedure, see the *Base SAS Guide to Information Maps*.
‡ For a description of this procedure, see the *SAS Language Interfaces to Metadata*.

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	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.

Report Type	Procedure	Description
	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		
Calendars	CALENDAR	Produces schedule and summary calendars; can schedule tasks around nonwork periods and holidays, weekly work schedules, and daily work shifts.
Multipanel reports (telephone book listings)	REPORT	Produces multipanel reports.
Low-resolution graphical reports *		
	CHART	Produces bar charts, histograms, block charts, pie charts, and star charts that display frequencies and other statistics.
	PLOT	Produces scatter diagrams that plot one variable against another.
	TIMEPLOT	Produces plots of one or more variables over time intervals.

* These reports quickly produce a simple graphical picture of the data. To produce high-resolution graphical reports, use SAS/GRAPH software.

Statistical Procedures

Available Statistical Procedures

The following table lists statistical procedures according to task. Table A1.1 on page 1667 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	Computes most of the same statistics as PROC TABULATE; allows customization of format.

Report type	Procedure	Description
	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.
Standardizing data	STANDARD	Creates an output data set that contains variables that are standardized to a given mean and standard deviation.
Low-resolution graphics*		
	CHART	Produces a graphical report that can show one of the following statistics for the chart variable: frequency counts, percentages, cumulative frequencies, cumulative percentages, totals, or averages.
	UNIVARIATE	Produces descriptive plots such as stem-and-leaf plot, box plots, and normal probability plots.

* 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-intense. For more information, see "Quantiles" on page 798 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

"SAS Elementary Statistics Procedures" on page 1665 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.

Tasks	Procedure	Description
	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.
	OPTLOAD	Reads SAS system option settings that are stored in the SAS registry or a SAS data set.
	OPTSAVE	Zaves SAS system option settings to the SAS registry or a SAS data set.
Affect printing and Output Delivery System output	DOCUMENT **	Manipulates procedure output that is stored in ODS documents.
	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.
	PRTDEF	Creates printer definitions.
	PRTEXP	Exports printer definition attributes to a SAS data set.
	TEMPLATE**	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.
	FSLIST	Browses external files such as files that contain SAS source lines or SAS procedure output.
	INFOMAPS†††	Creates or updates a SAS Information Map.
	SQL	Creates SAS data sets using Structured Query Language and SAS features.
Transform data	DBCSTAB†	Produces conversion tables for the double-byte character sets that SAS supports.
	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.

Tasks	Procedure	Description
	SQL	Sorts SAS data sets by one or more variables.
	TRANSPOSE	Transforms SAS data sets so that observations become variables and variables become observations.
	TRANTAB†	Creates, edits, and displays customized translation tables.
	XSL	Transforms an XML document into another format, such as HTML, text, or another XML document type.
Manage SAS files	APPEND	Appends one SAS data set to the end of another.
	AUTHLIB	Manages metadata-bound libraries.
	BMDP*	Invokes a BMDP program to analyze data in a SAS data set.
	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*	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.
	CV2VIEW***	Converts SAS/ACCESS view descriptors to PROC SQL views.
	DATASETS	Manages SAS files.
	EXPORT††	Reads data from a SAS data set and writes them to an external data source.
	IMPORT††	Reads data from an external data source and writes them to a SAS data set.
	MIGRATE	Migrates members in a SAS library forward to the most current release of SAS.
	PDS*	Lists, deletes, and renames the members of a partitioned data set.
	PDSCOPY*	Copies partitioned data sets from disk to tape, disk to disk, tape to tape, or tape to disk.

Tasks	Procedure	Description
	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*	Releases unused space at the end of a disk data set under the z/OS environment.
	SOURCE*	Provides an easy way to back up and process source library data sets.
	SQL	Concatenates SAS data sets.
	TAPECOPY*	Copies an entire tape volume or files from one or more tape volumes to one output tape volume.
	TAPELABEL*	Lists the label information of an IBM standard-labeled tape volume in the z/OS environment.
Control windows	PMENU	Creates customized menus for SAS applications.
Miscellaneous	DISPLAY	Executes SAS/AF applications.
	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.
Manage metadata in a SAS Metadata Repository	METADATA‡	Sends a method call, in the form of an XML string, to a SAS Metadata Server.
	METALIB‡	Updates metadata to match the tables in a library.
	METAOPERATE‡	Performs administrative tasks on a metadata server.

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

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

*** For a description of this procedure, see the *SAS/ACCESS for Relational Databases: Reference*.

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

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

††† For a description of this procedure, see the *Base SAS Guide to Information Maps*.

‡ For a description of this procedure, see the *SAS Language Interfaces to Metadata*.

Brief Descriptions of Base SAS Procedures

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

AUTHLIB procedure
: manages metadata-bound libraries, which are physical libraries that are tied to corresponding metadata secured table objects. Each physical table within a metadata-bound library has information in its header that points to a specific metadata object.

BMDP procedure
: invokes a BMDP program to analyze data in a SAS data set. For more information, see the SAS documentation for your operating environment.

CALENDAR procedure
: displays data from a SAS data set in a monthly calendar format. PROC CALENDAR can display holidays in the month, schedule tasks, and process data for multiple calendars with work schedules that vary.

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.

CHART procedure
: produces vertical and horizontal bar charts, block charts, pie charts, and star charts. These charts provide a quick visual representation of the values of a single variable or several variables. PROC CHART can also display a statistic associated with the values.

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.

CONVERT procedure
: converts BMDP system files, OSIRIS system files, and SPSS portable files to SAS data sets. For more information, see the SAS documentation for your operating environment.

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.

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.

CV2VIEW procedure
: converts SAS/ACCESS view descriptors to PROC SQL views. Starting in SAS System 9, conversion of SAS/ACCESS view descriptors to PROC SQL views is recommended because PROC SQL views are platform-independent and enable you to use the LIBNAME statement. For more information, see the *SAS/ACCESS for Relational Databases: Reference*.

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.

DBCSTAB procedure
: produces conversion tables for the double-byte character sets that SAS supports. For more information, see the *SAS National Language Support (NLS): Reference Guide*.

DISPLAY procedure
: executes SAS/AF applications. For information about building SAS/AF applications, see the *Guide to SAS/AF Applications Development*.

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*.

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.

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.

FSLIST procedure
: displays the contents of an external file or copies text from an external file to the SAS Text Editor.

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.

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

INFOMAPS
: creates or updates a SAS Information Map. For more information, see the *Base SAS Guide to Information Maps*.

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.

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.

METADATA procedure
: sends a method call, in the form of an XML string, to a SAS Metadata Server.

METALIB procedure
: updates metadata in a SAS Metadata Repository to match the tables in a library.

METAOPERATE procedure
: performs administrative tasks on a metadata server.

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.

PDS procedure
: lists, deletes, and renames the members of a partitioned data set. For more information, see the *SAS Companion for z/OS*.

PDSCOPY procedure
: copies partitioned data sets from disk to tape, disk to disk, tape to tape, or tape to disk. For more information, see the *SAS Companion for z/OS*.

PLOT procedure
: produces scatter plots that graph one variable against another. The coordinates of each point on the plot correspond to the two variables' values in one or more observations of the input data set.

PMENU procedure
: defines menus that you can use in DATA step windows, macro windows, and SAS/AF windows, or in any SAS application that enables you to specify customized menus.

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, as well as from any SAS function, subroutine, or method block that is declared in the COMPILE procedure.

PRTDEF procedure
: creates printer definitions for individual SAS users or all SAS users.

PRTEXP procedure
: exports printer definition attributes to a SAS data set so that they can be easily replicated and modified.

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.

RELEASE procedure
: releases unused space at the end of a disk data set in the z/OS environment. For more information, see the *SAS Companion for z/OS*.

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.

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. Fore 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.

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.

SUMMARY procedure
: computes descriptive statistics for the variables in a SAS data set across all observations and within groups of observations, and outputs 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.

TAPECOPY procedure
: copies an entire tape volume or files from one or more tape volumes to one output tape volume. For more information, see the *SAS Companion for z/OS*.

TAPELABEL procedure
: lists the label information of an IBM standard-labeled tape volume under the z/OS environment. For more information, see the *SAS Companion for z/OS*.

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*.

TIMEPLOT procedure
: produces plots of one or more variables over time intervals.

TRANSPOSE 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 the *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.

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

Chapter 2

Fundamental Concepts for Using Base SAS Procedures

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 (see "USER Library" on page 18). 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 an external storage location that stores SAS data sets in your operating environment. 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=
- FORMDLIM=
- 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*.

Global Statements

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

comment	ODS
DM	OPTIONS
ENDSAS	PAGE
FILENAME	RUN
FOOTNOTE	%RUN
%INCLUDE	SASFILE
LIBNAME	SKIP
%LIST	TITLE
LOCK	X

For information about all the above statements except for the ODS statement, refer to the *SAS Statements: Reference*. For information about the ODS statement, refer to "Output Delivery System" on page 34 and to *SAS Output Delivery System: User's Guide*.

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*.

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 PLOT PMENU 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 name 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.

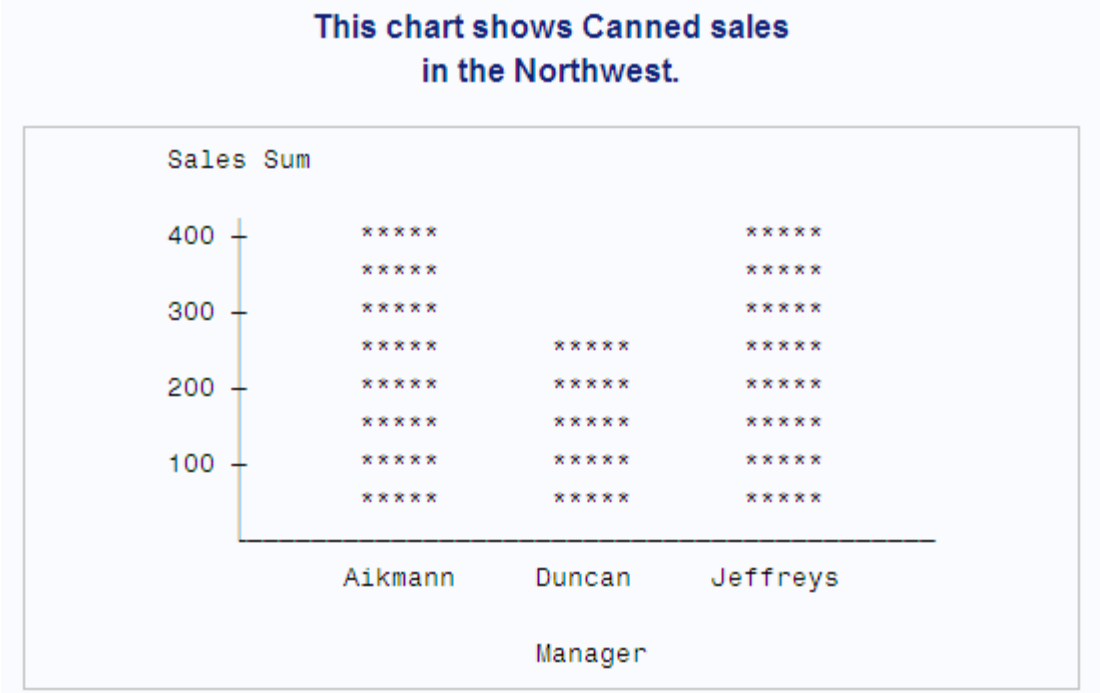
Example: Inserting a Value from Each BY Variable into the Title

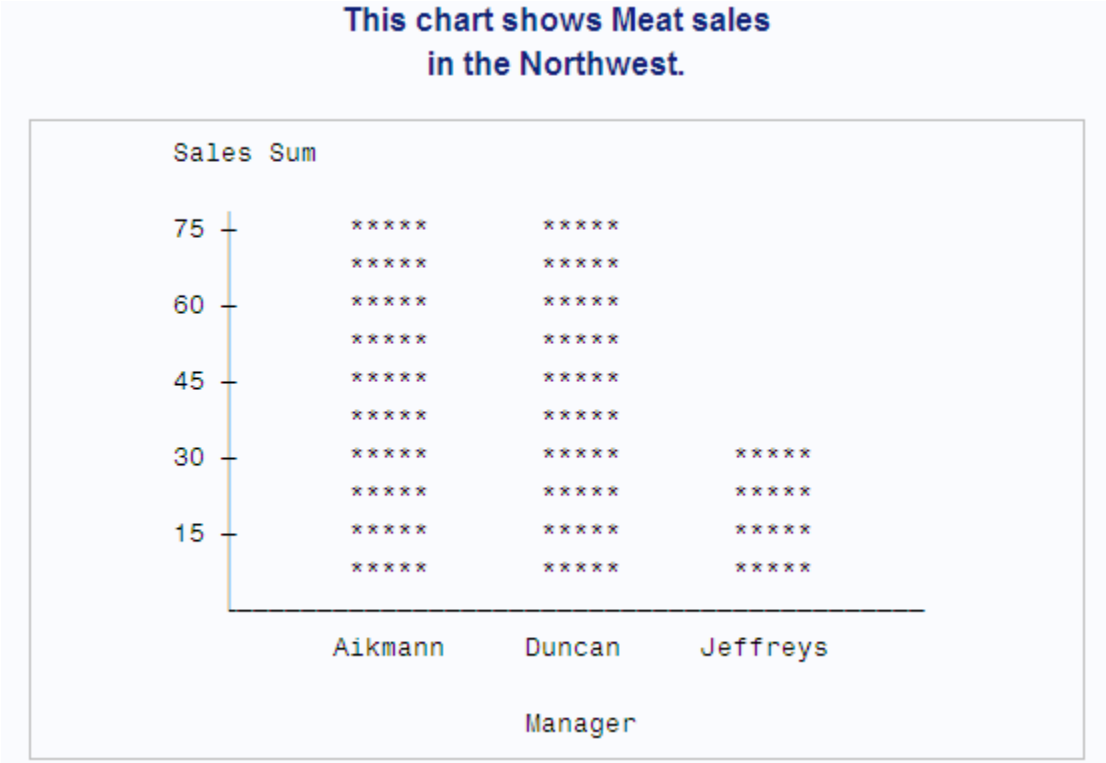
This example demonstrates these actions:

1. creates a data set, GROC, that contains data for stores from four regions. Each store has four departments. See "GROC" on page 1745 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 CHART 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; 1
   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;  2
   by region department;
run;
options nobyline nodate pageno=1
        linesize=64 pagesize=20;  3
proc chart data=groc;  4
   by region department;
   vbar manager / type=sum sumvar=sales;
   title1 'This chart shows #byval2 sales';
   title2 'in the #byval(region)..';
run;
options byline;  5
```

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





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

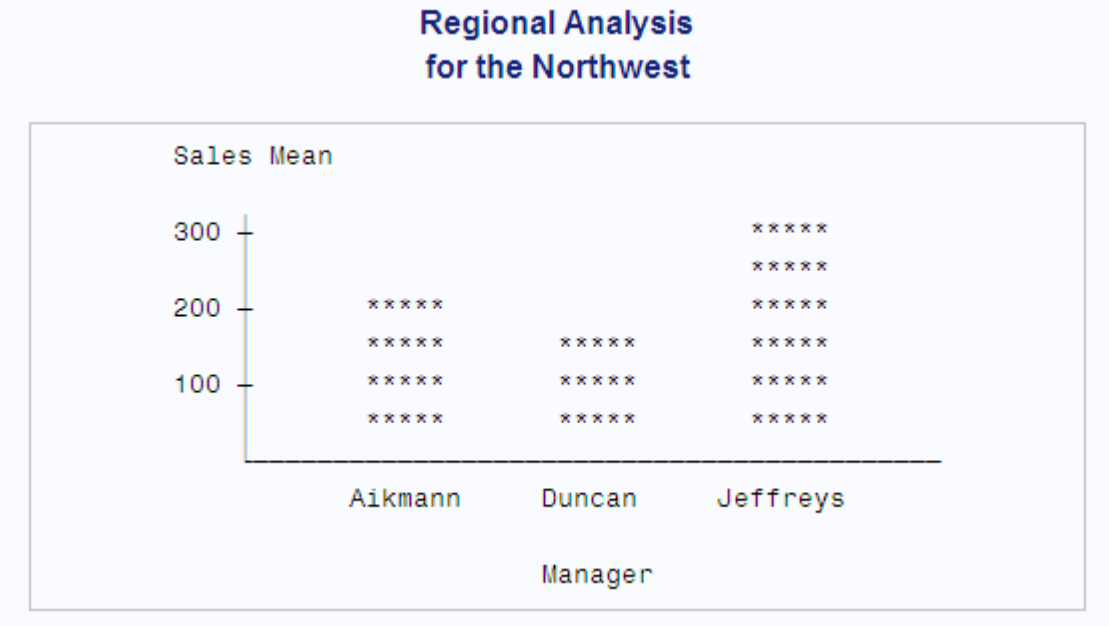
This example inserts the name of a BY variable and the value of a BY variable into the title. The program does these actions.

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 CHART 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 chart data=groc;   2
   by region;
   vbar manager / type=mean sumvar=sales;
   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:



Example: Inserting the Complete BY Line into a Title

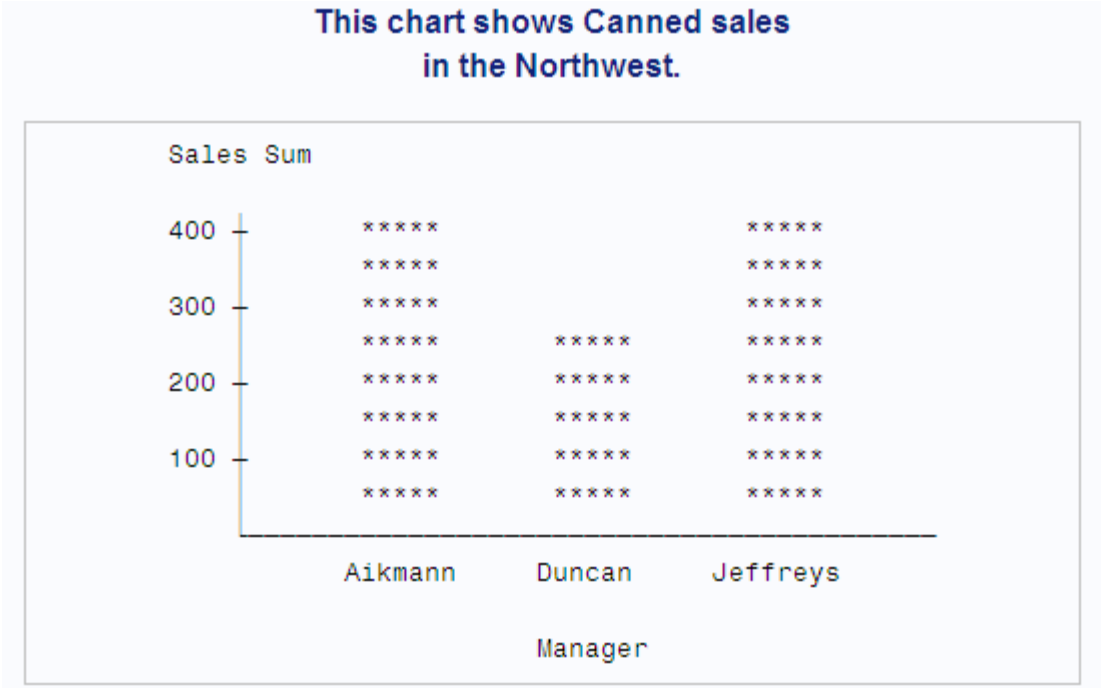
This example inserts the complete BY line into the title. The program does these actions:

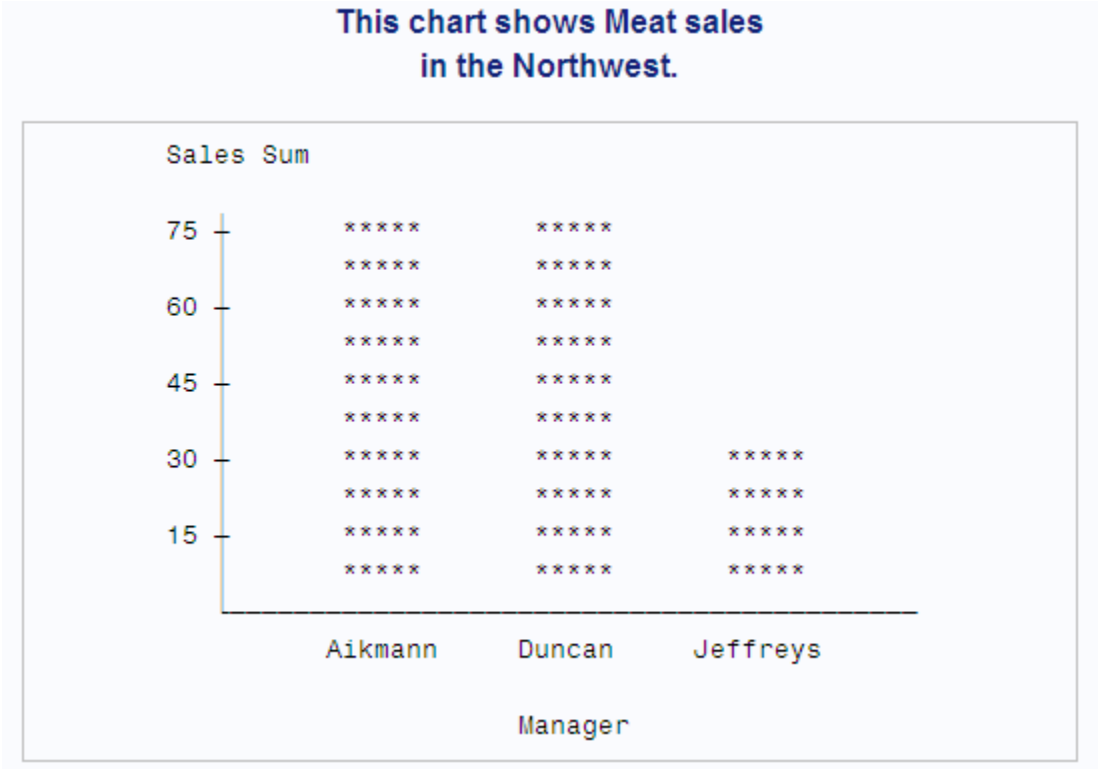
1. uses the SAS system option NOBYLINE to suppress the BY line that normally appears in output that is produced with BY-group processing.
2. uses PROC CHART to chart sales by Region and Department. In the TITLE statement, #BYLINE inserts the complete BY line into the title.
3. uses the SAS system option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





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-x*n*	Specifies variables X1 through X*n*. The numbers must be consecutive.
x:	Specifies all variables that begin with the letter X.
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 Language Reference: Concepts*.

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 1752.) 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 628.

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 9: Printing All the Data Sets in a SAS Library" on page 1068 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.

Operating Environment-Specific Procedures

Several Base SAS procedures are specific to one operating environment or one release. "Descriptions of Operating Environment-Specific Procedures" on page 1705 contains a table with additional information. These procedures are described in more detail in the SAS documentation for operating environments.

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 1666.

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

Statistic	Description	Procedures
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
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 32. 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.

Before Version 7, most SAS procedures generated output that was designed for a traditional line-printer. This type of output has limitations that prevent you from getting the most value from your results:

- Traditional SAS output is limited to monospace fonts. With today's desktop document editors and publishing systems, you need more versatility in printed output.
- Some commonly used procedures do not produce output data sets. Before ODS, if you wanted to use output from one of these procedures as input to another procedure, then you relied on PROC PRINTTO and the DATA step to retrieve results.

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

Chapter 3

Statements with the Same Function in Multiple Procedures

Statements with the Same Function in Multiple Procedures

Overview

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 Statements: Reference*, and others are documented in this section. The following list shows you where to find more information about each statement:

ATTRIB
: affects the procedure output and the output data set. The ATTRIB statement does not permanently alter the variables in the input data set. The LENGTH= option has no effect. For the complete documentation, see the *SAS Statements: Reference*.

BY
: orders the output according to the BY groups. See "BY" on page 36.

FORMAT
: affects the procedure output and the output data set. The FORMAT statement does not permanently alter the variables in the input data set. The DEFAULT= option is not valid. For the complete documentation, see the *SAS Statements: Reference*.

FREQ
: treats observations as if they appear multiple times in the input data set. See "FREQ" on page 40.

LABEL
: affects the procedure output and the output data set. The LABEL statement does not permanently alter the variables in the input data set except when it is used with the

MODIFY statement in PROC DATASETS. For complete documentation, see the *SAS Statements: Reference*.

QUIT
executes any statements that have not executed and ends the procedure. See "QUIT" on page 42.

WEIGHT
specifies weights for analysis variables in the statistical calculations. See "WEIGHT" on page 43.

WHERE
subsets the input data set by specifying certain conditions that each observation must meet before it is available for processing. See "WHERE" on page 48.

Statements

BY

Overview of the BY Statement

Orders the output according to the BY groups.

For more information, see "Creating Titles That Contain BY-Group Information " on page 21.

BY <DESCENDING> *variable-1*
<… <DESCENDING>*variable-n*>
<NOTSORTED>;

Required Arguments

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.

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 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 Statements: Reference*.
- For more information about PROC SORT, see Chapter 50, "SORT Procedure," on page 1425.
- For more information about creating indexes, see "INDEX CREATE Statement" on page 426.

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 the 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

CALENDAR	REPORT (nonwindowing environment only)
CHART	SORT (required)
COMPARE	STANDARD

CORR	SUMMARY
FREQ	TABULATE
MEANS	TIMEPLOT
PLOT	TRANSPOSE
PRINT	UNIVARIATE
RANK	

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.

Example

This example uses a BY statement in a PROC PRINT step. There is output for each value of the BY variable Year. The DEBATE data set is created in "Example: Temporarily Dissociating a Format from a Variable" on page 30.

```
options nodate pageno=1 linesize=64
        pagesize=40;
proc print data=debate noobs;
   by year;
   title 'Printing of Team Members';
   title2 'by Year';
run;
```

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

FREQ

Overview of the 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.

FREQ *variable*;

Required Arguments

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.

Procedures That Support the FREQ Statement

- CORR
- MEANS or SUMMARY
- REPORT
- STANDARD
- TABULATE
- UNIVARIATE

Example

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 following PROC MEANS step does not use a 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;
```

Without a frequency variable, each observation has a frequency of 1, and the total number of observations is 8.

Average Course and Speed

The MEANS Procedure

Variable	N	Mean
Course	8	48.50
Speed	8	7.88

The second PROC MEANS step uses Hours as a frequency variable:

```
proc means data=track maxdec=2 n mean;
   var course speed;
   freq hours;
   title 'Average Course and Speed';
run;
```

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).

Average Course and Speed

The MEANS Procedure

Variable	N	Mean
Course	141	49.28
Speed	141	8.06

QUIT

Overview of the QUIT Statement

Executes any statements that have not executed and ends the procedure.

QUIT;

Procedures That Support the QUIT Statement

- CATALOG
- DATASETS
- PLOT
- PMENU
- SQL

WEIGHT

Overview of the 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.

WEIGHT *variable*;

Required Arguments

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 behavior of the procedure when it encounters a nonpositive weight variable value is as follows:

Different behavior for nonpositive values is discussed in the WEIGHT statement syntax under the individual procedure.

Before Version 7 of SAS, no Base SAS procedure excluded 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

The procedure substitutes the value of the WEIGHT variable for w_j, which appears in "Keywords and Formulas" on page 1666.

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* .

Weighted Statistics Example

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/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 following statements create the data set SIZE:

```
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
;
```

The following 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.

```
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

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.

```
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;
```

The variance of the *i*th observation is assumed to be $var(x_i) = \sigma^2 / w_i$ 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.

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

The following statements create and print a data set with the weighted variance and weighted standard deviation of each observation. The 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;
```

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

Overview of the WHERE Statement

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

WHERE *where-expression*;

Required Arguments

where-expression

is a valid arithmetic or logical expression that generally consists of a sequence of operands and operators. See *SAS Statements: Reference* for more information about where processing.

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:

CALENDAR	RANK
CHART	REPORT
COMPARE	SORT
CORR	SQL
DATASETS (APPEND statement)	STANDARD

FREQ	TABULATE
MEANS or SUMMARY	TIMEPLOT
PLOT	TRANSPOSE
PRINT	UNIVARIATE

Details

- The CALENDAR and COMPARE procedures and the APPEND statement in PROC DATASETS accept more than one input data set. See the documentation for the specific procedure for more information.
- 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 Statements: Reference*.

Example

In this example, PROC PRINT prints only those observations that meet the condition of the WHERE expression. The DEBATE data set is created in "Example: Temporarily Dissociating a Format from a Variable" on page 30.

```
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;
```

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

Chapter 4

In-Database Processing of Base Procedures

Base Procedures That Are Enhanced for In-Database Processing

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 the third maintenance release for SAS 9.2, Base SAS procedures were enhanced to process data inside the Teradata Enterprise Data Warehouse (EDW), DB2 under UNIX, and Oracle database management systems (DBMS). In SAS 9.3, procedures have been enhanced to also process data inside the Netezza DBMS. The in-database procedures are used to generate more sophisticated queries that allow the aggregations and analytics to be run inside the database.

All of these in-database procedures generate SQL queries. You use SAS/ACCESS or SQL as the interface to the Teradata EDW.

The following Base SAS procedures support in-database processing.

Table 4.1 *In-Database Base Procedures*

Procedure	Description
PROC FREQ in *Base SAS(R) 9.3 Procedures Guide: Statistical Procedures*	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.

Procedure	Description
PROC MEANS on page 767	Computes descriptive statistics; can produce printed output and output data sets. By default, PROC MEANS produces printed output.
PROC RANK on page 1185	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 1296	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 1446	Orders SAS data set observations by the values of one or more character or numeric variables.
PROC SUMMARY on page 1475	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 1528	Displays descriptive statistics in tabular format, using some or all of the variables in a data set.

For more information, see “In-Database Procedures in Teradata” in *SAS/ACCESS for Relational Databases: Reference*.

Chapter 5

Base SAS Procedures Documented in Other Publications

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(R) 9.3 Procedures Guide: Statistical Procedures*
CV2VIEW	Converts SAS/ACCESS view descriptors into SQL views.	*SAS/ACCESS for Relational Databases: Reference*
DBCSTAB	Produces conversion tables for the double-byte character sets that SAS supports.	*SAS National Language Support (NLS): Reference Guide*
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
EXPLODE	Produces printed output with oversized text by expanding each letter into a matrix of characters.	The EXPLODE Procedure
FORMS	Produces labels for envelopes, mailing labels, external tape labels, file cards, and any other printer forms that have a regular pattern.	The FORMS Procedure
FREQ	Produces one-way to n-way frequency and contingency (crosstabulation) tables.	*Base SAS(R) 9.3 Procedures Guide: Statistical Procedures*
INFOMAPS	Enables you to create information maps programmatically	*Base SAS Guide to Information Maps*
METADATA	Sends an XML string to the SAS Metadata Server.	*SAS Language Interfaces to Metadata*
METALIB	Updates the metadata in the metadata server to match the tables in a library.	*SAS Language Interfaces to Metadata*
METAOPERATE	Enables you to perform administrative tasks in batch mode that are associated with the SAS Metadata Server.	*SAS Language Interfaces to Metadata*
SGDESIGN	Produces a graph from one or more input SAS data sets and a user-defined ODS Graphics Designer (SGD) file.	*SAS ODS Graphics: Procedures Guide*
SGPPANEL	Creates a panel of graph cells for the values of one or more classification variables.	*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*

Procedure	Description	Publication
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*
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(R) 9.3 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.

Part 2

Procedures

Chapter 6

APPEND Procedure

Overview: APPEND Procedure

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

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= 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.

Syntax: APPEND Procedure

Tips: You can use the ATTRIB, FORMAT, LABEL, and WHERE statements.

You can use data set options with the BASE= and DATA= options.

Complete documentation for the APPEND procedure is located within the DATASETS procedure in APPEND Statement on page 386.

PROC APPEND BASE=<*libref.*>*SAS-data-set* <DATA=<*libref.*>*SAS-data-set*> <FORCE> <APPENDVER=V6> <GETSORT>;

Statement	Task	Example
"APPEND Statement"	Add observations from one SAS data set to the end of another SAS data set	Ex. 1

Statement	Task	Example
"APPENDVER=V6"	Add observations to the data set one at a time	
"BASE=<libref.>SAS-data-set"	Name of destination data set	Ex. 1
"DATA=<libref.>SAS-data-set"	Name of source data set	Ex. 1
"FORCE"	Force the append when variables are different	Ex. 1
"GETSORT"	Copy the sort indicator that was established by using PROC SORT from the DATA= data set to the BASE= data set	Ex. 2
"NOWARN"	Suppress the warning message when used with the FORCE option to concatenate two data sets with different variables	

Using 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: APPEND statement options
- BASE=
- DATA=
- FORCE=

This example appends one data set to the end of another data set.

The BASE= data set, EXP.RESULTS

```
The EXP.RESULTS Data Set                    1

  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
```

The data set EXP.SUR contains the variable WT6MOS, but the EXP.RESULTS data set does not.

```
      The EXP.SUR Data Set                  2

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
```

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 EXP.RESULTS Data Set';
run;
```

Program Description

```
options pagesize=40 linesize=64 nodate pageno=1;

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 EXP.RESULTS Data Set';
run;
```

Output 6.1 *Output*

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: Getting Sort Indicator Information

Features: APPEND statement option
GETSORT
Other feature
SORTEDBY data set option

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;
```

Program Description

The following example shows that a sort indicator can be inherited using the GETSORT option with the APPEND statement.

Create a "shell" data set that contains no observations.

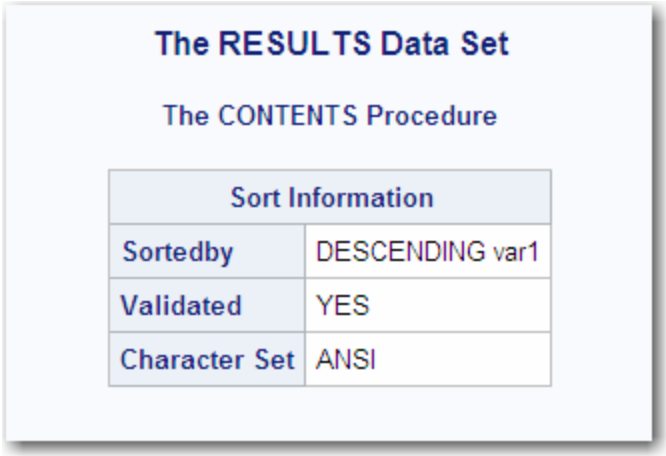
```
data mtea;
    length var1 8.;
    stop;
run;
```

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;
```

Output 6.2 *Sort Information Output*



The RESULTS Data Set

The CONTENTS Procedure

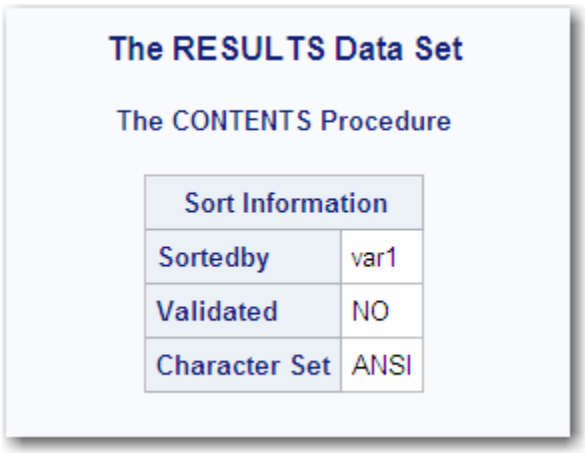
Sort Information	
Sortedby	DESCENDING var1
Validated	YES
Character Set	ANSI

This example shows sort indicators using the SORTEDBY data set option and the SORT procedure.

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;
```

Output 6.3 *Sort Information Output*



The RESULTS Data Set

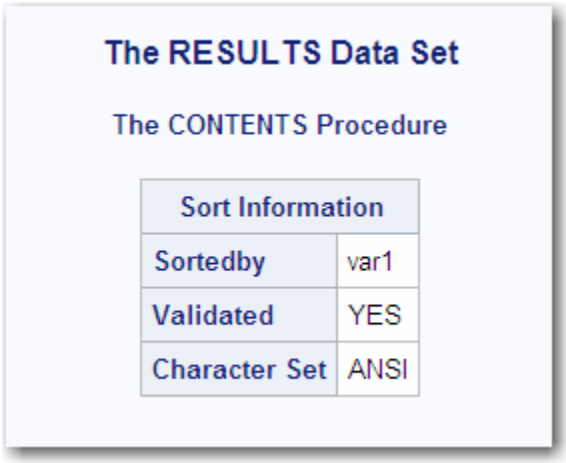
The CONTENTS Procedure

Sort Information	
Sortedby	var1
Validated	NO
Character Set	ANSI

This example shows the sort indicator information using the SORT procedure.

```
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 6.4 *Sort Information Output*



The RESULTS Data Set

The CONTENTS Procedure

Sort Information	
Sortedby	var1
Validated	YES
Character Set	ANSI

Chapter 7

AUTHLIB Procedure

Overview: AUTHLIB Procedure

The AUTHLIB procedure is a utility procedure that manages metadata-bound libraries. With PROC AUTHLIB, you can do the following:

- create a metadata-bound library by binding a physical library to metadata within a SAS Metadata Repository

- modify password values for a metadata-bound library
- repair metadata-bound libraries by recovering security information, secured library objects, and secured table objects (this functionality is preproduction in this release)
- remove the physical security information and metadata objects that protect a metadata-bound library
- report inconsistencies between physical library contents and corresponding metadata objects within a specified metadata-bound library

Users cannot access metadata-bound data sets from any release of SAS prior to the second maintenance release of SAS 9.3.

Note: For a z/OS direct-access bound library that has been bound to metadata, the constraint is slightly broader—neither the library nor any of its members can be accessed by earlier releases of SAS.

Concepts: AUTHLIB Procedure

What Is a Metadata-Bound Library?

A metadata-bound library is a physical library that is tied to a corresponding metadata secured table object. Each physical table within a metadata-bound library has information in its header that points to a specific metadata object. The pointer creates a security binding between the physical table and the metadata object. The binding ensures that SAS universally enforces metadata-layer access requirements for the physical table —regardless of how a user requests access from SAS. For more information, see *SAS Guide to Metadata-Bound Libraries*.

What Are Metadata-Bound Library Passwords?

A metadata-bound library has a single set of passwords stored in the secured library object, which are added to all data sets that are created in the metadata-bound library. These passwords are not used to authorize user access to the data, but rather to authorize administrator access to repair the binding of physical data to the secured library or table metadata objects. They are also validated in the process of authorizing a user's access to a data set but do not determine the permissions that any user is authorized to have.

The metadata-bound library passwords are intended to be known only by the administrators of the metadata-bound library. Knowledge of these passwords is required to restore or re-create secured library and secured table objects in a SAS Metadata Server for data sets in a data library that have lost their previously recorded metadata objects and permissions. The metadata-bound library passwords also prevent a user from exporting the secured library and secured table objects from a SAS Metadata Server and then importing them to a SAS Metadata Server that an unauthorized user created and controls. This prevents the unauthorized user from using such objects where the user has modified the permissions.

The metadata-bound library passwords are always stored and transmitted in encrypted formats. The encrypted password is not usable to access the data if it is captured from a transmission and presented to SAS as a password value in the SAS language. Administrators might choose to use the PWENCODE procedure to encode the passwords for use in a PROC AUTHLIB statement. Using an encoded password

prevents a casual observer from seeing the clear-text password in the PROC AUTHLIB statements that the administrator types.

There are three passwords in the metadata-bound library set that correspond to the Read, Write, and Alter passwords of SAS data sets. For greater simplicity in administration of metadata-bound libraries, it is recommended that you use the PW= option in PROC AUTHLIB statements to specify a single password value, rather than specifying different password values using READ=, WRITE=, and ALTER= options. In the context of metadata-bound libraries, the READ=, WRITE=, and ALTER= options do not create access distinctions. If you are concerned that a single eight character password does not meet your security requirements, you can choose to set three different password values (using READ=, WRITE=, and ALTER=). Setting different values for these three options can create a 24-character password. However, you must keep track of all password values that you have assigned to a metadata-bound library as you must specify them to unbind the library, modify the passwords, or repair any inconsistencies in the binding information between what is recorded in the physical files and the actual metadata objects.

CAUTION:

If you lose the password (or passwords) for a metadata-bound library, you cannot unbind the library or change its passwords. Be sure to keep track of passwords that you assign in the CREATE and MODIFY statements.

Data Sets in a Metadata-Bound Library That Are Not Bound to Secured Table Objects

It is possible to have physical data sets in a metadata-bound library that do not have the metadata-bound library passwords. This can occur if the data sets existed with passwords that differ from the metadata library passwords when the library was bound. See "Example 2: Binding a Physical Library That Contains Password-Protected Data Sets" on page 89. It can also occur if data sets with different passwords are copied into the library by an operating system copy utility. These data sets are not considered to be part of the bound library for authorization purposes. If the data set was in an operating system copied from another metadata-bound library, the data set is still protected by the permissions users have in the secured table object to which it is bound in the original secured library. If the data set is not copied from a metadata-bound library, then metadata permissions do not apply, and you must supply the appropriate passwords to access the data. You can use the MODIFY statement of PROC AUTHLIB to modify the passwords to those of the metadata-bound library so that it will be bound to a secured table object in the secured library object to which the library is bound. See "Example 5: Changing Passwords on Data Sets " on page 93.

Requirements to Use PROC AUTHLIB Statements

Except for the REPORT statement, all statements within PROC AUTHLIB require that you must meet the following criteria:

- The SAS session runs under an account that has host-layer control of the target physical library. To ensure that only users who have host control can bind a physical library to metadata, the SAS session must run under a privileged host account as follows:
  - On UNIX, the account must be the owner of the directory.
  - On Windows, the account must have full control of the directory.

  - On z/OS, for UNIX file system libraries, the account must be the owner of the directory.
  - On z/OS, for direct-access bound libraries, the account must have RACF ALTER access authority to the library data set.
- The SAS session connects to the metadata server as an identity that has ReadMetadata and WriteMemberMetadata permissions to the target secured data folder.
- You must supply the password(s) in CREATE, MODIFY, REPAIR, and REMOVE statements.

The REPORT statement requirements are less restrictive and are documented with that statement.

Syntax: AUTHLIB Procedure

Restrictions: Users cannot access metadata-bound data sets from any release of SAS prior to the second maintenance release of SAS 9.3.

This procedure is intended for use by SAS administrators. Users who lack sufficient privileges in either the metadata layer or the host layer cannot use this statement.

This procedure does not support libraries accessed through a SAS/SHARE Server.

Requirement: A connection to the target metadata server.

See: For more information, see *SAS Guide to Metadata-Bound Libraries*.

PROC AUTHLIB <*option-1* <*...option-n*>>;

CREATE

SECUREDLIBRARY='*secured-library-name*'
<SECUREDFOLDER='*secured-folder-path*'>
<LIBRARY=*libref*>
PW=*all-password-value* | |
ALTER=*alter-password-value*
READ=*read-password-value*
WRITE=*write-password-value*;

MODIFY

<LIBRARY=*libref*>
PW=*all-password* | |
ALTER=*alter-password*
READ=*read-password*
WRITE=*write-password*
<TABLESONLY=YES|NO>;

REMOVE

<LIBRARY=*libref*>
PW=*all-password* | |
ALTER=*alter-password*
READ=*read-password*
WRITE=*write-password*;

REPAIR ADD | UPDATE | DELETE

LOCATION | METADATA

SECUREDLIBRARY='*secured-library-name*'
SECUREDFOLDER='*secured-folder-path*'
<SECUREDLIBRARYGUID='*secured-library-guid*'>
<LIBRARY=*libref*>
PW=*all-password* | |
ALTER=*alter-password*
READ=*read-password*
WRITE=*write-password*
<TABLESONLY=YES | NO>;

REPORT

<LIBRARY=*libref*>;

TABLES *SAS-file-1* | _ALL_ | _NONE_

<*SAS-file-n*>
<LIBRARY=*libref*>
PW=*all-password* | |
ALTER=*alter-password*
READ=*read-password*
WRITE=*write-password*
< SECUREDTABLEGUID='*secured-table-guid*'>;

Statement	Task	Example
"PROC AUTHLIB Statement"	Create and manage metadata-bound libraries	

Statement	Task	Example
"CREATE Statement"	Create the secured library object in the SAS Metadata Server and record the physical security information in the directory or bound files	Ex. 1, Ex. 2, Ex. 3, Ex. 4
"MODIFY Statement"	Modify password values for a metadata-bound library	Ex. 5, Ex. 6
"REMOVE Statement"	Remove the physical security information and metadata objects that protect a metadata-bound library	Ex. 7
"(Preproduction) REPAIR Statement"	Recover security information (in physical data) or secured library and table objects (in metadata)	
"REPORT Statement"	For a specified metadata-bound library, compare physical library contents with corresponding metadata objects (in order to identify any inconsistencies).	Ex. 8
"TABLES Statement"	Specify which tables within a specified metadata-bound library are affected by certain AUTHLIB statements	Ex. 4, Ex. 9

PROC AUTHLIB Statement

Manages metadata-bound libraries.

Syntax

PROC AUTHLIB <*option-1* <*...option-n*>>;

Optional Arguments

LIBRARY=*libref*
is the name of the physical library for which the secured library object is created and the security information is stored.

If the LIBRARY= option is not specified, the LIBRARY=*libref* (physical library) from the CREATE, MODIFY, REMOVE, REPORT, or REPAIR statement is used.

Alias: LIB=, DDNAME=, DD=

Restriction: The physical library specified cannot be a concatenated library or temporary library and must be processed by an engine that supports metadata-bound libraries.

NOWARN
suppresses the `file not found` error message when a data set in a TABLES statement does not exist.

PWREQ= YES | NO
controls the pop up of a dialog box for a data set password in interactive mode.

YES
specifies that a dialog box appear if a missing or an invalid password is entered when required.

NO
prevents a dialog box from appearing. If a missing or invalid password is entered, the data set is not opened, and an error message is written to the SAS log.

Default: PWREQ=NO

CREATE Statement

Binds a physical library to metadata by generating corresponding metadata objects in the SAS Metadata Repository and creating a record of the metadata objects in the physical directory.

Requirement: A connection to the target metadata server. For more requirements, see "Requirements to Use PROC AUTHLIB Statements" on page 71.

Syntax

PROC AUTHLIB <*option-1* <*...option-n*>>;
CREATE
SECUREDLIBRARY='*secured-library-name*'
<SECUREDFOLDER='*secured-folder-path*'>
<LIBRARY=*libref*>
PW=*all-password-value* | |
ALTER=*alter-password-value*
READ=*read-password-value*
WRITE=*write-password-value*;

Required Arguments

SECUREDLIBRARY='*secured-library-name*'
names the secured library object in the SAS Metadata Server.

Alias: SECLIB=

Restriction: The total length of the secured library object pathname including the fully qualified secured folder path cannot exceed 256 characters.

PW=*all-password-value*
sets a single password for a metadata-bound library.

ALTER=*alter-password-value*
sets one of a maximum of three password values for a metadata-bound library.

READ=*read-password-value*
sets one of a maximum of three password values for a metadata-bound library.

WRITE=*write-password-value*
sets one of a maximum of three password values for a metadata-bound library.

Optional Arguments

SECUREDFOLDER='*secured-folder-path*'
is the name of the metadata folder within the **`/System/Secured Libraries`** folder tree where the secured library object will be created.

If the SECUREDFOLDER= option is not specified, the metadata-bound library is created directly in the **`/System/Secured Libraries`** folder of the Foundation repository. If the SECUREDFOLDER= option does not begin with a slash (/), it is a relative path and the value will be appended to **`/System/Secured Libraries/`**

to find the folder. If the SECUREDFOLDER= option begins with a slash (/), it is an absolute path and the value must begin with **`/System/Secured Libraries`** or **`/<repository_name>/System/Secured Libraries`**.

Alias: SECFLDR=

Restriction: The total length of the secured library object pathname including the fully qualified secured folder path cannot exceed 256 characters.

LIBRARY=*libref*
name of the physical library for which the secured library object is created and the security information is stored.

If the LIBRARY= option is not specified, the physical library from the AUTHLIB procedure is used.

Alias: LIB=, DDNAME=, DD=

Restriction: The physical library specified cannot be a concatenated library or temporary library and must be processed by an engine that supports metadata-bound libraries.

Details

Specifying Passwords

If your physical library does not contain password-protected data sets, you need to specify the new metadata-bound library password(s) with either the PW= option or READ=,WRITE=, and ALTER= options in the CREATE statement. This is the most common case. See "Example 1: Binding a Physical Library That Contains Unprotected Data Sets" on page 88.

If your physical library contains some password-protected data sets that all share the same current set of passwords, then you can specify the most restrictive password on the data sets before a slash (/) in the CREATE statement password option(s) and the new password(s) after the slash (/). See "Example 3: Securing a Library When Existing Data Sets Are Protected with the Same Passwords" on page 90.

If your physical library contains password-protected data sets with different sets of passwords, then you can specify the data sets with each set of passwords on separate TABLES statements (see "Example 4: Securing a Library When Existing Data Sets Are Protected with Different Passwords" on page 91) or you can subsequently use MODIFY and TABLES statements to change the passwords after the library has been bound with the CREATE statement (see "Example 5: Changing Passwords on Data Sets " on page 93).

MODIFY Statement

Modifies password values for a metadata-bound library.

Requirement: A connection to the target metadata server. For more requirements, see "Requirements to Use PROC AUTHLIB Statements" on page 71.

Syntax

PROC AUTHLIB <*option-1* <*...option-n*>>;

MODIFY

<LIBRARY=*libref*>
PW=*all-password* | |
ALTER=*alter-password*
READ=*read-password*
WRITE=*write-password*
<TABLESONLY=YES | NO>;

Required Arguments

PW=*all-password*
modifies a single password for a metadata-bound library.

ALTER=*alter-password*
modifies one of a maximum of three password values for a metadata-bound library.

READ=*read-password*
modifies one of a maximum of three password values for a metadata-bound library.

WRITE=*write-password*
modifies one of a maximum of three password values for a metadata-bound library.

Optional Arguments

LIBRARY=*libref*
name of the physical library for which the metadata-bound library is created and the security information is stored.

If the LIBRARY= option is not specified, the physical library from the AUTHLIB procedure is used.

Alias: LIB=, DDNAME=, DD=

Restriction: The physical library specified cannot be a concatenated library or temporary library and must be processed by an engine that supports metadata-bound libraries.

TABLESONLY=YES | NO
specifies whether the MODIFY statement action is applied at the library level or just to the tables. If TABLESONLY=NO, the action is applied to the library and data sets. If TABLESONLY=YES, the action is applied only to the data sets.

Default: NO

Details

Using the MODIFY Statement

The MODIFY statement can modify the value of the required metadata-bound library passwords. This statement can also modify passwords on data sets (tables) that do not have the required metadata-bound library password values. The TABLES statement follows the MODIFY statement to specify current passwords in the data sets.

If your physical library is currently bound to a metadata library with one set of passwords and you want to change the metadata-bound library passwords to another set, specify the current and new values for the metadata-bound library passwords separated by a **/** in the MODIFY statement. See "Example 6: Changing Metadata-Bound Library Passwords" on page 94.

If your physical library contains password-protected data sets with different sets of passwords from the metadata-bound library passwords, then you can modify the data set passwords to match the metadata-bound library required passwords using the MODIFY and TABLES statements. Specify the metadata-bound library passwords in the MODIFY statement. Specify the data sets with each set of passwords on separate TABLES statements. See "Example 5: Changing Passwords on Data Sets " on page 93.

Using the LIBRARY= Option

If you want to override the default library from the AUTHLIB procedure, use LIBRARY=.

```
AUTHLIB MODIFY <LIBRARY=library-name>
```

If for some reason you want to modify the passwords for a secured library object that is no longer bound to a physical library, specify LIBRARY=_NONE_ with the SECUREDLIBRARY= and SECUREDFOLDER= options to locate the secured library object.

```
AUTHLIB MODIFY <LIBRARY=_NONE_ SECUREDLIBRARY=secured-library-name>
               <SECUREDFOLDER=secured-folder-name>
```

CAUTION:

Do not use LIB=_none_ when the secured library object is bound to a physical library. LIB=_none_ causes the action to operate only on the secured library object and has no effect on the physical data.

REMOVE Statement

Removes the physical security information and metadata objects that protect a metadata-bound library so it is no longer a metadata-bound library.

Requirement: A connection to the target metadata server. For more requirements, see "Requirements to Use PROC AUTHLIB Statements" on page 71.

Syntax

PROC AUTHLIB <*option-1* <*...option-n*>>;

REMOVE

<LIBRARY=*libref*>
PW=*all-password* | |
ALTER=*alter-password*
READ=*read-password*
WRITE=*write-password*;

Required Arguments

PW=*all-password*
specifies a single password for a metadata-bound library.

ALTER=*alter-password*
specifies one of a maximum of three password values for a metadata-bound library.

READ=*read-password*
specifies one of a maximum of three password values for a metadata-bound library.

WRITE=*write-password*
specifies one of a maximum of three password values for a metadata-bound library.

Optional Argument

LIBRARY=*libref*
name of the physical library where the metadata-bound library is created and the security information is stored.

If the LIBRARY= option is not specified, the physical library from the PROC AUTHLIB statement is used.

Alias: LIB=, DDNAME=, DD=

Restriction: The physical library specified cannot be a concatenated library or temporary library and must be processed by an engine that supports metadata-bound libraries.

Details

The REMOVE statement is used to unbind the metadata-bound library feature from a SAS data library and the data sets within it. This statement also removes the secured library and secured table objects from the SAS Metadata Server. The data sets remain in the physical library protected by the metadata-bound library passwords unless the administrator specifies password modifications in the REMOVE statement. Since the metadata-bound library feature is being removed and there is no longer a requirement that the data set passwords match the metadata-bound library passwords, the data set passwords can be removed by using a `/` after the current password but not specifying a new password value. If you choose to do this, you are warned in the SAS log that the data sets no longer have any SAS protection.

The REMOVE statement will not unbind any data sets that are currently bound to a secured table object in a different secured library than the one to which this physical library is bound.

Note: Ensure that all physical tables that are protected by a particular metadata-bound library remain within that library (directory). This standard, default state maximizes clarity and is essential for REMOVE statements to be fully effective. Special circumstances (for example, a table that is host copied to another directory) can prevent a REMOVE statement from unbinding the relocated data set.

(Preproduction) REPAIR Statement

Recovers security information (in physical data) or secured library and table objects (in metadata).

Requirement: A connection to the target metadata server. For more requirements, see “Requirements to Use PROC AUTHLIB Statements” on page 71.

Note: The REPAIR statement is a preproduction feature. For more information, see “Details” on page 81.

Syntax

PROC AUTHLIB <*option-1* <*...option-n*>>;

REPAIR ADD | UPDATE | DELETE

LOCATION | METADATA

SECUREDLIBRARY='*secured-library-name*'
SECUREDFOLDER='*secured-folder-path*'
<SECUREDLIBRARYGUID='*secured-library-guid*'>
<LIBRARY=*libref*>
PW=*all-password* | |
ALTER=*alter-password*
READ=*read-password*
WRITE=*write-password*
<TABLESONLY=YES | NO>;

Required Arguments

ADD | UPDATE | DELETE
one of these actions must be specified.

LOCATION | METADATA
clarifies whether the action is to apply to the physical security information in the file system, to the metadata objects in the SAS Metadata Server, or to both.

PW=*all-password*
specifies a single password for a metadata-bound library.

ALTER=*alter-password*
assigns, changes, or removes an Alter password from the secured library object and from the data sets in the physical library.

READ=*read-password*
assigns, changes, or removes a Read password from the secured library object and from the data sets in the physical library.

WRITE=*write-password*
assigns, changes, or removes a Write password from the secured library object and from the data sets in the physical library.

Optional Arguments

LIBRARY=*libref*
name of the physical library where the security information is stored.

If the LIBRARY= option is not specified, the physical library from the PROC AUTHLIB statement is used.

Alias: LIB=, DDNAME=, DD=

Restriction: The physical library specified cannot be a concatenated library or temporary library and must be processed by an engine that supports metadata-bound libraries.

SECUREDLIBRARY='*secured-library-name*'
names the secured library object in the SAS Metadata Server.

Alias: SECLIB=

Restriction: The total length of the secured library object pathname including the fully qualified secured folder path cannot exceed 256 characters.

SECUREDFOLDER='*secured-folder-path*'
name of the metadata folder within a **`/System/Secured Libraries`** folder tree where the secured library will be repaired or re-created.

Alias: SECFLDR=

Restriction: The total length of the secured library object pathname including the fully qualified secured folder path cannot exceed 256 characters.

SECUREDLIBRARYGUID=*secured-library-guid*
external identity assigned to the secured library object and stored as part of the security information in the physical library.

Note: The secured library GUID can be found in the REPORT statement output. The GUID that is stored as an external ID in the metadata is not reported, but it matches when correctly configured.

TABLESONLY=YES | NO
specifies whether the REPAIR statement action is applied at the library level or just to the tables. If TABLESONLY=NO, the action is applied to the library and the tables. If TABLESONLY=YES, the action is applied only to the tables. This is especially important for REPAIR because it gives the administrator a way to delete specific secured table objects without deleting the secured library and all secured tables.

Default: NO

Details

CAUTION:
Repairing a metadata-bound library is an advanced task. Make sure you have a current backup (of both metadata and physical data) before you use this statement.

The REPAIR statement is a preproduction feature, which means it is a preliminary release of software that has not completed full development and testing. Because it has not been fully tested, preproduction software should be used with care. After final testing is completed, preproduction software is likely to be offered in a future release as a production-quality component or product.

Use the REPAIR statement to restore metadata-bound library security information or metadata objects that are inadvertently deleted. The administrator can carefully use the REPAIR statement to make some repairs to inconsistencies reported by the REPORT statement. If there are a significant number of groupings in the REPORT listing, it might be more advisable to do the following:

1. Create a new operating system directory and metadata-bound library, and then use SAS Management Console to set appropriate default library permissions for the new secured library object.
2. Access the current library with the AUTHADMIN=YES, AUTHPW= or AUTHALTER=, AUTHWRITE=, and AUTHREAD= options in the LIBNAME statement.
3. Use the SAS COPY procedure to copy the SAS data sets to the new library. Use CONSTRAINT=YES if any data sets have referential integrity constraints. Use SAS Management Console to set any permissions on the secured table objects that differ from those inherited from the secured library object. The following is an example of using the COPY procedure.

Metadata-bound library ABCDE also has data sets EMPLOYEES, EMPINFO, and PRODUCT. The REPORT statement has shown some inconsistencies between the

physical library contents and the corresponding metadata objects. This is an example of a way to resolve these differences.

```
libname klmno "c:\lib2";

proc authlib lib=klmno;
 create securedfolder="Department XYZZY"
        securedlibrary="KLMNOEmps"
        pw=password;
run;
quit;

libname abcde "c:\mylib"
   AUTHADMIN=yes
   AUTHPW=password;

proc copy in=abcde out=klmno ;run;
```

Log 7.1 *Using PROC COPY to Resolve Differences*

```
88   proc copy in=abcde out=klmno ;run;

NOTE: Copying ABCDE.EMPINFO to KLMNO.EMPINFO (memtype=DATA).
NOTE: Data set ABCDE.EMPINFO.DATA has secured table object location information, but the
      secured library object location information that it contains:
           SecuredFolder:      /System/Secured Libraries/Department XYZZY
           SecuredLibrary:     ABCDEEmps
           SecuredLibraryGUID: 38C24AF4-9CF5-458B-8389-52092307007E
      is different from the registered location for the library ABCDE:
           SecuredFolder:
           SecuredLibrary:
           SecuredLibraryGUID:
      The data set might have been copied to this directory with a host copy utility.
NOTE: Permissions are obtained from the secured table and the secured library objects that are
      referenced in the header of the metadata-bound table.
NOTE: Metadata-bound library permissions are used for KLMNO.EMPINFO.DATA.
NOTE: Successfully added new secured table object "EMPINFO.DATA" to the secured library object
      at path "/System/Secured Libraries/Department XYZZY/KLMNOEmps" for data set
      KLMNO.EMPINFO.DATA.
NOTE: There were 5 observations read from the data set ABCDE.EMPINFO.
NOTE: The data set KLMNO.EMPINFO has 5 observations and 6 variables.
NOTE: Copying ABCDE.EMPLOYEES to KLMNO.EMPLOYEES (memtype=DATA).
NOTE: Data set ABCDE.EMPLOYEES.DATA has secured table object location information, but the
      secured library object location information that it contains:
           SecuredFolder:      /System/Secured Libraries/Department XYZZY
           SecuredLibrary:     ABCDEEmps
           SecuredLibraryGUID: 38C24AF4-9CF5-458B-8389-52092307007E
      is different from the registered location for the library ABCDE:
           SecuredFolder:
           SecuredLibrary:
           SecuredLibraryGUID:
      The data set might have been copied to this directory with a host copy utility.
NOTE: Permissions are obtained from the secured table and the secured library objects that are
      referenced in the header of the metadata-bound table.
NOTE: Metadata-bound library permissions are used for KLMNO.EMPLOYEES.DATA.
NOTE: Successfully added new secured table object "EMPLOYEES.DATA" to the secured library
      object at path "/System/Secured Libraries/Department XYZZY/KLMNOEmps" for data set
      KLMNO.EMPLOYEES.DATA.
```

```
NOTE: There were 5 observations read from the data set ABCDE.EMPLOYEES.
NOTE: The data set KLMNO.EMPLOYEES has 5 observations and 6 variables.
NOTE: Copying ABCDE.PRODUCT to KLMNO.PRODUCT (memtype=DATA).
NOTE: Data set ABCDE.PRODUCT.DATA has secured table object location information, but the
      secured library object location information that it contains:
           SecuredFolder:      /System/Secured Libraries/Department XYZZY
           SecuredLibrary:     ABCDEEmps
           SecuredLibraryGUID: 38C24AF4-9CF5-458B-8389-52092307007E
      is different from the registered location for the library ABCDE:
           SecuredFolder:
           SecuredLibrary:
           SecuredLibraryGUID:
      The data set might have been copied to this directory with a host copy utility.
NOTE: Permissions are obtained from the secured table and the secured library objects that are
      referenced in the header of the metadata-bound table.
NOTE: Metadata-bound library permissions are used for KLMNO.PRODUCT.DATA.
NOTE: Successfully added new secured table object "PRODUCT.DATA" to the secured library object
      at path "/System/Secured Libraries/Department XYZZY/KLMNOEmps" for data set
      KLMNO.PRODUCT.DATA.
NOTE: There were 5 observations read from the data set ABCDE.PRODUCT.
NOTE: The data set KLMNO.PRODUCT has 5 observations and 2 variables.
NOTE: PROCEDURE COPY used (Total process time):
      real time           0.14 seconds
      cpu time            0.04 seconds
```

When using the REPAIR statement, one of the ADD, UPDATE, or DELETE actions must be specified. LOCATION, METADATA, or both are used to clarify if the action is to apply to the metadata security information in the file system, to the metadata objects in the SAS Metadata Server, or to both.

One or more TABLES statements can follow the REPAIR statement to perform the same action on the specified data sets. An implicit Tables _ALL_ is used if no TABLES statement follows the REPAIR statement.

Inconsistencies between the metadata security information stored in the operating system files and the secured library object in the SAS Metadata Server that need repair can prevent the assignment of a LIBNAME statement to the physical library. The administrator that owns the physical library and knows the metadata-bound library passwords can perform a library assignment and repair the data by adding the AUTHADMIN=YES option to the LIBNAME statement. Best practice is to use the AUTHADMIN=YES option when performing any REPAIR actions.

CAUTION:

If you lose the password (or passwords) for a metadata-bound library, you cannot unbind the library or change its passwords. Be sure to keep track of passwords that you assign in the CREATE and MODIFY statements.

REPORT Statement

For a specified metadata-bound library, compares physical library contents with corresponding metadata objects (in order to identify any inconsistencies).

Requirement: A connection to the target metadata server. For more requirements, see “Requirements for Using the REPORT Statement” on page 84.

Syntax

PROC AUTHLIB <*option-1* <*...option-n*>>;

REPORT

<LIBRARY=*libref*>;

Optional Argument

LIBRARY=*libref*

name of the physical library where the metadata-bound library is created and the metadata security information is stored.

If the LIBRARY= option is not specified, the physical library from the PROC AUTHLIB statement is used.

Alias: LIB=, DDNAME=, DD=

Restriction: The physical library specified cannot be a concatenated library or temporary library and must be processed by an engine that supports metadata-bound libraries.

Details

Requirements for Using the REPORT Statement

An administrator uses the REPORT statement to identify any inconsistencies between a physical metadata-bound library and its corresponding metadata objects.

In order to use the REPORT statement, you must meet the following criteria:

- The SAS session runs under an account that has host-layer Read access to the target physical library. This is necessary in order to assign the libref.
- The SAS session connects to the metadata server as an identity that has the ReadMetadata permission for the target secured library object and secured table objects.
- If the library has secured library object location information and the secured library object cannot be obtained, you will need to use the AUTHADMIN=YES option in the LIBNAME= statement in order to assign the library.

Reporting Inconsistencies

The REPORT statement is used to report any inconsistencies between the physical library contents and the corresponding metadata objects.

Inconsistencies between the metadata security information in the physical directory, data sets, the secured library, and secured table objects might occur if the metadata or the operating system files are manipulated using nonstandard SAS processing. For example, an operating system data set copied from one directory into a metadata-bound library directory using an operating system copy utility will not have the appropriate security information for that metadata-bound library. Another example is that an administrator might mistakenly delete a secured library or secured table object using SAS Management Console.

The REPORT statement reports the secured table and metadata-bound library security information for each data set in the operating system directory of the library. This data set information is grouped by the metadata-bound library attributes that all the data sets share. If any data sets in the physical library are correctly registered to the secured library object for the library and have the required passwords, those data sets and attributes will be listed as the first grouping in the report. Subsequent groupings are for

data sets with either passwords that differ from the metadata-bound library passwords or whose metadata-bound library security information does not match the metadata-bound library location registered for the operating system directory.

TABLES Statement

Used after a CREATE, MODIFY, REPAIR, and REPORT statement to specify the tables to process a statement action and to specify the current passwords on the data sets, if different from the metadata-bound library passwords.

Default: When no TABLES statement is specified, the TABLES _ALL_ statement is the default behavior.

Requirement:
- The TABLES statement can follow only a CREATE, MODIFY, REPAIR, or REPORT statement.
- A connection to the target metadata server.

Syntax

TABLES *SAS-dataset-1* <*SAS-dataset-n*> | _ALL_ | _NONE_
</>
<PW=*all-password*> | |
<ALTER=*alter-password*>
<READ=*read-password*>
<WRITE=*write-password*>
<MEMTYPE= DATA || VIEW>
< SECUREDTABLEGUID='*secured-table-guid*'>;

Optional Arguments

/
is required if any options are included, such as passwords or MEMTYPE=. Here is an example:

```
tables table-name / pw=password;
```

PW=*all-password*
specifies the current password of the data set.

ALTER=*alter-password*
specifies the current ALTER= password of the data set.

READ=*read-password*
specifies the current READ= password of the data set.

WRITE=*write-password*
specifies the current WRITE= password of the data set.

MEMTYPE= DATA || VIEW
restricts processing to a single member type of DATA or VIEW. If not specified, the default is both types.

DATA
specifies SAS data file member type.

VIEW
specifies SAS view member type.

Alias: MTYPE=, MT=

Default: ALL

SECUREDTABLEGUID=*secured-table-guid*
the external identity of the secured table object that was assigned when the object was created and that is stored as part of the location information in physical data sets that are bound to the secured table object.

Restriction: SECUREDTABLEGUID= option is used only in TABLES statements that follow certain REPAIR statements.

Details

Using the TABLES Statement

CAUTION:

If you lose the password (or passwords) for a metadata-bound library, you cannot unbind the library or change its passwords. Be sure to keep track of passwords that you assign in the CREATE and MODIFY statements.

The TABLES statement is primarily used to specify the current password(s) on data sets when different from the current metadata-bound library required password(s). It usually follows a CREATE or MODIFY statement to make the data set passwords change to the metadata-bound library passwords. See "Example 4: Securing a Library When Existing Data Sets Are Protected with Different Passwords" on page 91.

If you are removing the binding of the physical library to metadata or the physical library is not bound to a secured library, then you might want to modify the data set passwords to some other value. You are not restricted to changing to a common metadata-bound library password. In that case, you might choose to specify both a current and new password separated by a slash / for data sets in a TABLES statement.

TABLES _NONE_ can be used to limit the action of the previous CREATE, MODIFY, or REPAIR statements to the library level and not apply the action to any table. TABLES _ALL_ is the default behavior if no TABLES statement is specified. You might wish to write an explicit TABLES _ALL_ if you want to specify password options to apply to all data sets.

Using the TABLES Statement with the CREATE Statement

The CREATE statement can be followed by one or more TABLES statements to specify current passwords for data sets when different from the metadata-bound library passwords. If the TABLES statement is not used, only two groups of data sets will be bound:

- data sets without passwords
- data sets with passwords matching the metadata-bound library

In effect, omitting TABLES statements is equivalent to specifying one TABLES _ALL_ statement. For more information, see "CREATE Statement" on page 75.

Using the TABLES Statement with the MODIFY Statement

The MODIFY statement can be followed by one or more TABLES statements to specify modifications to passwords in the data sets. If no TABLES statement follows the MODIFY statement, there is an implicit TABLES _ALL_ statement. A separate TABLES statement is required for sets of data sets (tables) that might have different current passwords. For more information, see "MODIFY Statement" on page 76.

Using the TABLES Statement with the REPAIR Statement

When using the REPAIR statement, one of the ADD, UPDATE, or DELETE actions must be specified. LOCATION, METADATA, or both are used to clarify if the action is to apply to the physical security information in file system, to the metadata objects in the SAS Metadata Server, or to both. The REPAIR statement can be followed by one or more TABLES statements to perform the same action on the specified data sets. For more information, see "(Preproduction) REPAIR Statement" on page 79.

Using the TABLES Statement with the REMOVE Statement

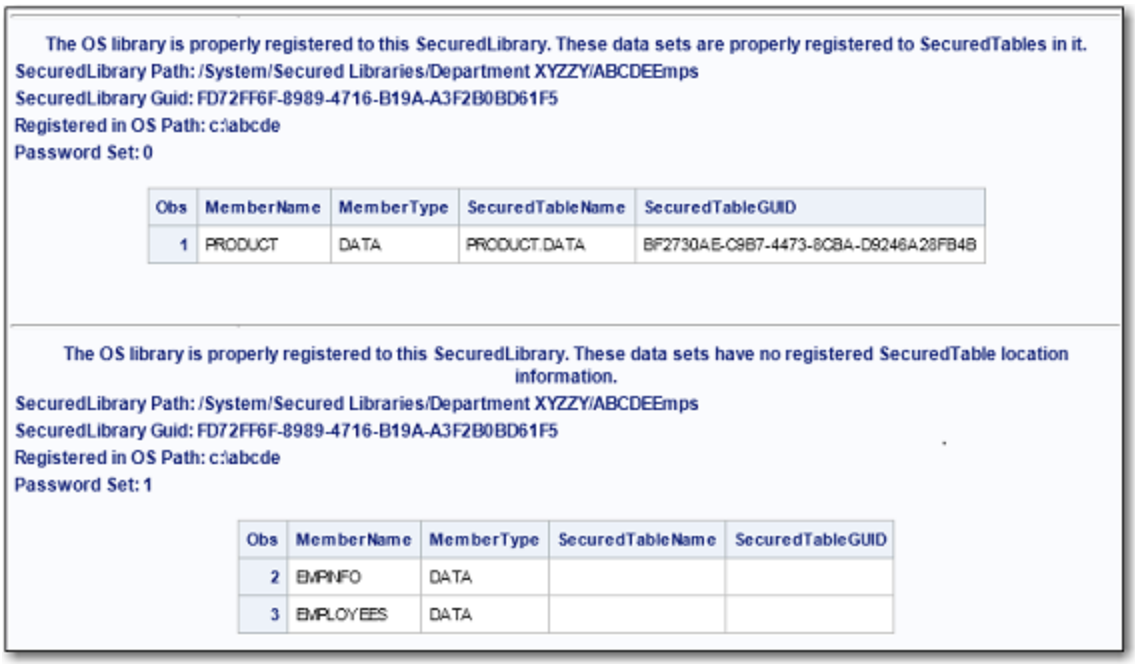
Using a TABLES statement with a REMOVE statement that does not select all tables will produce an ERROR and not execute.

Using the TABLES Statement with the REPORT Statement

The TABLES statement is syntactically accepted with the REPORT statement but has little or no use.

Results: AUTHLIB Procedure

The REPORT statement produces the following output.



The OS library is properly registered to this SecuredLibrary. These data sets are properly registered to SecuredTables in it.
SecuredLibrary Path: /System/Secured Libraries/Department XYZZY/ABCDEEmps
SecuredLibrary Guid: FD72FF6F-8989-4716-B19A-A3F2B0BD61F5
Registered in OS Path: c:\abcde
Password Set: 0

Obs	MemberName	MemberType	SecuredTableName	SecuredTableGUID
1	PRODUCT	DATA	PRODUCT.DATA	BF2730AE-C9B7-4473-8CBA-D9246A28FB4B

The OS library is properly registered to this SecuredLibrary. These data sets have no registered SecuredTable location information.
SecuredLibrary Path: /System/Secured Libraries/Department XYZZY/ABCDEEmps
SecuredLibrary Guid: FD72FF6F-8989-4716-B19A-A3F2B0BD61F5
Registered in OS Path: c:\abcde
Password Set: 1

Obs	MemberName	MemberType	SecuredTableName	SecuredTableGUID
2	EMPINFO	DATA		
3	EMPLOYEES	DATA		

Examples: AUTHLIB Procedure

Example 1: Binding a Physical Library That Contains Unprotected Data Sets

Features: CREATE Statement and Options
SECUREDLIBRARY=
SECUREDFOLDER=
PW=

Details

Here is an example of binding a physical library that contains data sets that do not have passwords. Library ZYXWVUT contains three data sets — EMPLOYEES, EMPINFO, and PRODUCT — that do not have passwords. The library and data sets are bound with the password **secretpw**. The binding is straightforward, as PROC AUTHLIB has unhindered access to the data.

Program

```
proc authlib lib=zyxwvut;
  create securedfolder="Department XYZZY"
    securedlibrary="ZYXWVUTEmps"
    pw=secretpw;

run;
quit;
```

Log

Log 7.2 *Unprotected Data Sets*

```
79   proc authlib lib=zyxwvut;
80
81    create securedfolder="Department XYZZY"
82           securedlibrary="ZYXWVUTEmps"
83           pw=XXXXXXXX;
84
85   run;

NOTE: Successfully created a secured library object for the physical library ZYXWVUT and recorded its
      location as:
           SecuredFolder:      /System/Secured Libraries/Department XYZZY
           SecuredLibrary:     ZYXWVUTEmps
           SecuredLibraryGUID: 1A323C03-A3D8-4A83-9615-2BC2CB9FAAE2
NOTE: Successfully added new secured table object "EMPINFO.DATA" to the secured library object at
      path "/System/Secured Libraries/Department XYZZY/ZYXWVUTEmps" for data set ZYXWVUT.EMPINFO.DATA.
NOTE: The passwords on ZYXWVUT.EMPINFO.DATA were successfully modified.
NOTE: Successfully added new secured table object "EMPLOYEES.DATA" to the secured library object at
      path "/System/Secured Libraries/Department XYZZY/ZYXWVUTEmps" for data set
      ZYXWVUT.EMPLOYEES.DATA.
NOTE: The passwords on ZYXWVUT.EMPLOYEES.DATA were successfully modified.
NOTE: Successfully added new secured table object "PRODUCT.DATA" to the secured library object at
      path "/System/Secured Libraries/Department XYZZY/ZYXWVUTEmps" for data set ZYXWVUT.PRODUCT.DATA.
NOTE: The passwords on ZYXWVUT.PRODUCT.DATA were successfully modified.
86   quit;

NOTE: PROCEDURE AUTHLIB used (Total process time):
      real time           0.68 seconds
      cpu time            0.03 seconds
```

Example 2: Binding a Physical Library That Contains Password-Protected Data Sets

Features: CREATE Statement and Options
SECUREDLIBRARY=
SECUREDFOLDER=
PW=

Details

Library ABCDE also has EMPLOYEES, EMPINFO, and PRODUCT data sets. However, in library ABCDE, the EMPLOYEES and EMPINFO data sets are protected with a Read password of **`abcd`**, a Write password of **`efgh`**, and an Alter password of **`ijkl`** before the library is secured by the statements in the last example. The third data set, PRODUCT, is not protected with passwords.

Program

```
/*  To secure a library that has password-protected data sets  */
/*  that all share the same alter password ijkl with */
/* PROC AUTHLIB CREATE, submit: */
```

```
proc authlib lib=abcde;
 create securedfolder="Department XYZZY"
     securedlibrary="ABCDEEmps"
     pw=secretpw;
run;
quit;
```

Log

The library was bound and the unprotected data set password was set. The protected data sets' passwords did not change because their current passwords were not specified.

Log 7.3 *Password-Protected Data Sets*

```
179  proc authlib lib=abcde;
180
181   create securedfolder="Department XYZZY"
182          securedlibrary="ABCDEEmps"
183          pw=XXXXXXXX;
184
185  run;

NOTE: Successfully created a secured library object for the physical library ABCDE and recorded its
      location as:
           SecuredFolder:      /System/Secured Libraries/Department XYZZY
           SecuredLibrary:     ABCDEEmps
           SecuredLibraryGUID: 4881263D-C346-41F7-AC49-BF9181AF13D2
ERROR: The ALTER password is the most restrictive on ABCDE.EMPINFO.DATA. You must supply its value in
       order to alter or add any passwords.
ERROR: The ALTER password is the most restrictive on ABCDE.EMPLOYEES.DATA. You must supply its value
       in order to alter or add any passwords.
NOTE: Successfully added new secured table object "PRODUCT.DATA" to the secured library object at
      path "/System/Secured Libraries/Department XYZZY/ABCDEEmps" for data set ABCDE.PRODUCT.DATA.
NOTE: The passwords on ABCDE.PRODUCT.DATA were successfully modified.
NOTE: Some statement actions not processed because of errors noted above.
186  quit;

NOTE: The SAS System stopped processing this step because of errors.
NOTE: PROCEDURE AUTHLIB used (Total process time):
      real time           0.14 seconds
      cpu time            0.09 seconds
```

Example 3: Securing a Library When Existing Data Sets Are Protected with the Same Passwords

Features: CREATE Statement and Options
SECUREDLIBRARY=
SECUREDFOLDER=
PW=

Details

This example shows how you could have modified the passwords for the EMPLOYEES and EMPINFO data sets from the preceding example in the PROC AUTHLIB CREATE statement. The EMPLOYEES and EMPINFO data sets are protected with the same

passwords. The Alter password for the data sets,**ijkl**, is specified in the PW= argument before the new password, separated by a slash (/).

Program

```
proc authlib lib=abcde;
 create securedlibrary="ABCDEEmps"
    securedfolder="Department XYZZY"
    pw=ijkl/secretpw;
run;
quit;
```

Log

Log 7.4 *Securing a Library with Data Sets That Are Protected with the Same Passwords*

```
99   proc authlib lib=abcde;
100   create securedlibrary="ABCDEEmps"
101        securedfolder="Department XYZZY" pw=XXXX/XXXXXXXX;
102  run;

NOTE: Successfully created a secured library object for the physical library ABCDE and recorded its
      location as:
           SecuredFolder:      /System/Secured Libraries/Department XYZZY
           SecuredLibrary:     ABCDEEmps
           SecuredLibraryGUID: 87165DCD-3C60-4C7D-BD53-903AAC68CCC7
NOTE: Successfully added new secured table object "EMPINFO.DATA" to the secured library object at
      path "/System/Secured Libraries/Department XYZZY/ABCDEEmps" for data set ABCDE.EMPINFO.DATA.
NOTE: The passwords on ABCDE.EMPINFO.DATA were successfully modified.
NOTE: Successfully added new secured table object "EMPLOYEES.DATA" to the secured library object at
      path "/System/Secured Libraries/Department XYZZY/ABCDEEmps" for data set ABCDE.EMPLOYEES.DATA.
NOTE: The passwords on ABCDE.EMPLOYEES.DATA were successfully modified.
NOTE: Successfully added new secured table object "PRODUCT.DATA" to the secured library object at
      path "/System/Secured Libraries/Department XYZZY/ABCDEEmps" for data set ABCDE.PRODUCT.DATA.
NOTE: The passwords on ABCDE.PRODUCT.DATA were successfully modified.
103  quit;

NOTE: PROCEDURE AUTHLIB used (Total process time):
      real time           0.38 seconds
      cpu time            0.09 seconds
```

Example 4: Securing a Library When Existing Data Sets Are Protected with Different Passwords

Features: CREATE Statement and Options
- SECUREDLIBRARY=
- SECUREDFOLDER=
- PW=
- ALTER=
- READ=
- WRITE=
- TABLES

Details

Library KLMNO has three data sets: EMPLOYEES, EMPINFO, and PRODUCT. In this library, the EMPLOYEES data set is protected with the PW password **lmno**. The EMPINFO data set is protected with a Read password of **abcd**, a Write password of **efgh**, and an Alter password of **ijkl**. The PRODUCT data set is not protected. Because these data sets have different passwords, to change these passwords, you must use TABLES statements with the CREATE statement. When the TABLES statement is used, a TABLES statement must be specified for all tables. This example also uses three passwords to bind the library: READ=ABCDEFGH, WRITE=IJKLMNO, and ALTER=PQRSTUVW.

Program

```
proc authlib lib=klmno;
    create securedlibrary="KLMNOEmps"
       securedfolder="Department XYZZY"
       read=abcdefgh write=ijklmno alter=pqrstuvw;

tables employees / pw=lmno;
tables empinfo / read=abcd write=efgh alter=ijkl;
tables product;

run;
quit;
```

Log

Log 7.5 *Securing a Library with Existing Data Sets That Are Protected with Different Passwords*

```
187  proc authlib lib=klmno;
188
189   create securedlibrary="KLMNOEmps"
190
191        securedfolder="Department XYZZY"
192        read=XXXXXXXX write=XXXXXXX alter=XXXXXXXX;
193
194  tables employees / pw=XXXX;
195  tables empinfo / read=XXXX write=XXXX alter=XXXX;
196  tables product;
197
198  run;

NOTE: Successfully created a secured library object for the physical library KLMNO and recorded its
      location as:
           SecuredFolder:      /System/Secured Libraries/Department XYZZY
           SecuredLibrary:     KLMNOEmps
           SecuredLibraryGUID: ABA4F5FB-F30A-4F20-9C0B-9F05E877B7BD
NOTE: Successfully added new secured table object "EMPLOYEES.DATA" to the secured library object at
      path "/System/Secured Libraries/Department XYZZY/KLMNOEmps" for data set KLMNO.EMPLOYEES.DATA.
NOTE: The passwords on KLMNO.EMPLOYEES.DATA were successfully modified.
NOTE: Successfully added new secured table object "EMPINFO.DATA" to the secured library object at
      path "/System/Secured Libraries/Department XYZZY/KLMNOEmps" for data set KLMNO.EMPINFO.DATA.
NOTE: The passwords on KLMNO.EMPINFO.DATA were successfully modified.
NOTE: Successfully added new secured table object "PRODUCT.DATA" to the secured library object at
      path "/System/Secured Libraries/Department XYZZY/KLMNOEmps" for data set KLMNO.PRODUCT.DATA.
NOTE: The passwords on KLMNO.PRODUCT.DATA were successfully modified.
199  quit;

NOTE: PROCEDURE AUTHLIB used (Total process time):
      real time           0.17 seconds
      cpu time            0.04 seconds
```

Example 5: Changing Passwords on Data Sets

Features: MODIFY Statement and Options
- LIBRARY=
- PW=
- TABLESONLY=

Details

This example shows a different approach for modifying the passwords of existing data sets to match the metadata-bound library passwords. It uses the MODIFY statement. Here, the MODIFY statement is used to modify the data set passwords of the EMPLOYEES and EMPINFO data sets from Example 2 on page 89 to match the metadata-bound library password. The MODIFY statement can also be used to modify the data set passwords of data sets that are copied into a metadata-bound library by operating system commands after the library has been bound. The existing data sets' Alter password is specified in the PW= argument before the metadata-bound password, separated by a slash (/). The TABLESONLY statement specifies to modify table passwords only.

Program

```
proc authlib lib=abcde;
   modify tablesonly=yes pw=ijkl/secretpw;
run;
quit;
```

Log

Log 7.6 *Changing Data Set Passwords*

```
200  proc authlib lib=abcde;
201   modify tablesonly=yes pw=XXXX/XXXXXXXX;
202    run;

NOTE: Successfully added new secured table object "EMPINFO.DATA" to the secured library object at
      path "/System/Secured Libraries/Department XYZZY/ABCDEEmps" for data set ABCDE.EMPINFO.DATA.
NOTE: The passwords on ABCDE.EMPINFO.DATA were successfully modified.
NOTE: Successfully added new secured table object "EMPLOYEES.DATA" to the secured library object at
      path "/System/Secured Libraries/Department XYZZY/ABCDEEmps" for data set ABCDE.EMPLOYEES.DATA.
NOTE: The passwords on ABCDE.EMPLOYEES.DATA were successfully modified.
NOTE: The passwords on ABCDE.PRODUCT.DATA do not require modification.
203  quit;

NOTE: PROCEDURE AUTHLIB used (Total process time):
      real time           0.07 seconds
      cpu time            0.01 seconds
```

Example 6: Changing Metadata-Bound Library Passwords

Features: MODIFY Statement and Options
SECUREDLIBRARY=
SECUREDFOLDER=
PW=

Details

If you believe that the metadata-bound library passwords have been compromised, use the MODIFY statement to modify the library passwords. The following code changes the library passwords and the data set passwords of all data sets in the library that contain the specified passwords.

Program

```
proc authlib lib=abcde;
   modify securedlibrary="ABCDEEmps"
      securedfolder="Department XYZZY"
      pw=secretpw/newpassd;
run;
quit;
```

Log

The preceding code changed the library passwords and the data set passwords of all data sets in the library with those passwords. An error message is displayed for any data set with different passwords.

Log 7.7 *Changing Metadata-bound Library Passwords*

```
217  proc authlib lib=abcde;
218    modify securedlibrary="ABCDEEmps"
219        securedfolder="Department XYZZY"
220        pw=XXXXXXXX/XXXXXXXX;
221
222  run;

NOTE: The passwords for the secured library object with path "/System/Secured Libraries/Department
      XYZZY/ABCDEEmps" were successfully modified."
NOTE: The passwords on ABCDE.EMPINFO.DATA were successfully modified.
NOTE: The passwords on ABCDE.EMPLOYEES.DATA were successfully modified.
NOTE: The passwords on ABCDE.PRODUCT.DATA were successfully modified.
223  quit;

NOTE: PROCEDURE AUTHLIB used (Total process time):
      real time           0.09 seconds
      cpu time            0.03 seconds
```

Example 7: Using the REMOVE Statement

Features: REMOVE Statement and Options
LIBRARY=
PW=

Details

This example shows how to unbind a metadata-bound library. The code deletes metadata that describes the library and its tables from the SAS Metadata Repository, removes security bindings from the physical library and data sets, and removes the assigned password from the data sets, leaving them unprotected. The slash (/) after the password is optional and used to remove or replace the password from the data sets. Note that if a library is bound with Read, Write, and Alter passwords, as in Example 4 on page 91, you must specify all of the passwords, and they must each have a /.

Program

```
proc authlib lib=abcde;
   remove pw=newpassd/;
run;
quit;
```

Log

Log 7.8 *Unbinding a Metadata-Bound Library*

```
25   proc authlib lib=abcde;
26    remove pw=XXXXXXXX/;
27   run;

WARNING: Some or all the passwords on ABCDE.EMPINFO.DATA were removed along with the secured library
         object location, leaving the data set unprotected.
NOTE: The secured table object location for ABCDE.EMPINFO.DATA was successfully removed.
WARNING: Some or all the passwords on ABCDE.EMPLOYEES.DATA were removed along with the secured
         library object location, leaving the data set unprotected.
NOTE: The secured table object location for ABCDE.EMPLOYEES.DATA was successfully removed.
WARNING: Some or all the passwords on ABCDE.PRODUCT.DATA were removed along with the secured library
         object location, leaving the data set unprotected.
NOTE: The secured table object location for ABCDE.PRODUCT.DATA was successfully removed.
NOTE: Successfully deleted the secured library object that was located at:
           SecuredFolder:      /System/Secured Libraries/Department XYZZY
           SecuredLibrary:     ABCDEEmps
           SecuredLibraryGUID: 99F963DC-CD45-4704-96C7-DB9355B65857
NOTE: Successfully deleted the recorded location of the secured library object for the physical
      library ABCDE.
28   quit;

NOTE: PROCEDURE AUTHLIB used (Total process time):
      real time           0.75 seconds
      cpu time            0.26 seconds
```

Example 8: Using the REPORT Statement

Features: Report Statement
LIBRARY=

Details

The following code checks a library's bindings.

Program

```
proc authlib lib=abcde;
  report;
run;
quit;
```

Log

Log 7.9 *Creating a Report*

```
49   proc authlib lib=abcde;
50    report;
51   run;

52   quit;

NOTE: PROCEDURE AUTHLIB used (Total process time):
      real time           0.37 seconds
      cpu time            0.21 seconds
```

Results

The OS library is properly registered to this SecuredLibrary. These data sets are properly registered to SecuredTables in it.
SecuredLibrary Path: /System/Secured Libraries/Department XYZZY/ABCDEEmps
SecuredLibrary Guid: 87165DCD-3C60-4C7D-BD53-903AAC68CCC7
Registered in OS Path: c:\abcde
Password Set: 0

Obs	MemberName	MemberType	SecuredTableName	SecuredTableGUID
1	EMPINFO	DATA	EMPINFO.DATA	7C07DF02-D4A5-4443-BDB3-5D0DCE0D83AA
2	EMPLOYEES	DATA	EMPLOYEES.DATA	F0296103-88A2-4735-B898-D35D0D271E7F
3	PRODUCT	DATA	PRODUCT.DATA	8F2D3F85-3797-42D2-9DF3-71334BC222EB

Example 9: Using the TABLES Statement

Features: TABLE Statements and Options
- PW=
- ALTER=
- READ=
- WRITE=

Details

Library KLMNO has three data sets in the TABLES statements: EMPLOYEES, EMPINFO, and PRODUCT.

Program

```
proc authlib lib=klmno;
    create securedlibrary="KLMNOEmps"
       securedfolder="Department XYZZY"
       read=abcdefgh write=ijklmno alter=pqrstuvw;

tables employees / pw=lmno;
tables empinfo / read=abcd write=efgh alter=ijkl;
tables product;
```

```
run;
quit;
```

Log

Log 7.10 *TABLES Statement*

```
187  proc authlib lib=klmno;
188
189   create securedlibrary="KLMNOEmps"
190
191        securedfolder="Department XYZZY"
192        read=XXXXXXXX write=XXXXXXX alter=XXXXXXXX;
193
194  tables employees / pw=XXXX;
195  tables empinfo / read=XXXX write=XXXX alter=XXXX;
196  tables product;
197
198  run;

NOTE: Successfully created a secured library object for the physical library KLMNO and recorded its
      location as:
           SecuredFolder:      /System/Secured Libraries/Department XYZZY
           SecuredLibrary:     KLMNOEmps
           SecuredLibraryGUID: ABA4F5FB-F30A-4F20-9C0B-9F05E877B7BD
NOTE: Successfully added new secured table object "EMPLOYEES.DATA" to the secured library object at
      path "/System/Secured Libraries/Department XYZZY/KLMNOEmps" for data set KLMNO.EMPLOYEES.DATA.
NOTE: The passwords on KLMNO.EMPLOYEES.DATA were successfully modified.
NOTE: Successfully added new secured table object "EMPINFO.DATA" to the secured library object at
      path "/System/Secured Libraries/Department XYZZY/KLMNOEmps" for data set KLMNO.EMPINFO.DATA.
NOTE: The passwords on KLMNO.EMPINFO.DATA were successfully modified.
NOTE: Successfully added new secured table object "PRODUCT.DATA" to the secured library object at
      path "/System/Secured Libraries/Department XYZZY/KLMNOEmps" for data set KLMNO.PRODUCT.DATA.
NOTE: The passwords on KLMNO.PRODUCT.DATA were successfully modified.
199  quit;

NOTE: PROCEDURE AUTHLIB used (Total process time):
      real time           0.17 seconds
      cpu time            0.04 seconds
```

Chapter 8

CALENDAR Procedure

Overview: CALENDAR Procedure

What Does the CALENDAR Procedure Do?

The CALENDAR procedure displays data from a SAS data set in a monthly calendar format. You can produce a *schedule calendar*, which schedules events around holidays and nonwork periods, or you can produce a *summary calendar*, which summarizes data and displays only one-day events and holidays. When you use PROC CALENDAR you can perform the following tasks:

- schedule work around holidays and other nonwork periods
- display holidays
- process data about *multiple calendars* in a single step and print them in a separate, mixed, or combined format
- apply different holidays, weekly work schedules, and daily work shifts to multiple calendars in a single PROC step
- produce a mean and a sum for variables based on either the number of days in a month or the number of observations

PROC CALENDAR also contains features that are specifically designed to work with PROC CPM in SAS/OR software, a project management scheduling tool.

What Types of Calendars Can PROC CALENDAR Produce?

Simple Schedule Calendar

The following output illustrates the simplest type of schedule calendar that you can produce. This calendar output displays activities that are planned by a banking executive. The following statements produce Output 8.1 on page 101.

```
proc calendar data=allacty;
   start date;
   dur long;
run;
```

For the activities data set shown that is in this calendar, see “Example 1: Schedule Calendar with Holidays: 5-Day Default” on page 138.

The following calendar uses one of the two default calendars, the 24-hour-day, 7-day-week calendar.

Output 8.1 *Simple Schedule Calendar*

The SAS System

July 2002

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
	1 +Dist. Mtg./+	2 +Mgrs. Meeting	3 +Interview/J+ /District 6=+	4	5 +VIP Banquet+	6
7	8 +============Trade Show/Knox============+ +======================Sales Drive/District 6======================+	9	10	11 +Planning Co+ +Mgrs. Meeting	12 +Seminar/Whi+ /District 7=+	13
14	15	16 +Dentist/JW=+	17 +Bank Meetin+	18 +NewsLetter + +Planning Co+	19 +Co. Picnic/+ +Seminar/Whi+	20 +======================Sales Drive/District 7======================+
21	22 +=========Inventors Show/Melvin=========+	23	24 +Birthday/Ma+	25 +==Close Sale +Planning Co+	26 /WYGIX Co.===+	27
28	29	30	31			

Advanced Schedule Calendar

The following output is an advanced schedule calendar produced by PROC CALENDAR. The statements that create this calendar can perform the following tasks:

- schedule activities around holidays
- identify separate calendars
- print multiple calendars in the same report
- apply different holidays to different calendars
- apply different work patterns to different calendars

For an explanation of the program that produces this calendar, see "Example 4: Multiple Schedule Calendars with Atypical Work Shifts (Combined and Mixed Output)" on page 152.

Output 8.2 *Advanced Schedule Calendar*

Well Drilling Work Schedule: Combined Calendars

July 2002

	Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
CAL1		1	2	3	4	5	6
					Independence	+Assemble T>	
						+Lay Power >	
		+========Drill Well/$1,000.00========>				<Drill Well+	
CAL2				+===============Excavate/$3,500.00================>			
	7	8	9	10	11	12	13
CAL1		+===========Build Pump House/$2,000.00============+					
		<=============Assemble Tank/$1,000.00=============+					
		<Lay Power Line/$2,000.0+			+Pour Foundation/$1,500.>		
CAL2		<Excavate/$>	**Vacation**	<Excavate/$+			
	14	15	16	17	18	19	20
CAL1		+======================Install Pump/$500.00======================+					
		<=====Pour Foundation/$1,500.00======+				+Install Pi>	
	21	22	23	24	25	26	27
CAL1		+====================Erect Tower/$2,500.00====================>					
		<Install Pipe/$1,000.00=+					
	28	29	30	31			
CAL1		<Erect Towe+					

Simple Summary Calendar

The following output shows a simple summary calendar that displays the number of meals served daily in a hospital cafeteria:

```
proc calendar data=meals;
   start date;
   sum brkfst lunch dinner;
```

```
    mean brkfst lunch dinner;
run;
```

In a summary calendar, each piece of information for a given day is the value of a variable for that day. The variables can be either numeric or character, and you can format them as necessary. You can use the SUM and MEAN options to calculate sums and means for any numeric variables. These statistics appear in a box below the calendar, as shown in the following output. The data set that is shown in this calendar is created in "Example 7: Summary Calendar with MEAN Values by Observation" on page 169.

Output 8.3 *Simple Summary Calendar*

The SAS System

December 2008

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
	1 123 234 238	2 188 188 198	3 123 183 176	4 200 267 243	5 176 165 177	6
7	8 178 198 187	9 165 176 187	10 187 176 231	11 176 187 222	12 187 187 123	13
14	15 176 165 177	16 156 . 167	17 198 143 167	18 178 198 187	19 165 176 187	20
21	22 187 187 123	23	24	25	26	27
28	29	30	31			

	Sum	Mean
Brkfst	2763	172.688
Lunch	2830	188.667
Dinner	2990	186.875

Advanced Scheduling and Project Management Tasks

For more complex scheduling tasks, consider using the CPM procedure in SAS/OR software. PROC CALENDAR requires that you specify the starting date of each activity.

When the beginning of one task depends on the completion of others and a date slips in a schedule, recalculating the schedule can be time-consuming. Instead of manually recalculating dates, you can use PROC CPM to calculate dates for project activities based on an initial starting date, activity durations, and which tasks are identified as *successors* to others. For an example, see "Example 6: Calculating a Schedule Based on Completion of Predecessor Tasks" on page 161.

Concepts: CALENDAR Procedure

Types of Calendars

PROC CALENDAR can produce two types of calendars: schedule and summary.

Table 8.1 *Summary of Schedule and Summary Calendars*

Type of Calendar	Task	Restriction
Schedule calendar	Schedule activities around holidays and nonwork periods	Cannot calculate sums and means
Schedule calendar	Schedule activities that last more than one day	
Summary calendar	Calculate sums and means	Activities can last only one day

Note: PROC CALENDAR produces a summary calendar if you do not use a DUR or FIN statement in the PROC step.

Schedule Calendar

Definition

A report in calendar format that shows when activities and holidays start and end.

Required Statements

You must supply a START statement and either a DUR or FIN statement. If you do not use a DUR or FIN statement, then PROC CALENDAR assumes that you want to create a summary calendar report.

Table 8.2 *Required Statements*

Statement	Variable Value
"START Statement" on page 135	Starting date of an activity
"DUR Statement" on page 128	Duration of an activity

Statement	Variable Value
"FIN Statement" on page 129	Ending date of an activity

Examples

- "Simple Schedule Calendar" on page 100
- "Advanced Schedule Calendar" on page 102
- "Example 1: Schedule Calendar with Holidays: 5-Day Default" on page 138
- "Example 2: Schedule Calendar Containing Multiple Calendars" on page 143
- "Example 3: Multiple Schedule Calendars with Atypical Work Shifts (Separated Output)" on page 147
- "Example 4: Multiple Schedule Calendars with Atypical Work Shifts (Combined and Mixed Output)" on page 152
- "Example 5: Schedule Calendar, Blank or with Holidays" on page 158
- "Example 6: Calculating a Schedule Based on Completion of Predecessor Tasks" on page 161

Summary Calendar

Definition

A report in calendar format that displays activities and holidays that last only one day and that can provide summary information in the form of sums and means.

Required Statements

You must supply a START statement. This statement identifies the variable in the activities data set that contains an activity's starting date.

Multiple Events on a Single Day

A summary calendar report can display only one activity on a given date. Therefore, if more than one activity has the same START value, then only the last observation that was read is used. In such situations, you might find PROC SUMMARY useful in collapsing your data set to contain one activity per starting date.

Examples

- "Simple Summary Calendar" on page 103
- "Example 7: Summary Calendar with MEAN Values by Observation" on page 169
- "Example 8: Multiple Summary Calendars with Atypical Work Shifts (Separated Output)" on page 173

The Default Calendars

Description

PROC CALENDAR provides two default calendars for simple applications. You can produce calendars without having to specify detailed work shifts and weekly work

patterns if your application can use one of two simple work patterns. Consider using a default calendar if the following conditions are true:

- your application uses a 5-day work week with 8-hour days or a 7-day work week with 24-hour days, as shown in the following table
- you want to print all activities on the same calendar
- you do not need to identify separate calendars

Table 8.3 *Default Calendar Settings and Examples*

Scheduled Work Days	INTERVAL=	Default DAYLENGTH=	Work Period Length	Example
7 (M-Sun)	DAY	24	24-hour days	2
5 (M-F)	WORKDAY	8	8-hour days	1

When You Unexpectedly Produce a Default Calendar

If you want to produce a specialized calendar but do not provide all the necessary information, then PROC CALENDAR attempts to produce a default calendar. These errors cause PROC CALENDAR to produce a calendar with default features:

- If the activities data set does not contain a CALID variable, then PROC CALENDAR produces a default calendar.
- If *both* the holidays and calendar data sets do not contain a CALID variable, then PROC CALENDAR produces a default calendar *even if the activities data set contains a CALID variable*.
- If the activities and calendar data sets contain the CALID variable, but the holidays data set does not, then the default holidays are used.

Examples

- See the 7-day default calendar in Output 8.1 on page 101
- See the 5-day default calendar in "Example 1: Schedule Calendar with Holidays: 5-Day Default" on page 138

Calendars and Multiple Calendars

Definitions

calendar
: a logical entity that represents a weekly work pattern, which consists of weekly work schedules and daily shifts. PROC CALENDAR contains two default work patterns: 5-day week with an 8-hour day or a 7-day week with a 24-hour day. You can also define your own work patterns by using CALENDAR and WORKDAYS data sets.

calendar report
: a report in calendar format that displays activities, holidays, and nonwork periods. A calendar report can contain multiple calendars in one of three formats:

  separate
  : each identified calendar prints on separate output pages.

combined
all identified calendars print on the same output pages and each is identified.

mixed
all identified calendars print on the same output pages but are not identified as belonging to separate calendars.

multiple calendar
a logical entity that represents multiple weekly work patterns.

Why Create Multiple Calendars

Create a multiple calendar if you want to print a calendar report that shows activities that follow different work schedules or different weekly work patterns. For example, a construction project report might need to use different work schedules and weekly work patterns for work crews on different parts of the project.

Another use for multiple calendars is to identify activities so that you can choose to print them in the same calendar report. For example, if you identify activities as belonging to separate departments within a division, then you can choose to print a calendar report that shows all departmental activities on the same calendar.

Finally, using multiple calendars, you can produce separate calendar reports for each calendar in a single step. For example, if activities are identified by department, then you can produce a calendar report that prints the activities of each department on separate pages.

How to Identify Multiple Calendars

Because PROC CALENDAR can process only one data set of each type (activities, holidays, calendar, workdays) in a single PROC step, you must be able to identify for PROC CALENDAR which calendar an activity, holiday, or weekly work pattern belongs to. Use the CALID statement to specify the variable whose values identify the appropriate calendar. This variable can be numeric or character.

You can use the special variable name _CAL_ or you can use another variable name. PROC CALENDAR automatically looks for a variable named _CAL_ in the holiday and calendar data sets, even when the activities data set uses a variable with another name as the CALID variable. Therefore, if you use the name _CAL_ in your holiday and calendar data sets, then you can more easily reuse these data sets in different calendar applications.

Using Holidays or Calendar Data Sets with Multiple Calendars

When using a holidays or calendar data set with multiple calendars, PROC CALENDAR treats the variable values in the following way:

- Every value of the CALID variable that appears in either the holidays or calendar data sets defines a calendar.
- If a CALID value appears in the HOLIDATA= data set but not in the CALEDATA= data set, then the work schedule of the default calendar is used.
- If a CALID value appears in the CALEDATA= data set but not in the HOLIDATA= data set, then the holidays of the default calendar are used.
- If a CALID value does not appear in either the HOLIDATA= or CALEDATA= data set, then the work schedule and holidays of the default calendar are used.
- If the CALID variable is not found in the holiday or calendar data set, then PROC CALENDAR looks for the default variable _CAL_ instead. If neither the CALID

variable nor a _CAL_ variable appears in a data set, then the observations in that data set are applied to a default calendar.

Types of Reports That Contain Multiple Calendars

Because you can associate different observations with different calendars, you can print a calendar report that shows activities that follow different work schedules or different work shifts or that contain different holidays. You can perform the following tasks:

- print separate calendars on the same page and identify each one
- print separate calendars on the same page without identifying them
- print separate pages for each identified calendar

For example, consider a calendar that shows the activities of all departments within a division. Each department can have its own calendar identification value and, if necessary, can have individual weekly work patterns, daily work shifts, and holidays.

If you place activities that are associated with different calendars in the same activities data sets, then you use PROC CALENDAR to produce calendar reports that print the following:

- the schedule and events for each department on a separate pages (separate output)
- the schedule and events for the entire division, each identified by department (combined output)
- the schedule and events for the entire division, but *not* identified by department (mixed output)

The multiple-calendar feature was added specifically to enable PROC CALENDAR to process the output of PROC CPM in SAS/OR software, a project management tool. See "Example 6: Calculating a Schedule Based on Completion of Predecessor Tasks" on page 161.

How to Identify Calendars with the CALID Statement and the Special Variable _CAL_

To identify multiple calendars, you must use the CALID statement to specify the variable whose values identify which calendar an event belongs with. This variable can be numeric or character.

You can use the special variable name _CAL_ or you can use another variable name. PROC CALENDAR automatically looks for a variable named _CAL_ in the holiday and calendar data sets, even when the activities data set uses a variable with another name as the CALID variable. Therefore, if you use the name _CAL_ in your holiday and calendar data sets, then you can more easily reuse these data sets in different calendar applications.

When You Use Holidays or Calendar Data Sets

When you use a holidays or calendar data set with multiple calendars, PROC CALENDAR treats the variable values in the following way:

- Every value of the CALID variable that appears in either the holidays or calendar data sets defines a calendar.
- If a CALID value appears in the HOLIDATA= data set but not in the CALEDATA= data set, then the work schedule of the default calendar is used.
- If a CALID value appears in the CALEDATA= data set but not in the HOLIDATA= data set, then the holidays of the default calendar are used.

- If a CALID value does not appear in either the HOLIDATA= or CALEDATA= data set, then the work schedule and holidays of the default calendar are used.
- If the CALID variable is not found in the holiday or calendar data sets, then PROC CALENDAR looks for the default variable _CAL_ instead. If neither the CALID variable nor a _CAL_ variable appears in a data set, then the observations in that data set are applied to a default calendar.

Examples

- "Example 2: Schedule Calendar Containing Multiple Calendars" on page 143
- "Example 3: Multiple Schedule Calendars with Atypical Work Shifts (Separated Output)" on page 147
- "Example 4: Multiple Schedule Calendars with Atypical Work Shifts (Combined and Mixed Output)" on page 152
- "Example 8: Multiple Summary Calendars with Atypical Work Shifts (Separated Output)" on page 173

Input Data Sets

You might need several data sets to produce a calendar, depending on the complexity of your application. PROC CALENDAR can process one of each of four data sets, as shown in the following table.

Table 8.4 *Four Possible Input Data Sets for PROC CALENDAR*

Data Set	Description	Option
Activities	Each *observation* contains information about a single activity.	"DATA=SAS-data-set" on page 120
Holidays	Each *observation* contains information about a holiday.	"HOLIDATA=SAS-data-set" on page 123
Calendar	Each *observation* defines one weekly work schedule.	"CALEDATA=SAS-data-set" on page 119
Workdays	Each *variable* represents one daily schedule of alternating work and nonwork periods.	"WORKDATA=SAS-data-set" on page 125

Activities Data Set

Purpose

The activities data set, specified with the DATA= option, contains information about the activities to be scheduled by PROC CALENDAR. Each observation describes a single activity.

Requirements and Restrictions

- An activities data set is required. (If you do not specify an activities data set with the DATA= option, then PROC CALENDAR uses the _LAST_ data set.)

- Only one activities data set is allowed.
- The activities data set must always be sorted or indexed by the START variable.
- If you use a CALID (calendar identifier) variable and want to produce output that shows multiple calendars on separate pages, then the activities data set must be sorted by or indexed on the CALID variable and then the START variable.
- If you use a BY statement, then the activities data set must be sorted by or indexed on the BY variables.

Structure

Each *observation* in the activities data set contains information about one activity. One variable must contain the starting date. If you are producing a schedule calendar, then another variable must contain either the activity duration or finishing date. Other variables can contain additional information about an activity.

Table 8.5 *Required Statements*

Variable Content	Statement	Calendar Type
Starting date	"START Statement" on page 135	Schedule Summary
Duration	"DUR Statement" on page 128	Schedule
Finishing date	"FIN Statement" on page 129	Schedule

Multiple Activities per Day in Summary Calendars

A summary calendar can display only one activity on a given date. Therefore, if more than one activity has the same START value, then only the last observation that is read is used. In such situations, you might find PROC SUMMARY useful to collapse your data set to contain one activity per starting date.

Examples

Every example in the Examples section uses an activities data set.

Holidays Data Set

Purpose

You can use a holidays data set, specified with the HOLIDATA= option, to identify the following:

- holidays on your calendar output.
- days that are not available for scheduling work. (In a schedule calendar, PROC CALENDAR does not schedule activities on these days.)

Structure

Each observation in the holidays data set must contain at least the holiday starting date. A holiday lasts only one day unless a duration or finishing date is specified. Supplying a holiday name is recommended, though not required. If you do not specify which variable

contains the holiday name, then PROC CALENDAR uses the word **DATE** to identify each holiday.

Table 8.6 *Required Statements*

Variable Content	Statement
Starting date	"HOLISTART Statement" on page 131
Name	"HOLIVAR Statement" on page 132
Duration	"HOLIDUR Statement" on page 130
Finishing date	"HOLIFIN Statement" on page 131

No Sorting Needed

You do not need to sort or index the holidays data set.

Using SAS Date versus SAS Datetime Values

PROC CALENDAR calculates time using SAS datetime values. Even when your data is in DATE. format, the procedure automatically calculates time in minutes and seconds. Therefore, if you specify only date values, then PROC CALENDAR prints messages similar to the following ones to the SAS log:

```
NOTE: All holidays are assumed to start at the
      time/date specified for the holiday variable
      and last one DTWRKDAY.
WARNING: The units of calculation are SAS datetime
         values while all the holiday variables are
         not. All holidays are converted to SAS
         datetime values.
```

Create a Generic Holidays Data Set

If you have many applications that require PROC CALENDAR output, then consider creating a generic holidays data set that contains standard holidays. You can begin with the generic holidays and add observations that contain holidays or nonwork events specific to an application.

Holidays and Nonwork Periods

Do not schedule holidays during nonwork periods. Holidays that are defined in the HOLIDATA= data set cannot occur during any nonwork periods that are defined in the work schedule. For example, you cannot schedule Sunday as a vacation day if the work week is defined as Monday through Friday. When such a conflict occurs, the holiday is rescheduled to the next available working period following the nonwork day.

Examples

Every example in the Examples section uses a holidays data set.

Calendar Data Set

Purpose

You can use a calendar data set, specified with the CALEDATA= option, to specify work schedules for different calendars.

Structure

Each observation in the calendar data set defines one weekly work schedule. The data set created in the DATA step shown below defines weekly work schedules for two calendars, CALONE and CALTWO.

```
data cale;
   input _sun_ $ _mon_ $ _tue_ $ _wed_ $ _thu_ $ /
         _fri_ $ _sat_ $ _cal_ $ d_length time6.;
   datalines;
holiday workday workday workday workday
workday holiday calone 8:00
holiday shift1 shift1 shift1 shift1
shift2 holiday caltwo 9:00
;
```

These are the variables in this calendar data set:

SUN through _SAT_
: the name of each day of the week that appears in the calendar. The values of these variables contain the name of work shifts. These are the valid values for work shifts:

- **WORKDAY** (the default work shift)
- **HOLIDAY** (a nonwork period)
- names of variables in the WORKDATA= data set (in this example, **SHIFT1** and **SHIFT2**)

CAL
: the CALID (calendar identifier) variable. The values of this variable identify different calendars. If this variable is not present, then the first observation in this data set defines the work schedule that is applied to all calendars in the activities data set.

  If the CALID variable contains a missing value, then the character or numeric value for the default calendar (**DEFAULT** or 0) is used. For more details, see "The Default Calendars" on page 106.

D_LENGTH
: the daylength identifier variable. Values of D_LENGTH indicate the length of the standard workday to be used in calendar calculations. You can set the workday length either by placing this variable in your calendar data set or by using the DAYLENGTH= option.

  Missing values for this variable default to the number of hours specified in the DAYLENGTH= option; if the DAYLENGTH= option is not used, the day length defaults to 24 hours if INTERVAL=DAY, or eight hours if INTERVAL=WORKDAY.

Using Default Work Shifts Instead of a Workdays Data Set

You can use a calendar data set with or without a workdays data set. Without a workdays data set, WORKDAY in the calendar data set is equal to one of two standard workdays, depending on the setting of the INTERVAL= option:

Table 8.7 Workday Settings

INTERVAL=	Work-Shift Start	Day Length
DAY	00:00	24 hours
WORKDAY	9:00	8 hours

You can reset the length of the standard workday with the DAYLENGTH= option or a D_LENGTH variable in the calendar data set. You can define other work shifts in a workdays data set.

Examples

The following examples feature a calendar data set:

- “Example 3: Multiple Schedule Calendars with Atypical Work Shifts (Separated Output)” on page 147
- “Example 4: Multiple Schedule Calendars with Atypical Work Shifts (Combined and Mixed Output)” on page 152
- “Example 7: Summary Calendar with MEAN Values by Observation” on page 169

Workdays Data Set

Purpose

You can use a workdays data set, specified with the WORKDATA= option, to define the daily work shifts named in a CALEDATA= data set.

Use Default Work Shifts or Create Your Own?

You do not need a workdays data set if your application can use one of two default work shifts:

Table 8.8 Default Work Shifts

INTERVAL=	Work-Shift Start	Day Length
DAY	00:00	24 hours
WORKDAY	9:00	8 hours

See the “INTERVAL=DAY | WORKDAY ” on page 124.

Structure

Each variable in the workdays data set contains one daily schedule of alternating work and nonwork periods. For example, this DATA step creates a data set that contains specifications for two work shifts:

```
data work;
   input shift1 time6. shift2 time6.;
   datalines;
7:00  7:00
12:00 11:00
13:00   .
17:00   .
;
```

The variable SHIFT1 specifies a 10-hour workday, with one nonwork period (a lunch hour); the variable SHIFT2 specifies a 4-hour workday with no nonwork periods.

How Missing Values Are Treated

The missing values default to 00:00 in the first observation and to 24:00 in all other observations. Two consecutive values of 24:00 define a zero-length time period, which is ignored.

Examples

See "Example 3: Multiple Schedule Calendars with Atypical Work Shifts (Separated Output)" on page 147

Missing Values in Input Data Sets

The following table summarizes the treatment of missing values for variables in the data sets used by PROC CALENDAR.

Table 8.9 *Treatment of Missing Values in PROC CALENDAR*

Data set	Variable	Treatment of Missing Values
Activities (DATA=)	"_CAL_" on page 113	Default calendar value is used.
	"START Statement" on page 135	Observation is not used.
	"DUR Statement" on page 128	1.0 is used.
	"FIN Statement" on page 129	START value + daylength is used.
	"VAR Statement" on page 136	If a summary calendar or the MISSING option is specified, then the missing value is used. Otherwise, no value is used.
	"SUM Statement" on page 135 , "MEAN Statement" on page 132	0

Data set	Variable	Treatment of Missing Values
Calendar (CALEDATA=)	"_CAL_" on page 113	Default calendar value is used.
	"_SUN_ through _SAT_" on page 113	Corresponding shift for default calendar is used.
	"D_LENGTH" on page 113	If available, DAYLENGTH= value is used, or, if INTERVAL=DAY, 24:00 is used. Otherwise 8:00 is used.
	"SUM Statement" on page 135 , "MEAN Statement" on page 132	0
Holiday (HOLIDATA=)	"_CAL_" on page 113	All holidays apply to all calendars.
	"HOLISTART Statement" on page 131	Observation is not used.
	"HOLIDUR Statement" on page 130	If available, HOLIFIN value is used instead of HOLIDUR value. Otherwise 1.0 is used.
	"HOLIFIN Statement" on page 131	If available, HOLIDUR value is used instead of HOLIFIN value. Otherwise, HOLISTART value + day length is used.
	"HOLIVAR Statement" on page 132	No value is used.
Workdays (WORKDATA=)	any	For the first observation, 00:00 is used. Otherwise, 24:00 is used.

Syntax: CALENDAR Procedure

Requirements: You must use a START statement.

For schedule calendars, you must also use a DUR or a FIN statement.

Tips: If you use a DUR or FIN statement, then PROC CALENDAR produces a schedule calendar.

ODSTIP

You can use the FORMAT, LABEL, and WHERE statements as well as any global statements.

PROC CALENDAR <*option(s)*>;
START *variable*;
BY <DESCENDING> *variable-1*
<…<DESCENDING>*variable-n*>
<NOTSORTED>;
CALID *variable*
</ OUTPUT=COMBINE|MIX|SEPARATE>;
DUR *variable*;
FIN *variable*;
HOLISTART *variable*;
HOLIDUR *variable*;
HOLIFIN *variable*;
HOLIVAR *variable*;
MEAN *variable(s)* </ FORMAT=*format-name*>;
OUTSTART *day-of-week*;
OUTDUR *number-of-days*;
OUTFIN *day-of-week*;
SUM *variable(s)* </ FORMAT=*format-name*>;
VAR *variable(s)*;

Statement	Task	Example
"PROC CALENDAR Statement"	Display data from a SAS data set in a monthly calendar format	Ex. 1, Ex. 3, Ex. 4, Ex. 5, Ex. 6, Ex. 8
"BY Statement"	Process activities separately for each BY group, producing a separate calendar for each value of the BY variable	
"CALID Statement"	Process activities in groups defined by the values of a calendar identifier variable	Ex. 2, Ex. 3, Ex. 4, Ex. 6, Ex. 7, Ex. 8
"DUR Statement"	Specify the variable that contains the duration of each activity	Ex. 1, Ex. 2, Ex. 3, Ex. 4, Ex. 5
"FIN Statement"	Specify the variable in the activities data set that contains the finishing date of each activity	Ex. 6
"HOLIDUR Statement"	Specify the variable in the holidays data set that contains the duration of each holiday for a schedule calendar	Ex. 1, Ex. 5
"HOLIFIN Statement"	Specify the variable in the holidays data set that contains the finishing date of each holiday	
"HOLISTART Statement"	Specify a variable in the holidays data set that contains the starting date of each holiday	Ex. 1, Ex. 5
"HOLIVAR Statement"	Specify a variable in the holidays data set whose values are used to label the holidays	Ex. 1, Ex. 5

Statement	Task	Example
"MEAN Statement"	Specify numeric variables in the activities data set for which mean values are to be calculated for each month	
"OUTDUR Statement"	Specify in days the length of the week to be displayed	
"OUTFIN Statement"	Specify the last day of the week to display in the calendar	Ex. 3, Ex. 4, Ex. 8
"OUTSTART Statement"	Specify the starting day of the week to display in the calendar	Ex. 3, Ex. 4, Ex. 8
"START Statement"	Specify the variable in the activities data set that contains the starting date of each activity	Ex. 1
"SUM Statement"	Specify numeric variables in the activities data set to total for each month	Ex. 8
"VAR Statement"	Specify the variables that you want to display for each activity	Ex. 6

PROC CALENDAR Statement

Displays data from a SAS data set in a monthly calendar format.

Examples:

"Example 1: Schedule Calendar with Holidays: 5-Day Default" on page 138
"Example 3: Multiple Schedule Calendars with Atypical Work Shifts (Separated Output)" on page 147
"Example 4: Multiple Schedule Calendars with Atypical Work Shifts (Combined and Mixed Output)" on page 152
"Example 5: Schedule Calendar, Blank or with Holidays" on page 158
"Example 6: Calculating a Schedule Based on Completion of Predecessor Tasks" on page 161
"Example 8: Multiple Summary Calendars with Atypical Work Shifts (Separated Output)" on page 173

Syntax

PROC CALENDAR <*option(s)*>;

Summary of Optional Arguments

Control printing

FILL
displays all months, even if no activities exist.

FORMCHAR <(*position(s)*)>='*formatting-character(s)*'
defines characters used for outlines, dividers, and so on.

HEADER=SMALL | MEDIUM | LARGE
specifies the type of heading to use in printing the name of the month.

LOCALE
displays month and weekday names in the local language.

MISSING
specifies how to show missing values.

WEEKDAYS
suppresses the display of Saturdays and Sundays in the output.

Control summary information

LEGEND
prints the names of the variables whose values appear in the calendar.

MEANTYPE=NOBS | NDAYS
specifies the type of mean to calculate for each month.

Specify data sets containing

CALEDATA=*SAS-data-set*
weekly work schedules

DATA=*SAS-data-set*
activities

HOLIDATA=*SAS-data-set*
holidays

WORKDATA=*SAS-data-set*
unique shift patterns

Specify time or duration

DATETIME
specifies that START and FIN variables contain values in DATETIME. format.

DAYLENGTH=*hours*
specifies the number of hours in a standard work day.

INTERVAL=DAY | WORKDAY
specifies the units of the DUR and HOLIDUR variables.

Optional Arguments

CALEDATA=*SAS-data-set*
specifies the *calendar data set*, a SAS data set that contains weekly work schedules for multiple calendars.

Default: If you omit the CALEDATA= option, then PROC CALENDAR uses a default work schedule.

Tip: A calendar data set is useful if you are using multiple calendars or a nonstandard work schedule.

See:
"The Default Calendars" on page 106

"Calendar Data Set " on page 113

Examples:
"Example 3: Multiple Schedule Calendars with Atypical Work Shifts (Separated Output)" on page 147

"Example 4: Multiple Schedule Calendars with Atypical Work Shifts (Combined and Mixed Output)" on page 152

DATA=*SAS-data-set*

specifies the *activities data set*, a SAS data set that contains starting dates for all activities and variables to display for each activity. Activities must be sorted or indexed by starting date.

Default: If you omit the DATA= option, then the most recently created SAS data set is used.

See: "Activities Data Set " on page 110

Example: "Example 1: Schedule Calendar with Holidays: 5-Day Default" on page 138

DATETIME

specifies that START and FIN variables contain values in DATETIME. format.

Default: If you omit the DATETIME option, then PROC CALENDAR assumes that the START and FIN values are in the DATE. format.

Examples:

"Example 3: Multiple Schedule Calendars with Atypical Work Shifts (Separated Output)" on page 147

"Example 4: Multiple Schedule Calendars with Atypical Work Shifts (Combined and Mixed Output)" on page 152

"Example 8: Multiple Summary Calendars with Atypical Work Shifts (Separated Output)" on page 173

DAYLENGTH=*hours*

The hour value must be a SAS TIME value.

Default: 24 if INTERVAL=DAY (the default), 8 if INTERVAL=WORKDAY.

Restriction: DAYLENGTH= applies only to schedule calendars.

Interactions:

If you specify the DAYLENGTH= option and the calendar data set contains a D_LENGTH variable, then PROC CALENDAR uses the DAYLENGTH= value only when the D_LENGTH value is missing.

When INTERVAL=DAY and you have no CALEDATA= data set, specifying a DAYLENGTH= value has no effect.

Tips:

The DAYLENGTH= option is useful when you use the DUR statement and your work schedule contains days of varying lengths (for example, a work week of five half-days). In a work week with varying day lengths, you need to set a standard day length to use in calculating duration times. For example, an activity with a duration of 3.0 workdays lasts 24 hours if DAYLENGTH=8:00 or 30 hours if DAYLENGTH=10:00.

Instead of specifying the DAYLENGTH= option, you can specify the length of the working day by using a D_LENGTH variable in the CALEDATA= data set. If you use this method, then you can specify different standard day lengths for different calendars.

See: "Calendar Data Set " on page 113 for more information about setting the length of the standard workday

FILL

displays all months between the first and last activity, start and finish dates inclusive, including months that contain no activities.

Default: If you do not specify FILL, then PROC CALENDAR prints only months that contain *activities*. (Months that contain only *holidays* are not printed.)

Example: "Example 5: Schedule Calendar, Blank or with Holidays" on page 158

FORMCHAR <(*position(s)*)>='*formatting-character(s)*'
defines the characters to use for constructing the outlines and dividers for the cells in the calendar as well as all identifying markers (such as asterisks and arrows) used to indicate holidays or continuation of activities in PROC CALENDAR output.

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 system formatting characters, in order.

Range: PROC CALENDAR uses 17 of the 20 formatting characters that SAS provides.

See:
Table 8.10 on page 122 shows the formatting characters that PROC CALENDAR uses.

Figure 8.1 on page 122 illustrates their use in PROC CALENDAR output.

formatting-character(s)
lists the characters to use for the specified positions. PROC CALENDAR assigns characters in *formatting-character(s)* to *position(s)*, in the order in which they are listed. For example, the following option assigns an asterisk (*) to the 12th position, assigns a single hyphen (-) to the 13th, and does not alter remaining characters:

```
formchar(12 13)='*-'
```

These new settings change the activity line from this:

```
+=================ACTIVITY===============+
```

to this:

```
*-----------------ACTIVITY--------------*
```

Figure 8.1 *Formatting Characters in PROC CALENDAR Output*

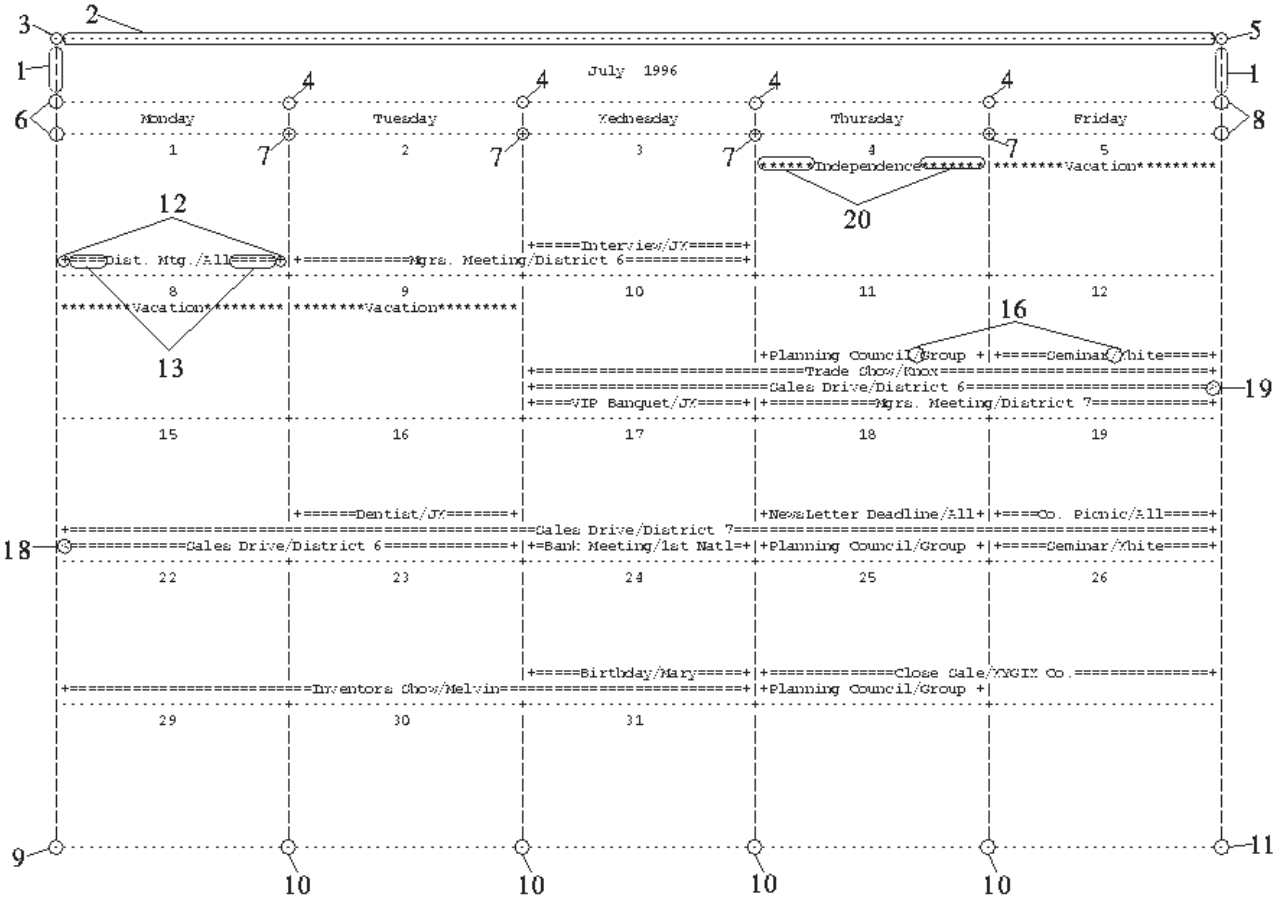


Table 8.10 *Formatting Characters Used by PROC CALENDAR*

Position	Default	Used to Draw
1	|	Vertical bar
2	-	Horizontal bar
3	-	Cell: upper left corner
4	-	Cell: upper middle intersection
5	-	Cell: upper right corner
6	|	Cell: middle left cell side
7	+	Cell: middle middle intersection
8	|	Cell: middle right cell side
9	-	Cell: lower left corner
10	-	Cell: lower middle intersection

Position	Default	Used to Draw
11	-	Cell: lower right corner
12	+	Activity start and finish
13	=	Activity line
16	/	Activity separator
18	<	Activity continuation from
19	>	Activity continuation to
20	*	Holiday marker

Interaction: The SAS system option FORMCHAR= specifies the default formatting characters. The SAS system option defines the entire string of formatting characters. The FORMCHAR= option in a procedure can redefine selected characters.

Tip: You can use any character in *formatting-characters*, including hexadecimal characters. If you use hexadecimal characters, then you must put an **x** after the closing quotation mark. For example, the following option assigns the hexadecimal character 2-D to the third formatting character, the hexadecimal character 7C to the seventh character, and does not alter the remaining characters: **`formchar(3,7)='2D7C'x`**

See: For information about which hexadecimal codes to use for which characters, consult the documentation for your hardware.

HEADER=SMALL | MEDIUM | LARGE
specifies the type of heading to use in printing the name of the month.

SMALL
prints the month and year on one line.

MEDIUM
prints the month and year in a box four lines high.

LARGE
prints the month seven lines high using asterisks (*). The year is included if space is available.

Default: MEDIUM

HOLIDATA=*SAS-data-set*
specifies the *holidays data set*, a SAS data set that contains the holidays that you want to display in the output. One variable must contain the holiday names and another must contain the starting dates for each holiday. PROC CALENDAR marks holidays in the calendar output with asterisks (*) when space permits.

Interaction: Displaying holidays on a calendar requires a holidays data set and a HOLISTART statement. A HOLIVAR statement is recommended for naming holidays. HOLIDUR is required if any holiday lasts longer than one day.

Tip: The holidays data set does not require sorting.

See: “Holidays Data Set ” on page 111

Examples:

"Example 1: Schedule Calendar with Holidays: 5-Day Default" on page 138

"Example 5: Schedule Calendar, Blank or with Holidays" on page 158

INTERVAL=DAY | WORKDAY

specifies the units of the DUR and HOLIDUR variables to one of two default daylengths:

DAY

specifies the values of the DUR and HOLIDUR variables in units of 24-hour days and specifies the default 7-day calendar. For example, a DUR value of 3.0 is treated as 72 hours. The default calendar work schedule consists of seven working days, all starting at 00:00 with a length of 24:00.

WORKDAY

specifies the values of the DUR and HOLIDUR variables in units of 8-hour days. WORKDAY also specifies that the default calendar contains five days a week, Monday through Friday, all starting at 09:00 with a length of 08:00. When WORKDAY is specified, PROC CALENDAR treats the values of the DUR and HOLIDUR variables in units of working days, as defined in the DAYLENGTH= option, the CALEDATA= data set, or the default calendar. For example, if the working day is eight hours long, then a DUR value of 3.0 is treated as 24 hours.

Example: "Example 5: Schedule Calendar, Blank or with Holidays" on page 158

Default: DAY

Interactions:

If there is no CALEDATA= data set, PROC CALENDAR uses the work schedule defined in a default calendar.

The WEEKDAYS option automatically sets the INTERVAL= value to WORKDAY.

See: "Calendars and Multiple Calendars" on page 107 and "Calendar Data Set " on page 113 for more information about the INTERVAL= option and the specification of working days; "The Default Calendars" on page 106

Example: "Example 5: Schedule Calendar, Blank or with Holidays" on page 158

LEGEND

prints the names of the variables whose values appear in the calendar. This identifying text, or legend box, appears at the bottom of the page for each month if space permits. Otherwise, it is printed on the following page. PROC CALENDAR identifies each variable by name or by label if one exists. The order of variables in the legend matches their order in the calendar.

Restriction: LEGEND applies only to summary calendars.

Interaction: If you use the SUM and MEAN statements, then the legend box also contains SUM and MEAN values.

Example: "Example 8: Multiple Summary Calendars with Atypical Work Shifts (Separated Output)" on page 173

LOCALE

prints the names of months and weekdays in the language that is indicated by the value of the LOCALE= SAS system option. The LOCALE option in PROC CALENDAR does not change the starting day of the week.

Default: If LOCALE is not specified, then names of months and weekdays are printed in English.

MEANTYPE=NOBS | NDAYS

specifies the type of mean to calculate for each month.

NOBS
calculates the mean over the number of *observations* displayed in the month.

NDAYS
calculates the mean over the number of *days* displayed in the month.

Default: NOBS

Restriction: MEANTYPE= applies only to summary calendars.

Interaction: Normally, PROC CALENDAR displays all days for each month. However, it might omit some days if you use the OUTSTART statement with the OUTDUR or OUTFIN statement.

Example: "Example 7: Summary Calendar with MEAN Values by Observation" on page 169

MISSING
determines how missing values are treated, based on the type of calendar.

Summary Calendar
If there is a day without an activity scheduled, then PROC CALENDAR prints the values of variables for that day by using the SAS or user-defined that is format specified for missing values.

Default: If you omit MISSING, then days without activities contain no values.

Schedule Calendar
variables with missing values appear in the label of an activity, using the format specified for missing values.

Default: If you do not specify MISSING, then PROC CALENDAR ignores missing values in labeling activities.

See: "Missing Values in Input Data Sets" on page 115 for more information about missing values

WEEKDAYS
suppresses the display of Saturdays and Sundays in the output. It also specifies that the value of the INTERVAL= option is WORKDAY.

```
  proc calendar weekdays;
    start date;
run;
proc calendar interval=workday;
    start date;
outstart monday;
outfin friday;
run;
```

Default: If you omit WEEKDAYS, then the calendar displays all seven days.

Tip: The WEEKDAYS option is an alternative to using the combination of INTERVAL=WORKDAY and the OUTSTART and OUTFIN statements, as shown here:

Example: "Example 1: Schedule Calendar with Holidays: 5-Day Default" on page 138

WORKDATA=*SAS-data-set*
specifies the *workdays data set*, a SAS data set that defines the work pattern during a standard working day. Each numeric variable in the workdays data set denotes a unique work-shift pattern during one working day.

Tip: The workdays data set is useful in conjunction with the calendar data set.

See: "Workdays Data Set " on page 114 and "Calendar Data Set " on page 113

Examples:

"Example 3: Multiple Schedule Calendars with Atypical Work Shifts (Separated Output)" on page 147

"Example 4: Multiple Schedule Calendars with Atypical Work Shifts (Combined and Mixed Output)" on page 152

BY Statement

Processes activities separately for each BY group, producing a separate calendar for each value of the BY variable.

Supports: Summary and schedule calendars

See: "CALID Statement" on page 127

"BY" on page 36 (main discussion)

Example: "Example 1: Schedule Calendar with Holidays: 5-Day Default" on page 138

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, but the observations in the data set must be sorted by all the variables that you specify or have an appropriate index. 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).

Details

When you use the CALID statement, you can process activities that apply to different calendars, indicated by the value of the CALID variable. Because you can specify only one CALID variable, however, you can create only one level of grouping. For example, if you want a calendar report to show the activities of several departments within a company, then you can identify each department with the value of the CALID variable and produce calendar output that shows the calendars for all departments.

When you use a BY statement, however, you can further divide activities into related groups. For example, you can print calendar output that groups departmental calendars by division. The observations for activities must contain a variable that identifies which department an activity belongs to and a variable that identifies the division that a

department resides in. Specify the variable that identifies the department with the CALID statement. Specify the variable that identifies the division with the BY statement.

CALID Statement

Processes activities in groups defined by the values of a calendar identifier variable.

Supports: Summary and schedule calendars

Tip: Useful for producing multiple schedule calendars and for use with SAS/OR software.

See: "Calendar Data Set " on page 113

Examples: "Example 2: Schedule Calendar Containing Multiple Calendars" on page 143
"Example 3: Multiple Schedule Calendars with Atypical Work Shifts (Separated Output)" on page 147
"Example 4: Multiple Schedule Calendars with Atypical Work Shifts (Combined and Mixed Output)" on page 152
"Example 6: Calculating a Schedule Based on Completion of Predecessor Tasks" on page 161
"Example 7: Summary Calendar with MEAN Values by Observation" on page 169
"Example 8: Multiple Summary Calendars with Atypical Work Shifts (Separated Output)" on page 173

Syntax

CALID *variable*
</ OUTPUT=COMBINE|MIX|SEPARATE>;

Required Argument

variable
a character or numeric variable that identifies which calendar an observation contains data for.

Requirement: If you specify the CALID variable, then both the activities and holidays data sets must contain this variable. If either of these data sets does not contain the CALID variable, then a default calendar is used.

Interaction: SAS/OR software uses this variable to identify which calendar an observation contains data for.

Tip: You do not need to use a CALID statement to create this variable. You can include the default variable _CALID_ in the input data sets.

See: "Calendar Data Set " on page 113

Optional Argument

OUTPUT=COMBINE|MIX|SEPARATE
controls the amount of space required to display output for multiple calendars.

COMBINE
produces one page for each month that contains activities and subdivides each day by the CALID value.

Restriction: The input data must be sorted by or indexed on the START variable.

Examples:

"Example 2: Schedule Calendar Containing Multiple Calendars" on page 143

"Example 4: Multiple Schedule Calendars with Atypical Work Shifts (Combined and Mixed Output)" on page 152

MIX

produces one page for each month that contains activities and does not identify activities by the CALID value.

Restriction: The input data must be sorted by or indexed on the START variable.

Tip: MIX requires the least space for output.

Example: "Example 4: Multiple Schedule Calendars with Atypical Work Shifts (Combined and Mixed Output)" on page 152

SEPARATE

produces a separate page for each value of the CALID variable.

Restriction: The input data must be sorted by the CALID variable and then by the START variable or must contain an appropriate composite index.

Examples:

"Example 3: Multiple Schedule Calendars with Atypical Work Shifts (Separated Output)" on page 147

"Example 8: Multiple Summary Calendars with Atypical Work Shifts (Separated Output)" on page 173

"Example 7: Summary Calendar with MEAN Values by Observation" on page 169

Default: COMBINE

DUR Statement

Specifies the variable that contains the duration of each activity.

Alias: DURATION

Interaction: If you use both a DUR and a FIN statement, then DUR is ignored.

Supports: Schedule calendars

Tip: To produce a schedule calendar, you must use either a DUR or FIN statement.

Examples:

"Example 1: Schedule Calendar with Holidays: 5-Day Default" on page 138

"Example 2: Schedule Calendar Containing Multiple Calendars" on page 143

"Example 3: Multiple Schedule Calendars with Atypical Work Shifts (Separated Output)" on page 147

"Example 4: Multiple Schedule Calendars with Atypical Work Shifts (Combined and Mixed Output)" on page 152

"Example 5: Schedule Calendar, Blank or with Holidays" on page 158

Syntax

DUR *variable*;

Required Argument

variable
contains the duration of each activity in a schedule calendar.

Range: The duration can be a real or integral value.

Restriction: This variable must be in the activities data set.

See: For more information about activity durations, see "Activities Data Set " on page 110 and "Calendar Data Set " on page 113

Details

Duration is measured inclusively from the start of the activity (as given in the START variable). In the output, any activity that lasts part of a day is displayed as lasting a full day.

The INTERVAL= option in a PROC CALENDAR statement automatically sets the unit of the duration variable, depending on its own value as follows:

Table 8.11 *INTERVAL= Settings*

INTERVAL=	Default Length of the Duration Unit
DAY (the default)	24 hours
WORKDAY	8 hours

You can override the default length of a duration unit by using one of the following:

- the DAYLENGTH= option
- a D_LENGTH variable in the CALEDATA= data set

FIN Statement

Specifies the variable in the activities data set that contains the finishing date of each activity.

Alias: FINISH

Interaction: If you use both a FIN and a DUR statement, then FIN is used.

Supports: Schedule calendars

Tip: To produce a schedule calendar, you must use either a FIN or DUR statement.

Example: "Example 6: Calculating a Schedule Based on Completion of Predecessor Tasks" on page 161

Syntax

FIN *variable*;

Required Argument

variable
contains the finishing date of each activity.

Restrictions:

The values of *variable* must be either SAS date or datetime values.

If the FIN variable contains datetime values, then you must specify the DATETIME option in the PROC CALENDAR statement.

Both the START and FIN variables must have matching formats. For example, if one contains datetime values, then so must the other.

HOLIDUR Statement

Specifies the variable in the holidays data set that contains the duration of each holiday for a schedule calendar.

Alias: HOLIDURATION

Default: If you do not use a HOLIDUR or HOLIFIN statement, then all holidays last one day.

Restriction: Cannot use with a HOLIFIN statement.

Supports: Schedule calendars

Examples: "Example 1: Schedule Calendar with Holidays: 5-Day Default" on page 138

"Example 5: Schedule Calendar, Blank or with Holidays" on page 158

Syntax

HOLIDUR *variable*;

Required Argument

variable

contains the duration of each holiday.

Range: The duration can be a real or integral value.

Restriction: This variable must be in the holidays data set.

Examples:

"Example 3: Multiple Schedule Calendars with Atypical Work Shifts (Separated Output)" on page 147

"Example 8: Multiple Summary Calendars with Atypical Work Shifts (Separated Output)" on page 173

Details

- If you use both the HOLIFIN and HOLIDUR statements, then PROC CALENDAR uses the HOLIFIN variable value to define each holiday's duration.
- Set the *unit* of the holiday duration variable in the same way that you set the unit of the duration variable; use either the INTERVAL= and DAYLENGTH= options or the CALEDATA= data set.
- Duration is measured inclusively from the start of the holiday (as given in the HOLISTART variable). In the output, any holiday lasting at least half a day appears as lasting a full day.

HOLIFIN Statement

Specifies the variable in the holidays data set that contains the finishing date of each holiday.

Alias: HOLIFINISH

Default: If you do not use a HOLIFIN or HOLIDUR statement, then all holidays last one day.

Supports: Schedule calendars

Syntax

HOLIFIN *variable*;

Required Argument

variable
contains the finishing date of each holiday.

Restrictions:

This variable must be in the holidays data set.

Values of *variable* must be in either SAS date or datetime values.

If the HOLIFIN variable contains datetime values, then you must specify the DATETIME option in the PROC CALENDAR statement.

If a HOLIFIN statement or a HOLIDUR statement is not specified, then all holidays last one day.

Details

If you use both the HOLIFIN and the HOLIDUR statements, then PROC CALENDAR uses only the HOLIFIN variable.

HOLISTART Statement

Specifies a variable in the holidays data set that contains the starting date of each holiday.

Alias: HOLISTA, HOLIDAY

Requirement: When you use a holidays data set, HOLISTART is required.

Supports: Summary and schedule calendars

Examples: "Example 1: Schedule Calendar with Holidays: 5-Day Default" on page 138
"Example 5: Schedule Calendar, Blank or with Holidays" on page 158

Syntax

HOLISTART *variable*;

Required Argument

variable
contains the starting date of each holiday.

Restrictions:

Values of *variable* must be in either SAS date or datetime values.

If the HOLISTART variable contains datetime values, then specify the DATETIME option in the PROC CALENDAR statement.

Details

- The holidays data set need not be sorted.
- All holidays last only one day, unless you use a HOLIFIN or HOLIDUR statement.
- If two or more holidays occur on the same day, then PROC CALENDAR uses only the first observation.

HOLIVAR Statement

Specifies a variable in the holidays data set whose values are used to label the holidays.

Alias: HOLIVARIABLE, HOLINAME

Default: If you do not use a HOLIVAR statement, then PROC CALENDAR uses the word *DATE* to identify holidays.

Supports: Summary and schedule calendars

Examples: "Example 1: Schedule Calendar with Holidays: 5-Day Default" on page 138

"Example 5: Schedule Calendar, Blank or with Holidays" on page 158

Syntax

HOLIVAR *variable*;

Required Argument

variable

a variable whose values are used to label the holidays. Typically, this variable contains the names of the holidays.

Range: character or numeric.

Restrictions:

This variable must be in the holidays data set.

If a HOLIVAR statement is not specified, then PROC CALENDAR use the word **`DATE`** to identify holidays.

Tip: You can format the HOLIVAR variable as you like.

MEAN Statement

Specifies numeric variables in the activities data set for which mean values are to be calculated for each month.

Supports: Summary calendars

Tip: You can use multiple MEAN statements.

See: "Example 7: Summary Calendar with MEAN Values by Observation" on page 169

Syntax

MEAN *variable(s)* </ FORMAT=*format-name*>;

Required Argument

variable(s)
numeric variable for which mean values are calculated for each month.

Restriction: This variable must be in the activities data set.

Optional Argument

FORMAT=*format-name*
names a SAS or user-defined format to be used in displaying the means requested.

Alias: F=

Default: BEST. format

Example: "Example 7: Summary Calendar with MEAN Values by Observation" on page 169

Details

- The means appear at the bottom of the summary calendar page, if there is room; otherwise they appear on the following page.
- The means appear in the LEGEND box if you specify the LEGEND option.
- PROC CALENDAR automatically displays variables named in a MEAN statement in the calendar output, even if the variables are not named in the VAR statement.

OUTDUR Statement

Specifies in days the length of the week to be displayed.

Alias: OUTDURATION

Requirement: The OUTSTART statement is required.

Syntax

OUTDUR *number-of-days*;

Required Argument

number-of-days
an integer that expresses the length in days of the week to be displayed.

Details

Use either the OUTDUR or OUTFIN statement to supply the procedure with information about the length of the week to display. If you use both, then PROC CALENDAR ignores the OUTDUR statement.

OUTFIN Statement

Specifies the last day of the week to display in the calendar.

Alias: OUTFINISH

Requirement: The OUTSTART statement is required.

See: "Example 8: Multiple Summary Calendars with Atypical Work Shifts (Separated Output)" on page 173

Examples: "Example 3: Multiple Schedule Calendars with Atypical Work Shifts (Separated Output)" on page 147
"Example 4: Multiple Schedule Calendars with Atypical Work Shifts (Combined and Mixed Output)" on page 152
"Example 8: Multiple Summary Calendars with Atypical Work Shifts (Separated Output)" on page 173

Syntax

OUTFIN *day-of-week*;

Required Argument

day-of-week
the name of the last day of the week to display. For example,

```
outfin friday;
```

Details

Use either the OUTFIN or OUTDUR statement to supply the procedure with information about the length of the week to display. If you use both, then PROC CALENDAR uses only the OUTFIN statement.

OUTSTART Statement

Specifies the starting day of the week to display in the calendar.

Alias: OUTSTA

Default: If you do not use OUTSTART, then each calendar week begins with Sunday.

See: "Example 8: Multiple Summary Calendars with Atypical Work Shifts (Separated Output)" on page 173

Examples: "Example 3: Multiple Schedule Calendars with Atypical Work Shifts (Separated Output)" on page 147
"Example 4: Multiple Schedule Calendars with Atypical Work Shifts (Combined and Mixed Output)" on page 152
"Example 8: Multiple Summary Calendars with Atypical Work Shifts (Separated Output)" on page 173

Syntax

OUTSTART *day-of-week*;

Required Argument

day-of-week
the name of the starting day of the week for each week in the calendar. For example,

```
outstart monday;
```

Details

By default, a calendar displays all seven days in a week. Use OUTDUR or OUTFIN, in conjunction with OUTSTART, to control how many days are displayed and which day starts the week.

START Statement

Specifies the variable in the activities data set that contains the starting date of each activity.

Alias: STA, DATE, ID

Requirement: START is required for both summary and schedule calendars.

Example: "Example 1: Schedule Calendar with Holidays: 5-Day Default" on page 138

Syntax

START *variable*;

Required Argument

variable
contains the starting date of each activity.

Restrictions:

This variable must be in the activities data set.

Values of *variable* must be in either SAS date or datetime values.

If you use datetime values, then specify the DATETIME option in the PROC CALENDAR statement.

Both the START and FIN variables must have matching formats. For example, if one contains datetime values, then so must the other.

SUM Statement

Specifies numeric variables in the activities data set to total for each month.

Supports: Summary calendars

Tip: To apply different formats to variables that are being summed, use multiple SUM statements.

Examples: "Example 7: Summary Calendar with MEAN Values by Observation" on page 169

"Example 8: Multiple Summary Calendars with Atypical Work Shifts (Separated Output)" on page 173

Syntax

SUM *variable(s)* </ FORMAT=*format-name*>;

Required Argument

variable(s)
specifies one or more numeric variables to total for each month.

Restriction: This variable must be in the activities data set.

Optional Argument

FORMAT=*format-name*
names a SAS or user-defined format to use in displaying the sums requested.

Alias: F=

Default: BEST. format

Examples:

"Example 7: Summary Calendar with MEAN Values by Observation" on page 169

"Example 8: Multiple Summary Calendars with Atypical Work Shifts (Separated Output)" on page 173

Details

- The sum appears at the bottom of the calendar page, if there is room. Otherwise, it appears on the following page.
- The sum appears in the LEGEND box if you specify the LEGEND option.
- PROC CALENDAR automatically displays variables named in a SUM statement in the calendar output, even if the variables are not named in the VAR statement.

VAR Statement

Specifies the variables that you want to display for each activity.

Alias: VARIABLE

Example: "Example 6: Calculating a Schedule Based on Completion of Predecessor Tasks" on page 161

Syntax

VAR *variable(s)*;

Required Argument

variable(s)
specifies one or more variables that you want to display in the calendar.

Range: The values of *variable* can be either character or numeric.

Restriction: These variables must be in the activities data set.

Tip: You can apply a format to this variable.

Details

When VAR Is Not Used

If you do not use a VAR statement, then the procedure displays all variables in the activities data set in the order in which they occur in the data set, except for the BY, CALID, START, DUR, and FIN variables. However, not all variables are displayed if the LINESIZE= and PAGESIZE= settings do not allow enough space in the calendar.

Display of Variables

- PROC CALENDAR displays variables in the order in which they appear in the VAR statement. Not all variables are displayed, however, if the LINESIZE= and PAGESIZE= settings do not allow enough space in the calendar.
- PROC CALENDAR also displays any variable named in a SUM or MEAN statement for each activity in the calendar output. It displays the variable even if you do not name that variable in a VAR statement.

Results: CALENDAR Procedure

What Affects the Quantity of PROC CALENDAR Output

The quantity of printed calendar output depends on the following variables:

- the range of dates in the activities data set
- whether the FILL option is specified
- the BY statement
- the CALID statement

PROC CALENDAR always prints one calendar for every month that contains any activities. If you specify the FILL option, then the procedure prints every month between the first and last activities, including months that contain no activities. Using the BY statement prints one set of output for each BY value. Using the CALID statement with OUTPUT=SEPARATE prints one set of output for each value of the CALID variable.

How Size Affects the Format of PROC CALENDAR Output

PROC CALENDAR always attempts to fit the calendar within a single page, as defined by the SAS system options PAGESIZE= and LINESIZE=. If the PAGESIZE= and LINESIZE= values do not allow sufficient room, then PROC CALENDAR might print the legend box on a separate page. If necessary, PROC CALENDAR truncates or omits values to make the output fit the page and prints messages to that effect in the SAS log.

What Affects the Lines That Show Activity Duration

In a schedule calendar, the duration of an activity is shown by a continuous line through each day of the activity. Values of variables for each activity are printed on the same

line, separated by slashes (/). Each activity begins and ends with a plus sign (+). If an activity continues from one week to the next, then PROC CALENDAR displays arrows (< >) at the points of continuation.

The length of the activity lines depends on the amount of horizontal space available. You can increase the length by specifying the following variables:

- a larger line size with the LINESIZE= option in the OPTIONS statement
- the WEEKDAYS option to suppress the printing of Saturday and Sunday, which provides more space for Monday through Friday

Customizing the Calendar Appearance

PROC CALENDAR uses 17 of the 20 SAS formatting characters to construct the outline of the calendar and to print activity lines and to indicate holidays. You can use the FORMCHAR= option to customize the appearance of your PROC CALENDAR output by substituting your own characters for the default. See Figure 8.1 on page 122 and Table 8.10 on page 122.

If your printer supports an *extended character set* (one that includes graphics characters in addition to the regular alphanumeric characters), then you can greatly improve the appearance of your output by using the FORMCHAR= option to redefine formatting characters with hexadecimal characters. For information about which hexadecimal codes to use for which characters, consult the documentation for your hardware. For an example of assigning hexadecimal values, see "formatting-character(s)" on page 121.

Portability of ODS Output with PROC CALENDAR

Under certain circumstances, using PROC CALENDAR with the Output Delivery System produces files that are not portable. If the SAS system option FORMCHAR= in your SAS session uses nonstandard line-drawing characters, then the output might include strange characters instead of lines in operating environments in which the SAS Monospace font is not installed. To avoid this problem, specify the following OPTIONS statement before executing PROC CALENDAR:

```
options formchar="|----|+|---+=|-/\<>*";
```

Examples: CALENDAR Procedure

Example 1: Schedule Calendar with Holidays: 5-Day Default

Features: PROC CALENDAR statement options
- DATA=
- HOLIDATA=
- WEEKDAYS

Other statements
- DUR statement
- HOLISTART statement
- HOLIVAR statement
- HOLIDUR statement
- START statement

Other features
- PROC SORT statement
- BY statement
- 5-day default calendar

Details

This example does the following:

- creates a schedule calendar
- uses one of the two default work patterns: 8-hour day, 5-day week
- schedules activities around holidays
- displays a 5-day week

Program

```
data allacty;
   input date : date7. event $ 9-36 who $ 37-48 long;
   datalines;
01JUL02 Dist. Mtg.                  All          1
17JUL02 Bank Meeting                1st Natl     1
02JUL02 Mgrs. Meeting               District 6   2
11JUL02 Mgrs. Meeting               District 7   2
03JUL02 Interview                   JW           1
08JUL02 Sales Drive                 District 6   5
15JUL02 Sales Drive                 District 7   5
08JUL02 Trade Show                  Knox         3
22JUL02 Inventors Show              Melvin       3
11JUL02 Planning Council            Group II     1
18JUL02 Planning Council            Group III    1
25JUL02 Planning Council            Group IV     1
12JUL02 Seminar                     White        1
19JUL02 Seminar                     White        1
18JUL02 NewsLetter Deadline         All          1
05JUL02 VIP Banquet                 JW           1
19JUL02 Co. Picnic                  All          1
16JUL02 Dentist                     JW           1
24JUL02 Birthday                    Mary         1
25JUL02 Close Sale                  WYGIX Co.    2
;

data hol;
   input date : date7. holiday $ 11-25 holilong @27;
   datalines;
05jul02   Vacation        3
04jul02   Independence    1
;

proc sort data=allacty;
   by date;
run;

options formchar="|----|+|---+=|-/\<>*";

proc calendar data=allacty holidata=hol weekdays;
```

```
   start date;
   dur long;

   holistart date;
   holivar holiday;
   holidur holilong;

   title1 'Summer Planning Calendar:  Julia Cho';
   title2 'President, Community Bank';
run;
```

Program Description

Create the activities data set. ALLACTY contains both personal and business activities information for a bank president.

```
data allacty;
   input date : date7. event $ 9-36 who $ 37-48 long;
   datalines;
01JUL02 Dist. Mtg.                  All          1
17JUL02 Bank Meeting                1st Natl     1
02JUL02 Mgrs. Meeting               District 6   2
11JUL02 Mgrs. Meeting               District 7   2
03JUL02 Interview                   JW           1
08JUL02 Sales Drive                 District 6   5
15JUL02 Sales Drive                 District 7   5
08JUL02 Trade Show                  Knox         3
22JUL02 Inventors Show              Melvin       3
11JUL02 Planning Council            Group II     1
18JUL02 Planning Council            Group III    1
25JUL02 Planning Council            Group IV     1
12JUL02 Seminar                     White        1
19JUL02 Seminar                     White        1
18JUL02 NewsLetter Deadline         All          1
05JUL02 VIP Banquet                 JW           1
19JUL02 Co. Picnic                  All          1
16JUL02 Dentist                     JW           1
24JUL02 Birthday                    Mary         1
25JUL02 Close Sale                  WYGIX Co.    2
;
```

Create the holidays data set.

```
data hol;
   input date : date7. holiday $ 11-25 holilong @27;
   datalines;
05jul02   Vacation        3
04jul02   Independence    1
;
```

Sort the activities data set by the variable that contains the starting date. You are not required to sort the holidays data set.

```
proc sort data=allacty;
   by date;
run;
```

Set the FORMCHAR option. Setting FORMCHAR to this exact string renders better HTML output when it is viewed outside of the SAS environment where SAS Monospace fonts are not available.

```
options formchar="|----|+|---+=|-/\<>*";
```

Create the schedule calendar. DATA= identifies the activities data set; HOLIDATA= identifies the holidays data set. WEEKDAYS specifies that a week consists of five eight-hour work days.

```
proc calendar data=allacty holidata=hol weekdays;
```

Specify an activity start date variable and an activity duration variable. The START statement specifies the variable in the activities data set that contains the starting date of the activities; DUR specifies the variable that contains the duration of each activity. Creating a schedule calendar requires START and DUR.

```
   start date;
   dur long;
```

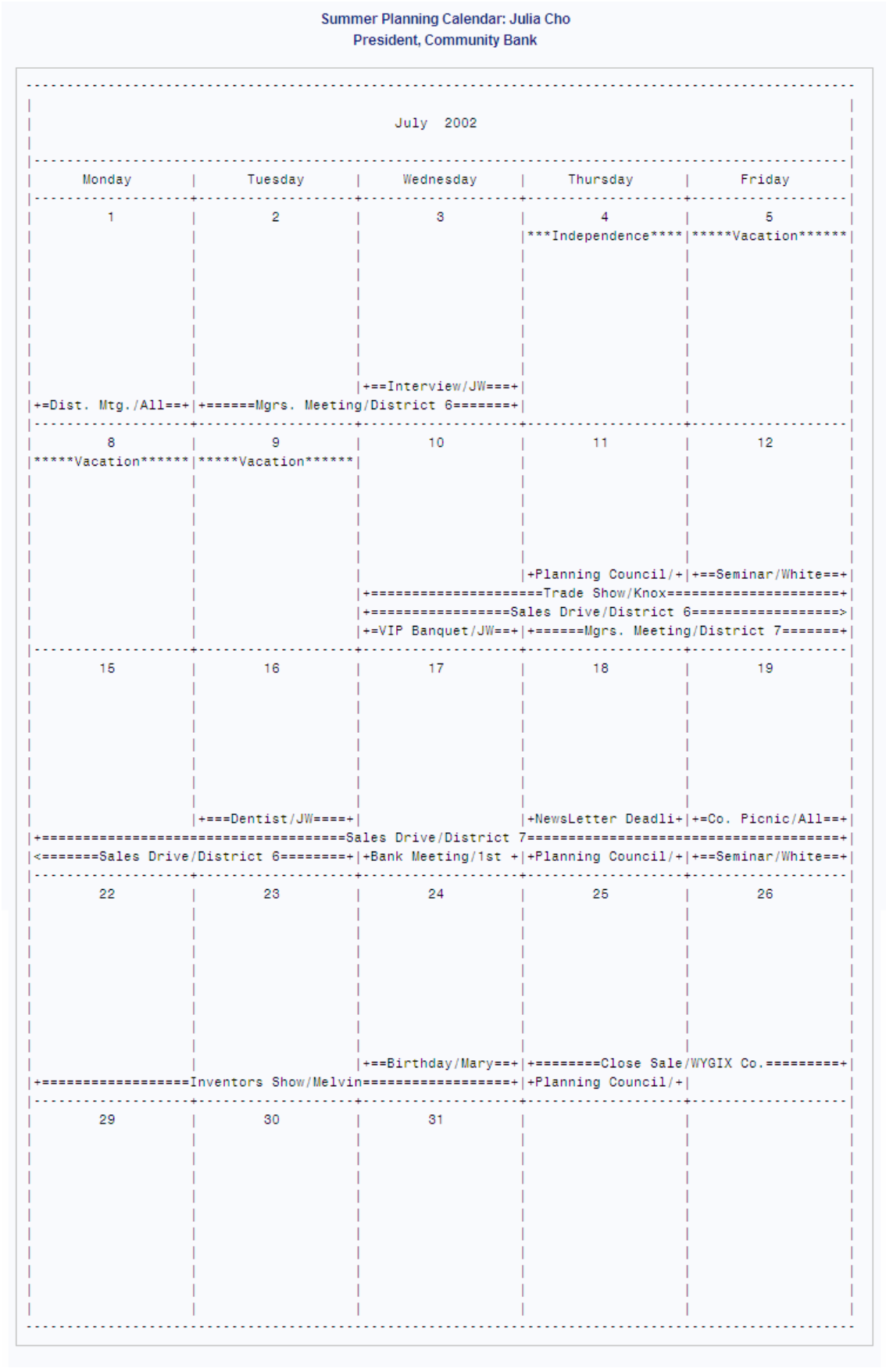
Retrieve holiday information. The HOLISTART, HOLIVAR, and HOLIDUR statements specify the variables in the holidays data set that contain the start date, name, and duration of each holiday, respectively. When you use a holidays data set, HOLISTART is required. Because at least one holiday lasts more than one day, HOLIDUR is required.

```
   holistart date;
   holivar holiday;
   holidur holilong;
```

Specify the titles.

```
   title1 'Summer Planning Calendar:  Julia Cho';
   title2 'President, Community Bank';
run;
```

Output: HTML



Summer Planning Calendar: Julia Cho
President, Community Bank

July 2002

Monday	Tuesday	Wednesday	Thursday	Friday
1	2	3	4 ***Independence****	5 *****Vacation******
		+==Interview/JW===+		
+=Dist. Mtg./All==+	+======Mgrs. Meeting/District 6=======+			
8 *****Vacation******	9 *****Vacation******	10	11	12
			+Planning Council/+	+==Seminar/White==+
		+==========Trade Show/Knox==========+		
		+==========Sales Drive/District 6==========>		
		+=VIP Banquet/JW==+	+======Mgrs. Meeting/District 7=======+	
15	16	17	18	19
	+===Dentist/JW====+		+NewsLetter Deadli+	+=Co. Picnic/All==+
+==========Sales Drive/District 7==========+				
<=======Sales Drive/District 6========+		+Bank Meeting/1st +	+Planning Council/+	+==Seminar/White==+
22	23	24	25	26
		+==Birthday/Mary==+	+========Close Sale/WYGIX Co.=========+	
+==========Inventors Show/Melvin==========+			+Planning Council/+	
29	30	31		

Example 2: Schedule Calendar Containing Multiple Calendars

Features: CALID statement
CAL variable
OUTPUT=COMBINE option
Others
DUR statement
24-hour day, 7-day week

Details

This example builds on Example 1 by identifying activities as belonging to one of two calendars, business or personal. This example does the following:

- produces a schedule calendar report
- prints two calendars on the same output page
- schedules activities around holidays
- uses one of the two default work patterns: 24-hour day, 7-day week
- identifies activities and holidays by calendar name

Program

```
data allacty2;
  input date:date7. happen $ 10-34 who $ 35-47 _CAL_ $ long;
  datalines;
01JUL02  Dist. Mtg.              All          CAL1   1
02JUL02  Mgrs. Meeting           District 6   CAL1   2
03JUL02  Interview               JW           CAL1   1
05JUL02  VIP Banquet             JW           CAL1   1
06JUL02  Beach trip              family       CAL2   2
08JUL02  Sales Drive             District 6   CAL1   5
08JUL02  Trade Show              Knox         CAL1   3
09JUL02  Orthodontist            Meagan       CAL2   1
11JUL02  Mgrs. Meeting           District 7   CAL1   2
11JUL02  Planning Council        Group II     CAL1   1
12JUL02  Seminar                 White        CAL1   1
14JUL02  Co. Picnic              All          CAL1   1
14JUL02  Business trip           Fred         CAL2   2
15JUL02  Sales Drive             District 7   CAL1   5
16JUL02  Dentist                 JW           CAL1   1
17JUL02  Bank Meeting            1st Natl     CAL1   1
17JUL02  Real estate agent       Family       CAL2   1
18JUL02  NewsLetter Deadline     All          CAL1   1
18JUL02  Planning Council        Group III    CAL1   1
19JUL02  Seminar                 White        CAL1   1
22JUL02  Inventors Show          Melvin       CAL1   3
24JUL02  Birthday                Mary         CAL1   1
25JUL02  Planning Council        Group IV     CAL1   1
25JUL02  Close Sale              WYGIX Co.    CAL1   2
27JUL02  Ballgame                Family       CAL2   1
;
```

```
data vac;
   input hdate:date7.  holiday $ 11-25 _CAL_ $ ;
   datalines;
29JUL02   vacation               CAL2
04JUL02   Independence           CAL1
;

proc sort data=allacty2;
   by date;
run;

options formchar="|----|+|---+=|-/\<>*";

proc calendar data=allacty2 holidata=vac;

   calid _CAL_ / output=combine;

   start date ;
   dur long;

   holistart hdate;
   holivar holiday;

   title1 'Summer Planning Calendar:  Julia Cho';
   title2 'President, Community Bank';
   title3 'Work and Home Schedule';
run;
```

Program Description

Create the activities data set and identify separate calendars. ALLACTY2 contains both personal and business activities for a bank president. The _CAL_ variable identifies which calendar an event belongs to.

```
data allacty2;
  input date:date7. happen $ 10-34 who $ 35-47 _CAL_ $ long;
  datalines;
01JUL02  Dist. Mtg.              All          CAL1   1
02JUL02  Mgrs. Meeting           District 6   CAL1   2
03JUL02  Interview               JW           CAL1   1
05JUL02  VIP Banquet             JW           CAL1   1
06JUL02  Beach trip              family       CAL2   2
08JUL02  Sales Drive             District 6   CAL1   5
08JUL02  Trade Show              Knox         CAL1   3
09JUL02  Orthodontist            Meagan       CAL2   1
11JUL02  Mgrs. Meeting           District 7   CAL1   2
11JUL02  Planning Council        Group II     CAL1   1
12JUL02  Seminar                 White        CAL1   1
14JUL02  Co. Picnic              All          CAL1   1
14JUL02  Business trip           Fred         CAL2   2
15JUL02  Sales Drive             District 7   CAL1   5
16JUL02  Dentist                 JW           CAL1   1
17JUL02  Bank Meeting            1st Natl     CAL1   1
17JUL02  Real estate agent       Family       CAL2   1
18JUL02  NewsLetter Deadline     All          CAL1   1
18JUL02  Planning Council        Group III    CAL1   1
19JUL02  Seminar                 White        CAL1   1
22JUL02  Inventors Show          Melvin       CAL1   3
24JUL02  Birthday                Mary         CAL1   1
```

```
25JUL02  Planning Council         Group IV     CAL1   1
25JUL02  Close Sale               WYGIX Co.    CAL1   2
27JUL02  Ballgame                 Family       CAL2   1
;
```

Create the holidays data set and identify which calendar a holiday affects. The _CAL_ variable identifies which calendar a holiday belongs to.

```
data vac;
   input hdate:date7.  holiday $ 11-25 _CAL_ $ ;
   datalines;
29JUL02   vacation               CAL2
04JUL02   Independence           CAL1
;
```

Sort the activities data set by the variable that contains the starting date. When creating a calendar with combined output, you sort only by the activity starting date, not by the CALID variable. You are not required to sort the holidays data set.

```
proc sort data=allacty2;
   by date;
run;
```

Set the FORMCHAR option. Setting FORMCHAR to this exact string renders better HTML output when it is viewed outside of the SAS environment where SAS Monospace fonts are not available.

```
options formchar="|----|+|---+=|-/\<>*";
```

Create the schedule calendar. DATA= identifies the activities data set; HOLIDATA= identifies the holidays data set. By default, the output calendar displays a 7-day week.

```
proc calendar data=allacty2 holidata=vac;
```

Combine all events and holidays on a single calendar. The CALID statement specifies the variable that identifies which calendar an event belongs to. OUTPUT=COMBINE places all events and holidays on the same calendar.

```
   calid _CAL_ / output=combine;
```

Specify an activity start date variable and an activity duration variable. The START statement specifies the variable in the activities data set that contains the starting date of the activities; DUR specifies the variable that contains the duration of each activity. Creating a schedule calendar requires START and DUR.

```
   start date ;
   dur long;
```

Retrieve holiday information. The HOLISTART and HOLIVAR statements specify the variables in the holidays data set that contain the start date and name of each holiday, respectively. HOLISTART is required when you use a holidays data set.

```
   holistart hdate;
   holivar holiday;
```

Specify the titles.

```
   title1 'Summer Planning Calendar:  Julia Cho';
   title2 'President, Community Bank';
```

```
   title3 'Work and Home Schedule';
run;
```

Output: HTML

Summer Planning Calendar: Julia Cho
President, Community Bank
Work and Home Schedule

July 2002

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
	1	2	3	4	5	6
CAL2						+Beach trip>
CAL1	+Dist. Mtg.+	+Mgrs. Meeting/District +	+Interview/+	Independence	+VIP Banque+	
7	8	9	10	11	12	13
CAL2	<Beach trip+	+Orthodonti+				
CAL1	+==========Trade Show/Knox==========+			+Planning C+	+Seminar/Wh+	
	+====================Sales Drive/District 6====================+			+Mgrs. Meeting/District +		
14	15	16	17	18	19	20
CAL2	+==Business trip/Fred===+		+Real estat+			
CAL1		+Dentist/JW+	+Bank Meeti+	+Planning C+	+Seminar/Wh+	
	+Co. Picnic+	+====================Sales Drive/District 7====================+		+NewsLetter+		
21	22	23	24	25	26	27
CAL2						+Ballgame/F+
CAL1	+=======Inventors Show/Melvin========+		+Birthday/M+	+=Close Sale/WYGIX Co.==+		
				+Planning C+		
28	29	30	31			
CAL2	**vacation**					

Example 3: Multiple Schedule Calendars with Atypical Work Shifts (Separated Output)

Features: PROC CALENDAR statement options
- CALEDATA=
- DATETIME
- WORKDATA=

CALID statement
- _CAL_ variable
- OUTPUT=SEPARATE option

Other statements:
- DUR statement
- OUTSTART statement
- OUTFIN statement

Details

This example does the following:

- produces separate output pages for each calendar in a single PROC step
- schedules activities around holidays
- displays an 8-hour day, 5 1/2-day week
- uses separate work patterns and holidays for each calendar

Producing Different Output for Multiple Calendars

This example and "Example 4: Multiple Schedule Calendars with Atypical Work Shifts (Combined and Mixed Output)" on page 152 use the same input data for multiple calendars to produce different output. The only differences in these programs are how the activities data set is sorted and how the OUTPUT= option is set.

Table 8.12 *Sort and OUTPUT= Settings*

Print Options	Sorting Variables	OUTPUT= Settings	Examples
Separate pages for each calendar	Calendar ID and starting date	SEPARATE	3, 8
All activities on the same page and identify each calendar	Starting date	COMBINE	4, 2
All activities on the same page and NOT identify each calendar	Starting date	MIX	4

Program

```
libname well 'SAS-library';
```

```
data well.act;
   input task & $16. dur : 5. date : datetime16.  _cal_ $ cost;
   datalines;
Drill Well          3.50  01JUL02:12:00:00  CAL1   1000
Lay Power Line      3.00  04JUL02:12:00:00  CAL1   2000
Assemble Tank       4.00  05JUL02:08:00:00  CAL1   1000
Build Pump House    3.00  08JUL02:12:00:00  CAL1   2000
Pour Foundation     4.00  11JUL02:08:00:00  CAL1   1500
Install Pump        4.00  15JUL02:14:00:00  CAL1    500
Install Pipe        2.00  19JUL02:08:00:00  CAL1   1000
Erect Tower         6.00  20JUL02:08:00:00  CAL1   2500
Deliver Material    2.00  01JUL02:12:00:00  CAL2    500
Excavate            4.75  03JUL02:08:00:00  CAL2   3500
;

data well.hol;
   input date date. holiday $ 11-25 _cal_ $;
   datalines;
09JUL02   Vacation            CAL2
04JUL02   Independence        CAL1
;

data well.cal;
   input _sun_ $ _sat_ $ _mon_ $ _tue_ $ _wed_ $ _thu_ $
         _fri_ $ _cal_ $;
   datalines;
Holiday Holiday  Workday Workday Workday Workday Workday CAL1
Holiday Halfday  Workday Workday Workday Workday Workday CAL2
;

data well.wor;
   input halfday time5.;
   datalines;
08:00
12:00
;

proc sort data=well.act;
   by _cal_ date;
run;

options formchar="|----|+|---+=|-/\<>*";

proc calendar data=well.act
              holidata=well.hol
              caledata=well.cal
              workdata=well.wor
              datetime;

   calid _cal_ / output=separate;

   start date;
   dur dur;

   holistart date;
   holivar holiday;

   outstart Monday;
   outfin Saturday;
```

```
   title1 'Well Drilling Work Schedule: Separate Calendars';
   format cost dollar9.2;
run;
```

Program Description

Specify a library so that you can permanently store the activities data set.

```
libname well 'SAS-library';
```

Create the activities data set and identify separate calendars. WELL.ACT is a permanent SAS data set that contains activities for a well construction project. The _CAL_ variable identifies the calendar that an activity belongs to.

```
data well.act;
   input task & $16. dur : 5. date : datetime16.  _cal_ $ cost;
   datalines;
Drill Well          3.50  01JUL02:12:00:00  CAL1   1000
Lay Power Line      3.00  04JUL02:12:00:00  CAL1   2000
Assemble Tank       4.00  05JUL02:08:00:00  CAL1   1000
Build Pump House    3.00  08JUL02:12:00:00  CAL1   2000
Pour Foundation     4.00  11JUL02:08:00:00  CAL1   1500
Install Pump        4.00  15JUL02:14:00:00  CAL1    500
Install Pipe        2.00  19JUL02:08:00:00  CAL1   1000
Erect Tower         6.00  20JUL02:08:00:00  CAL1   2500
Deliver Material    2.00  01JUL02:12:00:00  CAL2    500
Excavate            4.75  03JUL02:08:00:00  CAL2   3500
;
```

Create the holidays data set. The _CAL_ variable identifies the calendar that a holiday belongs to.

```
data well.hol;
   input date date. holiday $ 11-25 _cal_ $;
   datalines;
09JUL02   Vacation            CAL2
04JUL02   Independence        CAL1
;
```

Create the calendar data set. Each observation defines the work shifts for an entire week. The _CAL_ variable identifies to which calendar the work shifts apply. CAL1 uses the default 8-hour work shifts for Monday through Friday. CAL2 uses a half day on Saturday and the default 8-hour work shift for Monday through Friday.

```
data well.cal;
   input _sun_ $ _sat_ $ _mon_ $ _tue_ $ _wed_ $ _thu_ $
         _fri_ $ _cal_ $;
   datalines;
Holiday Holiday  Workday Workday Workday Workday Workday CAL1
Holiday Halfday  Workday Workday Workday Workday Workday CAL2
;
```

Create the workdays data set. This data set defines the daily work shifts that are named in the calendar data set. Each variable (not observation) contains one daily schedule of alternating work and nonwork periods. The HALFDAY work shift lasts 4 hours.

```
data well.wor;
   input halfday time5.;
   datalines;
08:00
12:00
;
```

Sort the activities data set by the variables that contain the calendar identification and the starting date, respectively. You are not required to sort the holidays data set.

```
proc sort data=well.act;
   by _cal_ date;
run;
```

Set the FORMCHAR option. Setting FORMCHAR to this exact string renders better HTML output when it is viewed outside of the SAS environment where SAS Monospace fonts are not available.

```
options formchar="|----|+|---+=|-/\<>*";
```

Create the schedule calendar. DATA= identifies the activities data set; HOLIDATA= identifies the holidays data set; CALEDATA= identifies the calendar data set; WORKDATA= identifies the workdays data set. DATETIME specifies that the variable specified with the START statement contains values in SAS datetime format.

```
proc calendar data=well.act
              holidata=well.hol
              caledata=well.cal
              workdata=well.wor
              datetime;
```

Print each calendar on a separate page. The CALID statement specifies that the _CAL_ variable identifies calendars. OUTPUT=SEPARATE prints information for each calendar on separate pages.

```
   calid _cal_ / output=separate;
```

Specify an activity start date variable and an activity duration variable. The START statement specifies the variable in the activities data set that contains the activity starting date; DUR specifies the variable that contains the activity duration. START and DUR are required for a schedule calendar.

```
   start date;
   dur dur;
```

Retrieve holiday information. HOLISTART and HOLIVAR specify the variables in the holidays data set that contain the start date and name of each holiday, respectively. HOLISTART is required when you use a holidays data set.

```
   holistart date;
   holivar holiday;
```

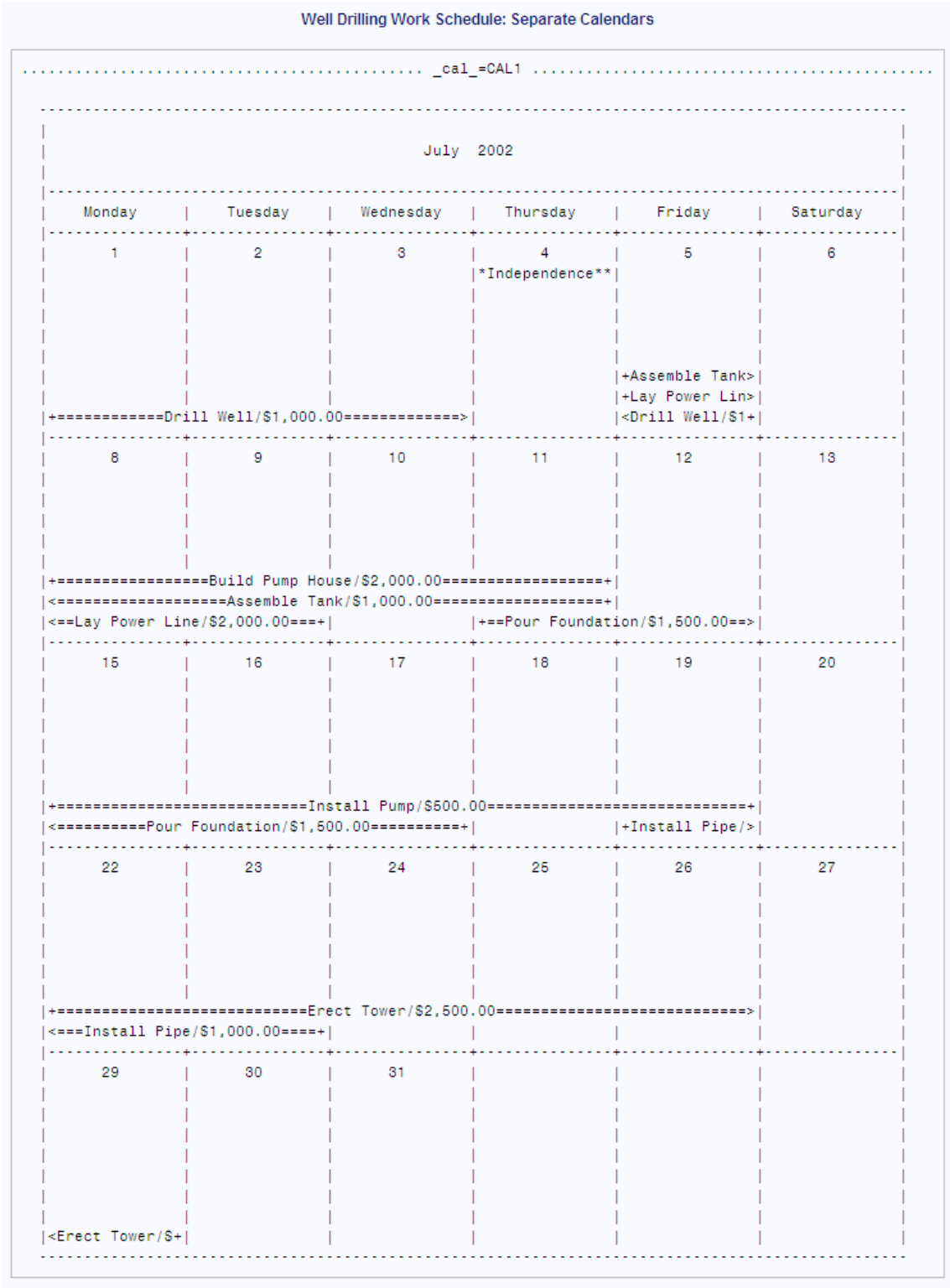
Customize the calendar appearance. OUTSTART and OUTFIN specify that the calendar display a 6-day week, Monday through Saturday.

```
   outstart Monday;
   outfin Saturday;
```

Specify the title and format the Cost variable.

```
    title1 'Well Drilling Work Schedule: Separate Calendars';
    format cost dollar9.2;
run;
```

Output: HTML



Well Drilling Work Schedule: Separate Calendars

```
.......................................... _cal_=CAL2 ..........................................

 -----------------------------------------------------------------------------------------------
 |                                                                                             |
 |                                        July  2002                                           |
 |                                                                                             |
 |---------------------------------------------------------------------------------------------|
 |    Monday     |    Tuesday    |   Wednesday   |   Thursday    |    Friday     |   Saturday    |
 |---------------+---------------+---------------+---------------+---------------+---------------|
 |       1       |       2       |       3       |       4       |       5       |       6       |
 |               |               |               |               |               |               |
 |               |               |               |               |               |               |
 |               |               |               |               |               |               |
 |               |               |               |               |               |               |
 |               |               |               |               |               |               |
 |               |               |+====================Excavate/$3,500.00=====================>|
 |+==========Deliver Material/$500.00===========+|               |               |               |
 |---------------+---------------+---------------+---------------+---------------+---------------|
 |       8       |       9       |      10       |      11       |      12       |      13       |
 |               |***Vacation****|               |               |               |               |
 |               |               |               |               |               |               |
 |               |               |               |               |               |               |
 |               |               |               |               |               |               |
 |               |               |               |               |               |               |
 |               |               |               |               |               |               |
 |<Excavate/$3,5>|               |<Excavate/$3,5+|               |               |               |
 |---------------+---------------+---------------+---------------+---------------+---------------|
 |      15       |      16       |      17       |      18       |      19       |      20       |
 |               |               |               |               |               |               |
 |               |               |               |               |               |               |
 |               |               |               |               |               |               |
 |               |               |               |               |               |               |
 |               |               |               |               |               |               |
 |               |               |               |               |               |               |
 |               |               |               |               |               |               |
 |---------------+---------------+---------------+---------------+---------------+---------------|
 |      22       |      23       |      24       |      25       |      26       |      27       |
 |               |               |               |               |               |               |
 |               |               |               |               |               |               |
 |               |               |               |               |               |               |
 |               |               |               |               |               |               |
 |               |               |               |               |               |               |
 |               |               |               |               |               |               |
 |               |               |               |               |               |               |
 |---------------+---------------+---------------+---------------+---------------+---------------|
 |      29       |      30       |      31       |               |               |               |
 |               |               |               |               |               |               |
 |               |               |               |               |               |               |
 |               |               |               |               |               |               |
 |               |               |               |               |               |               |
 |               |               |               |               |               |               |
 |               |               |               |               |               |               |
 |               |               |               |               |               |               |
 -----------------------------------------------------------------------------------------------
```

Example 4: Multiple Schedule Calendars with Atypical Work Shifts (Combined and Mixed Output)

Features: PROC CALENDAR statement options
CALEDATA=

DATETIME
WORKDATA=

CALID statement
CAL variable
OUTPUT=COMBINE option
OUTPUT=MIXED option

Other statement
DUR statement
OUTSTART statement
OUTFIN statement

Data sets: WELL.ACT
WELL.HOL
WELL.CAL
WELL.WOR

Details

This example does the following:

- produces a schedule calendar
- schedules activities around holidays
- uses separate work patterns and holidays for each calendar
- uses an 8-hour day, 5 1/2-day work week
- displays and identifies multiple calendars on each calendar page (combined output)
- displays *but does not identify* multiple calendars on each calendar page (mixed output)

This example creates both combined and mixed output. Producing combined or mixed calendar output requires only one change to a PROC CALENDAR step: the setting of the OUTPUT= option in the CALID statement. Combined output is produced first, then mixed output.

This example and "Example 3: Multiple Schedule Calendars with Atypical Work Shifts (Separated Output)" on page 147 use the same input data for multiple calendars to produce different output. The only differences in these programs are how the activities data set is sorted and how the OUTPUT= option is set.

Table 8.13 *Sort and OUTPUT= Settings*

Print Options	Sorting Variables	OUTPUT= Settings	Examples
Separate pages for each calendar	Calendar ID and starting date	SEPARATE	3, 8
All activities on the same page and identify each calendar	Starting date	COMBINE	4, 2
All activities on the same page and NOT identify each calendar	Starting date	MIX	4

Program for Combined Characters

```
libname well
'SAS-library';

proc sort data=well.act;
   by date;
run;

options formchar="|----|+|---+=|-/\<>*";

proc calendar data=well.act
              holidata=well.hol
              caledata=well.cal
              workdata=well.wor
              datetime;

   calid _cal_ / output=combine;

   start date;
   dur dur;

   holistart date;
   holivar holiday;

   title1 'Well Drilling Work Schedule: Combined Calendars';
   format cost dollar9.2;
run;
```

Program Description

Specify the SAS library where the activities data set is stored.

```
libname well
'SAS-library';
```

Sort the activities data set by the variable that contains the starting date. Do not sort by the CALID variable when producing combined calendar output.

```
proc sort data=well.act;
   by date;
run;
```

Set the FORMCHAR option. Setting FORMCHAR to this exact string renders better HTML output when it is viewed outside of the SAS environment where SAS Monospace fonts are not available.

```
options formchar="|----|+|---+=|-/\<>*";
```

Create the schedule calendar. DATA= identifies the activities data set; HOLIDATA= identifies the holidays data set; CALEDATA= identifies the calendar data set; WORKDATA= identifies the workdays data set. DATETIME specifies that the variable specified with the START statement contains values in SAS datetime format.

```
proc calendar data=well.act
              holidata=well.hol
              caledata=well.cal
              workdata=well.wor
              datetime;
```

Combine all events and holidays on a single calendar. The CALID statement specifies that the _CAL_ variable identifies the calendars. OUTPUT=COMBINE prints multiple calendars on the same page and identifies each calendar.

```
calid _cal_ / output=combine;
```

Specify an activity start date variable and an activity duration variable. The START statement specifies the variable in the activities data set that contains the starting date of the activities; DUR specifies the variable that contains the duration of each activity. START and DUR are required for a schedule calendar.

```
start date;
dur dur;
```

Retrieve holiday information. HOLISTART and HOLIVAR specify the variables in the holidays data set that contain the start date and name of each holiday, respectively. HOLISTART is required when you use a holidays data set.

```
holistart date;
holivar holiday;
```

Specify the title and format the Cost variable.

```
   title1 'Well Drilling Work Schedule: Combined Calendars';
   format cost dollar9.2;
run;
```

Output for Combined Calendars

Well Drilling Work Schedule: Combined Calendars

July 2002

	Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
		1	2	3	4	5	6
CAL1					Independence	+Assemble T>	
						+Lay Power >	
		+========Drill Well/$1,000.00========>				<Drill Well+	
CAL2				+================Excavate/$3,500.00================>			
		+======Deliver Material/$500.00======+					
	7	8	9	10	11	12	13
CAL1		+==========Build Pump House/$2,000.00==========+					
		<=============Assemble Tank/$1,000.00=============+					
		<Lay Power Line/$2,000.0+			+Pour Foundation/$1,500.>		
CAL2		<Excavate/$>	**Vacation**	<Excavate/$+			
	14	15	16	17	18	19	20
CAL1		+======================Install Pump/$500.00======================+					
		<=====Pour Foundation/$1,500.00======+				+Install Pi>	
	21	22	23	24	25	26	27
CAL1		+====================Erect Tower/$2,500.00====================>					
		<Install Pipe/$1,000.00=+					
	28	29	30	31			
CAL1		<Erect Towe+					

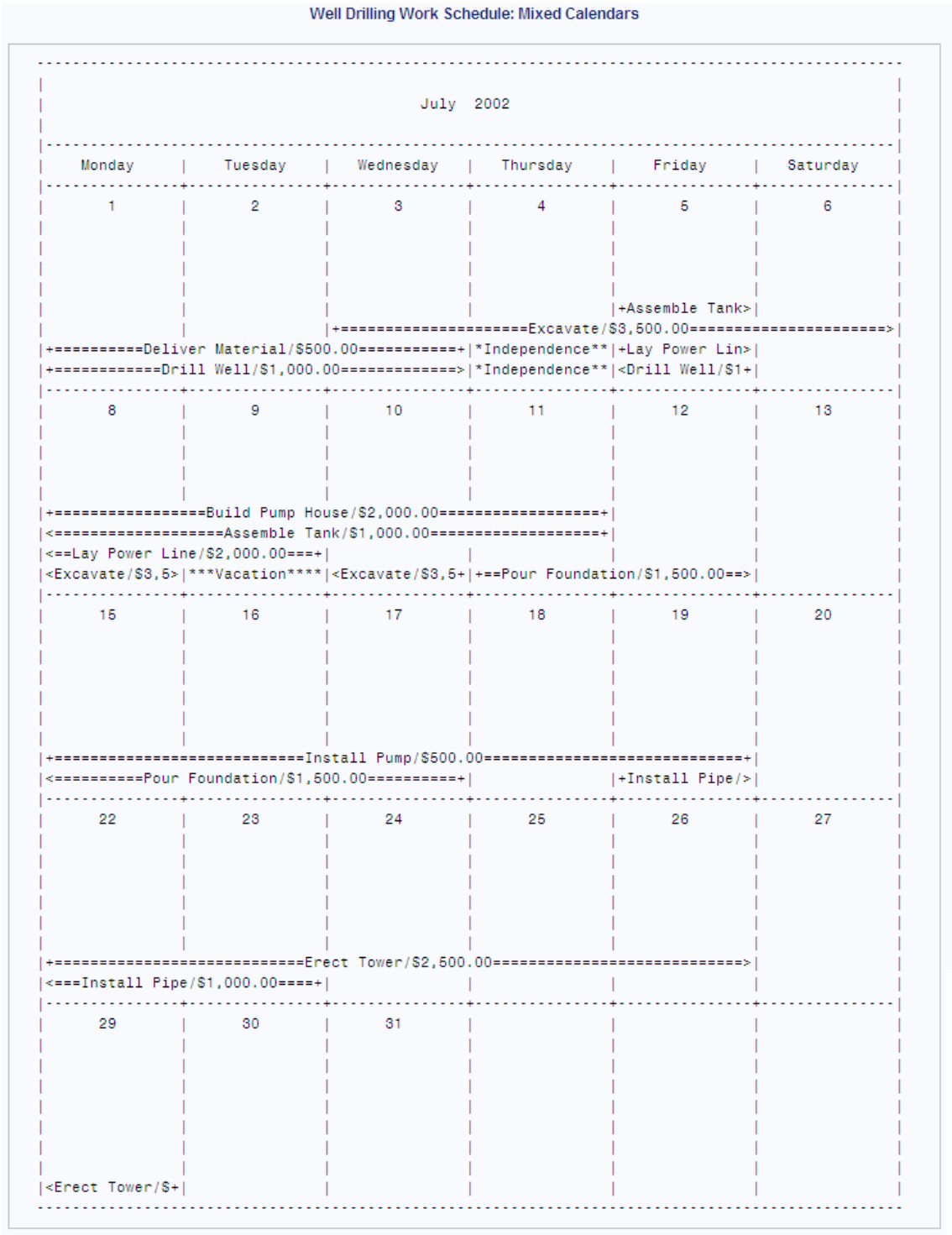
Program for Mixed Calendars

To produce mixed output instead of combined, use the same program and change the setting of the OUTPUT= option to OUTPUT=MIX:

```
proc calendar data=well.act
              holidata=well.hol
```

```
               caledata=well.cal
               workdata=well.wor
               datetime;
   calid _cal_ / output=mix;
   start date;
   dur dur;
   holistart date;
   holivar holiday;
   outstart Monday;
   outfin Saturday;
   title1 'Well Drilling Work Schedule: Mixed Calendars';
   format cost dollar9.2;
run;
```

Output for Mixed Calendars



Example 5: Schedule Calendar, Blank or with Holidays

Features: PROC CALENDAR statement options
FILL
HOLIDATA=

INTERVAL=WORKDAY

Other statements
- DUR statement
- HOLIDUR statement
- HOLISTART statement
- HOLIVAR statement

Details

This example produces a schedule calendar that displays only holidays. You can use this same code to produce a set of blank calendars by removing the HOLIDATA= option and the HOLISTART, HOLIVAR, and HOLIDUR statements from the PROC CALENDAR step.

Program

```
data acts;
   input sta : date7. act $ 11-30 dur;
   datalines;
01JAN03   Start                0
31DEC03   Finish               0
;

data holidays;
   input sta : date7. act $ 11-30 dur;
   datalines;
01JAN03   New Year's           1
30MAR03   Good Friday          1
28MAY03   Memorial Day         1
04JUL03   Independence Day     1
03SEP03   Labor Day            1
22NOV03   Thanksgiving         2
25DEC03   Christmas Break      5
;

options formchar="|----|+|---+=|-/\<>*";

proc calendar data=acts holidata=holidays fill interval=workday;

   start sta;
   dur dur;

   holistart sta;
   holivar act;
   holidur dur;

   title1 'Calendar of Holidays Only';
run;
```

Program Description

Create the activities data set. Specify one activity in the first month and one in the last, each with a duration of 0. PROC CALENDAR does not print activities with zero durations in the output.

```
data acts;
   input sta : date7. act $ 11-30 dur;
```

```
   datalines;
01JAN03   Start                0
31DEC03   Finish               0
;
```

Create the holidays data set.

```
data holidays;
   input sta : date7. act $ 11-30 dur;
   datalines;
01JAN03   New Year's           1
30MAR03   Good Friday          1
28MAY03   Memorial Day         1
04JUL03   Independence Day     1
03SEP03   Labor Day            1
22NOV03   Thanksgiving         2
25DEC03   Christmas Break      5
;
```

Set the FORMCHAR option. Setting FORMCHAR to this exact string renders better HTML output when it is viewed outside of the SAS environment where SAS Monospace fonts are not available.

```
options formchar="|----|+|---+=|-/\<>*";
```

Create the calendar. DATA= identifies the activities data set; HOLIDATA= identifies the holidays data set. FILL displays all months, even those with no activities. By default, only months with activities appear in the report. INTERVAL=WORKDAY specifies that activities and holidays are measured in 8-hour days and that PROC CALENDAR schedules activities only Monday through Friday.

```
proc calendar data=acts holidata=holidays fill interval=workday;
```

Specify an activity start date variable and an activity duration variable. The START statement specifies the variable in the activities data set that contains the starting date of the activities; DUR specifies the variable that contains the duration of each activity. Creating a schedule calendar requires START and DUR.

```
   start sta;
   dur dur;
```

Retrieve holiday information. The HOLISTART, HOLIVAR, and HOLIDUR statements specify the variables in the holidays data set that contain the start date, name, and duration of each holiday, respectively. When you use a holidays data set, HOLISTART is required. Because at least one holiday lasts more than one day, HOLIDUR (or HOLIFIN) is required.

```
   holistart sta;
   holivar act;
   holidur dur;
```

Specify the title.

```
   title1 'Calendar of Holidays Only';
run;
```

Output: HTML

The following output shows the December portion of the output. Without the INTERVAL=WORKDAY option, the 5-day Christmas break would be scheduled through the weekend.

Calendar of Holidays Only

December 2003

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25 Christmas Br	26 Christmas Br	27
28	29 Christmas Br	30 Christmas Br	31 Christmas Br			

Example 6: Calculating a Schedule Based on Completion of Predecessor Tasks

Features: PROC CALENDAR procedure features
- PROC CALENDAR statement
- CALID statement
- FIN statement
- VAR statement

Other features: PROC CPM step

PROC SORT step

Details

Program Description

This example does the following:

- calculates a project schedule containing multiple calendars (PROC CPM)
- produces a listing of the PROC CPM output data set (PROC PRINT)
- displays the schedule in calendar format (PROC CALENDAR)

This example features PROC CPM's ability to calculate a schedule that meets the following criteria:

- is based on an initial starting date
- applies different non-work periods to different calendars, such as personal vacation days to each employee's schedule
- includes milestones (activities with a duration of 0)

Automating Your Scheduling Task with SAS/OR Software

When changes occur to a schedule, you have to adjust the activity starting dates manually if you use PROC CALENDAR to produce a schedule calendar. Alternatively, you can use PROC CPM in SAS/OR software to reschedule work when dates change. Even more important, you can provide only an initial starting date for a project and let PROC CPM calculate starting dates for activities, based on identified successor tasks, that is, tasks that cannot begin until their predecessors end.

In order to use PROC CPM, you must complete the following steps:

1. Create an activities data set that contains activities with durations. (You can indicate nonwork days, weekly work schedules, and work shifts with holidays, calendar, and work-shift data sets.)
2. Indicate which activities are successors to others (precedence relationships).
3. Define resource limitations if you want them considered in the schedule.
4. Provide an initial starting date.

PROC CPM can process your data to generate a data set that contains the start and end dates for each activity. PROC CPM schedules the activities, based on the duration information, weekly work patterns, work shifts, as well as holidays and nonwork days that interrupt the schedule. You can generate several views of the schedule that is computed by PROC CPM, from a simple listing of start and finish dates to a calendar, a Gantt chart, or a network diagram.

See Also

This example introduces users of PROC CALENDAR to more advanced SAS scheduling tools. For an introduction to project management tasks and tools and several examples, see *Project Management Using the SAS System*. For more examples, see *SAS/OR Software: Project Management Examples*. For complete reference documentation, see *SAS/OR(R) 9.3 User's Guide: Mathematical Programming*.

Program

```
options formchar="|----|+|---+=|-/\<>*";

data grant;
   input jobnum Task $ 4-22 Days Succ1 $ 27-45 aldate : date7. altype $
         _cal_ $;
   format aldate date7.;
   datalines;
1  Run Exp 1          11  Analyze Exp 1       .       .   Student
2  Analyze Exp 1       5  Send Report 1       .       .   Prof.
3  Send Report 1       0  Run Exp 2           .       .   Prof.
4  Run Exp 2          11  Analyze Exp 2       .       .   Student
5  Analyze Exp 2       4  Send Report 2       .       .   Prof.
6  Send Report 2       0  Write Final Report  .       .   Prof.
7  Write Final Report  4  Send Final Report   .       .   Prof.
8  Send Final Report   0                      .       .   Student
9  Site Visit          1                    18jul07 ms  Prof.
;

data nowork;
   format holista date7. holifin date7.;
   input holista : date7. holifin : date7. name $ 17-32 _cal_ $;
   datalines;
04jul07 04jul07 Independence Day Prof.
03sep07 03sep07 Labor Day        Prof.
04jul07 04jul07 Independence Day Student
03sep07 03sep07 Labor Day        Student
16jul07 17jul07 PROF Vacation    Prof.
16aug07 17aug07 STUDENT Vacation Student
;

proc cpm data=grant
         date='01jul07'd
         interval=weekday
         out=gcpm1
         holidata=nowork;
   activity task;
   successor succ1;
   duration days;
   calid _cal_;
   id task;
   alignddate aldate;
   aligntype altype;
   holiday holista / holifin=holifin;
run;

proc print data=gcpm1;
   title 'Data Set GCPM1, Created with PROC CPM';
run;

proc sort data=gcpm1;
   by e_start;
run;

proc calendar data=gcpm1
              holidata=nowork
              interval=workday;
   start e_start;
```

```
   fin   e_finish;
   calid _cal_ / output=combine;
   holistart holista;
   holifin   holifin;
   holivar name;
   var task;
   title 'Schedule for Experiment X-15';
   title2 'Professor and Student Schedule';
run;
```

Program Description

Set the FORMCHAR option. Setting FORMCHAR to this exact string renders better HTML output when it is viewed outside of the SAS environment where SAS Monospace fonts are not available.

```
options formchar="|----|+|---+=|-/\<>*";
```

Create the activities data set and identify separate calendars. This data identifies two calendars: the professor's (the value of _CAL_ is *Prof.*) and the student's (the value of _CAL_ is *Student*). The Succ1 variable identifies which activity cannot begin until the current one ends. For example *Analyze Exp 1* cannot begin until *Run Exp 1* is completed. The DAYS value of *0* for JOBNUM *3, 6,* and *8* indicates that these jobs are milestones.

```
data grant;
   input jobnum Task $ 4-22 Days Succ1 $ 27-45 aldate : date7. altype $
         _cal_ $;
   format aldate date7.;
   datalines;
1  Run Exp 1          11  Analyze Exp 1       .       .   Student
2  Analyze Exp 1       5  Send Report 1       .       .   Prof.
3  Send Report 1       0  Run Exp 2           .       .   Prof.
4  Run Exp 2          11  Analyze Exp 2       .       .   Student
5  Analyze Exp 2       4  Send Report 2       .       .   Prof.
6  Send Report 2       0  Write Final Report  .       .   Prof.
7  Write Final Report  4  Send Final Report   .       .   Prof.
8  Send Final Report   0                      .       .   Student
9  Site Visit          1                    18jul07 ms  Prof.
;
```

Create the holidays data set and identify which calendar a nonwork day belongs to. The two holidays are listed twice, once for the professor's calendar and once for the student's. Because each person is associated with a separate calendar, PROC CPM can apply the personal vacation days to the appropriate calendars.

```
data nowork;
   format holista date7. holifin date7.;
   input holista : date7. holifin : date7. name $ 17-32 _cal_ $;
   datalines;
04jul07 04jul07 Independence Day Prof.
03sep07 03sep07 Labor Day        Prof.
04jul07 04jul07 Independence Day Student
03sep07 03sep07 Labor Day        Student
16jul07 17jul07 PROF Vacation    Prof.
16aug07 17aug07 STUDENT Vacation Student
;
```

Calculate the schedule with PROC CPM. PROC CPM uses information supplied in the activities and holidays data sets to calculate start and finish dates for each activity. The DATE= option supplies the starting date of the project. The CALID statement is not required, even though this example includes two calendars, because the calendar identification variable has the special name _CAL_.

```
proc cpm data=grant
         date='01jul07'd
         interval=weekday
         out=gcpm1
         holidata=nowork;
   activity task;
   successor succ1;
   duration days;
   calid _cal_;
   id task;
   aligndate aldate;
   aligntype altype;
   holiday holista / holifin=holifin;
run;
```

Print the output data set that was created with PROC CPM. This step is not required. PROC PRINT is a useful way to view the calculations produced by PROC CPM.

```
proc print data=gcpm1;
   title 'Data Set GCPM1, Created with PROC CPM';
run;
```

Sort GCPM1 by the variable that contains the activity start dates before using it with PROC CALENDAR.

```
proc sort data=gcpm1;
   by e_start;
run;
```

Create the schedule calendar. GCPM1 is the activity data set. PROC CALENDAR uses the S_START and S_FINISH dates, calculated by PROC CPM, to print the schedule. The VAR statement selects only the variable TASK to display on the calendar output.

```
proc calendar data=gcpm1
              holidata=nowork
              interval=workday;
   start e_start;
   fin   e_finish;
   calid _cal_ / output=combine;
   holistart holista;
   holifin   holifin;
   holivar name;
   var task;
   title 'Schedule for Experiment X-15';
   title2 'Professor and Student Schedule';
run;
```

Output: HTML

PROC PRINT displays the observations in GCPM1, showing the scheduling calculations created by PROC CPM.

Data Set GCPM1, Created with PROC CPM

Obs	Task	Succ1	Days	_cal_	E_START	E_FINISH	L_START	L_FINISH	T_FLOAT	F_FLOAT
1	Run Exp 1	Analyze Exp 1	11	Student	02JUL07	17JUL07	02JUL07	17JUL07	0	0
2	Analyze Exp 1	Send Report 1	5	Prof.	18JUL07	24JUL07	18JUL07	24JUL07	0	0
3	Send Report 1	Run Exp 2	0	Prof.	25JUL07	25JUL07	25JUL07	25JUL07	0	0
4	Run Exp 2	Analyze Exp 2	11	Student	25JUL07	08AUG07	25JUL07	08AUG07	0	0
5	Analyze Exp 2	Send Report 2	4	Prof.	09AUG07	14AUG07	09AUG07	14AUG07	0	0
6	Send Report 2	Write Final Report	0	Prof.	15AUG07	15AUG07	15AUG07	15AUG07	0	0
7	Write Final Report	Send Final Report	4	Prof.	15AUG07	20AUG07	15AUG07	20AUG07	0	0
8	Send Final Report		0	Student	21AUG07	21AUG07	21AUG07	21AUG07	0	0
9	Site Visit		1	Prof.	19JUL07	19JUL07	19JUL07	19JUL07	0	0

PROC CALENDAR created the following schedule calendar by using the S_START and S_FINISH dates that were calculated by PROC CPM. The activities on July 25 and August 15, because they are milestones, do not delay the start of a successor activity. Note that Site Visit occurs on July 18, the same day that Analyze Exp 1 occurs. To

prevent this overallocation of resources, you can use *resource constrained scheduling*, available in SAS/OR software.

Schedule for Experiment X-15
Professor and Student Schedule

July 2007

	Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
	1	2	3	4	5	6	7
PROF.				Independence			
STUDENT		+=======Run Exp 1=======>		Independence	<=======Run Exp 1=======>		
	8	9	10	11	12	13	14
STUDENT		<============================Run Exp 1============================>					
	15	16	17	18	19	20	21
PROF.		PROF Vacatio	PROF Vacatio		+Site Visit+		
				+===========Analyze Exp 1============>			
STUDENT		<=======Run Exp 1=======+					
	22	23	24	25	26	27	28
PROF.		<=====Analyze Exp 1=====+		+Send Repor+			
STUDENT				+==============Run Exp 2==============>			
	29	30	31				
STUDENT		<=======Run Exp 2=======>					

Schedule for Experiment X-15
Professor and Student Schedule

August 2007

	Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
				1	2	3	4
STUDENT				<==========Run Exp 2==========			
	5	6	7	8	9	10	11
PROF.					+=====Analyze Exp 2=====>		
STUDENT		<==========Run Exp 2==========+					
	12	13	14	15	16	17	18
PROF.				+=========Write Final Report=========>			
		<=====Analyze Exp 2=====+		+Send Repor+			
STUDENT					STUDENT Vaca	STUDENT Vaca	
	19	20	21	22	23	24	25
PROF.		<Write Fina+					
STUDENT			+Send Final+				
	26	27	28	29	30	31	

Example 7: Summary Calendar with MEAN Values by Observation

Features: CALID statement
CAL variable
OUTPUT=SEPARATE option

Other statements
FORMAT statement
LABEL statement
MEAN statement
SUM statement

Other features: PROC FORMAT
PICTURE statement

Details

This example does the following:

- produces a summary calendar
- displays holidays
- produces sum and mean values by business day (observation) for three variables
- prints a legend and uses variable labels
- uses picture formats to display values

To produce MEAN values based on *the number of days in the calendar month*, use MEANTYPE=NDAYS. By default, MEANTYPE=NOBS, which calculates the MEAN values according to *the number of days for which data exists*.

Program

```
data meals;
   input date : date7. Brkfst Lunch Dinner;
   datalines;
01Dec08       123 234 238
02Dec08       188 188 198
03Dec08       123 183 176
04Dec08       200 267 243
05Dec08       176 165 177
08Dec08       178 198 187
09Dec08       165 176 187
10Dec08       187 176 231
11Dec08       176 187 222
12Dec08       187 187 123
15Dec08       176 165 177
16Dec08       156   . 167
17Dec08       198 143 167
18Dec08       178 198 187
19Dec08       165 176 187
22Dec08       187 187 123
;
```

```
data closed;
   input date date. holiday $ 11-25;
   datalines;
26DEC08   Repairs
29DEC08   Repairs
30DEC08   Repairs
31DEC08   Repairs
23DEC08   Vacation
24DEC08   Christmas Eve
25DEC08   Christmas
;

proc sort data=meals;
   by date;
run;

proc format;
   picture bfmt other = '000 Brkfst';
   picture lfmt other = '000 Lunch ';
   picture dfmt other = '000 Dinner';
run;

options formchar="|----|+|---+=|-/\<>*";

proc calendar data=meals holidata=closed;
   start date;

   holistart date;
   holiname holiday;

   sum brkfst lunch dinner / format=4.0;
   mean brkfst lunch dinner / format=6.2;
   label brkfst = 'Breakfasts Served'
         lunch  = '   Lunches Served'
         dinner = '   Dinners Served';
   format brkfst bfmt.
          lunch lfmt.
          dinner dfmt.;

   title 'Meals Served in Company Cafeteria';
   title2 'Mean Number by Business Day';
run;

title;
```

Program Description

Create the activities data set. MEALS records how many meals were served for breakfast, lunch, and dinner on the days that the cafeteria was open for business.

```
data meals;
   input date : date7. Brkfst Lunch Dinner;
   datalines;
01Dec08       123 234 238
02Dec08       188 188 198
03Dec08       123 183 176
04Dec08       200 267 243
05Dec08       176 165 177
08Dec08       178 198 187
```

```
09Dec08       165 176 187
10Dec08       187 176 231
11Dec08       176 187 222
12Dec08       187 187 123
15Dec08       176 165 177
16Dec08       156   . 167
17Dec08       198 143 167
18Dec08       178 198 187
19Dec08       165 176 187
22Dec08       187 187 123
;
```

Create the holidays data set.

```
data closed;
   input date date. holiday $ 11-25;
   datalines;
26DEC08   Repairs
29DEC08   Repairs
30DEC08   Repairs
31DEC08   Repairs
23DEC08   Vacation
24DEC08   Christmas Eve
25DEC08   Christmas
;
```

Sort the activities data set by the activity starting date. You are not required to sort the holidays data set.

```
proc sort data=meals;
   by date;
run;
```

Create picture formats for the variables that indicate how many meals were served.

```
proc format;
   picture bfmt other = '000 Brkfst';
   picture lfmt other = '000 Lunch ';
   picture dfmt other = '000 Dinner';
run;
```

Set the FORMCHAR and LINESIZE options. Setting FORMCHAR to this exact string renders better HTML output when it is viewed outside of the SAS environment where SAS Monospace fonts are not available.

```
options formchar="|----|+|---+=|-/\<>*";
```

Create the summary calendar. DATA= identifies the activities data set; HOLIDATA= identifies the holidays data set. The START statement specifies the variable in the activities data set that contains the activity starting date; START is required.

```
proc calendar data=meals holidata=closed;
   start date;
```

Retrieve holiday information. The HOLISTART and HOLIVAR statements specify the variables in the holidays data set that contain the start date and the name of each holiday, respectively. HOLISTART is required when you use a holidays data set.

```
holistart date;
holiname holiday;
```

Calculate, label, and format the sum and mean values. The SUM and MEAN statements calculate sum and mean values for three variables and print them with the specified format. The LABEL statement prints a legend and uses labels instead of variable names. The FORMAT statement associates picture formats with three variables.

```
sum brkfst lunch dinner / format=4.0;
mean brkfst lunch dinner / format=6.2;
label brkfst = 'Breakfasts Served'
      lunch  = '   Lunches Served'
      dinner = '   Dinners Served';
format brkfst bfmt.
       lunch lfmt.
       dinner dfmt.;
```

Specify the titles.

```
   title 'Meals Served in Company Cafeteria';
   title2 'Mean Number by Business Day';
run;

title;
```

Output: HTML

Meals Served in Company Cafeteria
Mean Number by Business Day

December 2008

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
	1 123 Brkfst 234 Lunch 238 Dinner	2 188 Brkfst 188 Lunch 198 Dinner	3 123 Brkfst 183 Lunch 176 Dinner	4 200 Brkfst 267 Lunch 243 Dinner	5 176 Brkfst 165 Lunch 177 Dinner	6
7	8 178 Brkfst 198 Lunch 187 Dinner	9 165 Brkfst 176 Lunch 187 Dinner	10 187 Brkfst 176 Lunch 231 Dinner	11 176 Brkfst 187 Lunch 222 Dinner	12 187 Brkfst 187 Lunch 123 Dinner	13
14	15 176 Brkfst 165 Lunch 177 Dinner	16 156 Brkfst . 167 Dinner	17 198 Brkfst 143 Lunch 167 Dinner	18 178 Brkfst 198 Lunch 187 Dinner	19 165 Brkfst 176 Lunch 187 Dinner	20
21	22 187 Brkfst 187 Lunch 123 Dinner	23 **Vacation**	24 Christmas Ev	25 *Christmas**	26 **Repairs***	27
28	29 **Repairs***	30 **Repairs***	31 **Repairs***			

	Sum	Mean
Breakfasts Served	2763	172.69
Lunches Served	2830	188.67
Dinners Served	2990	186.88

Example 8: Multiple Summary Calendars with Atypical Work Shifts (Separated Output)

Features: PROC CALENDAR statement options

DATETIME
LEGEND

CALID statement
CAL variable
OUTPUT=SEPARATE option

Other statements
OUTSTART statement
OUTFIN statement
SUM statement

Data sets: WELL.ACT
WELL.HOL

Details

This example does the following:

- produces a summary calendar for multiple calendars in a single PROC step
- prints the calendars on separate pages
- displays holidays
- uses separate work patterns, work shifts, and holidays for each calendar

Producing Different Output for Multiple Calendars

This example produces separate output for multiple calendars. To produce combined or mixed output for this data, you need to change only the following two things:

- how the activities data set is sorted
- how the OUTPUT= option is set

Table 8.14 *Sort and OUTPUT= Settings*

Print Options	Sorting Variables	OUTPUT= Settings	Examples
Separate pages for each calendar	Calendar ID and starting date	SEPARATE	3, 8
All activities on the same page and identify each calendar	Starting date	COMBINE	4, 2
All activities on the same page and NOT identify each calendar	Starting date	MIX	4

Program

```
libname well
'SAS-library';
run;
```

```
proc sort data=well.act;
   by _cal_ date;
run;

options formchar="|----|+|---+=|-/\<>*" linesize=132;

proc calendar data=well.act
              holidata=well.hol
              datetime legend;

   calid _cal_ / output=separate;

   start date;
   holistart date;
   holivar holiday;

   sum cost / format=dollar10.2;

   outstart Monday;
   outfin Saturday;

   title 'Well Drilling Cost Summary';
   title2 'Separate Calendars';
   format cost dollar10.2;
run;
```

Program Description

Specify the SAS library where the activities data set is stored.

```
libname well
'SAS-library';
run;
```

Sort the activities data set by the variables containing the calendar identification and the starting date, respectively.

```
proc sort data=well.act;
   by _cal_ date;
run;
```

Set the FORMCHAR option. Setting FORMCHAR to this exact string renders better HTML output when it is viewed outside of the SAS environment where SAS Monospace fonts are not available. LINESIZE needs to be set in this example to prevent truncating data in the output.

```
options formchar="|----|+|---+=|-/\<>*" linesize=132;
```

Create the summary calendar. DATA= identifies the activities data set; HOLIDATA= identifies the holidays data set; CALDATA= identifies the calendar data set; WORKDATA= identifies the workdays data set. DATETIME specifies that the variable specified with the START statement contains a SAS datetime value. LEGEND prints text that identifies the variables.

```
proc calendar data=well.act
              holidata=well.hol
              datetime legend;
```

Print each calendar on a separate page. The CALID statement specifies that the _CAL_ variable identifies calendars. OUTPUT=SEPARATE prints information for each calendar on separate pages.

```
calid _cal_ / output=separate;
```

Specify an activity start date variable and retrieve holiday information. The START statement specifies the variable in the activities data set that contains the activity starting date. The HOLISTART and HOLIVAR statements specify the variables in the holidays data set that contain the start date and name of each holiday, respectively. These statements are required when you use a holidays data set.

```
start date;
holistart date;
holivar holiday;
```

Calculate sum values. The SUM statement totals the COST variable for all observations in each calendar.

```
sum cost / format=dollar10.2;
```

Display a 6-day week. OUTSTART and OUTFIN specify that the calendar display a 6-day week, Monday through Saturday.

```
outstart Monday;
outfin Saturday;
```

Specify the titles and format the Cost variable.

```
title 'Well Drilling Cost Summary';
title2 'Separate Calendars';
format cost dollar10.2;
run;
```

Output: HTML

Well Drilling Cost Summary
Separate Calendars

cal=CAL1

July 2002

Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
1 Drill Well 3.5 $1,000.00	2	3	4 ***Independence*** Lay Power Line 3 $2,000.00	5 Assemble Tank 4 $1,000.00	6
8 Build Pump House 3 $2,000.00	9	10	11 Pour Foundation 4 $1,500.00	12	13
15 Install Pump 4 $500.00	16	17	18	19 Install Pipe 2 $1,000.00	20 Erect Tower 6 $2,500.00
22	23	24	25	26	27
29	30	31			

Legend	Sum
task	
dur	
cost	$11,500.00

Well Drilling Cost Summary
Separate Calendars

cal=CAL2

July 2002

Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
1 Deliver Material 2 $500.00	2	3 Excavate 4.75 $3,500.00	4	5	6
8	9 *****Vacation*****	10	11	12	13
15	16	17	18	19	20
22	23	24	25	26	27
29	30	31			

Legend	Sum
task	
dur	
cost	$4,000.00

Chapter 9

CATALOG Procedure

Overview: CATALOG Procedure

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

For more information about SAS libraries and catalogs, refer to *SAS Language Reference: Concepts*.

To learn how to use the SAS windowing environment to manage entries in a SAS catalog, see the SAS online Help for the SAS Explorer window. You might prefer to use the Explorer window instead of using PROC CATALOG. The window can do most of what the procedure does.

Concepts

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. A simple RUN statement does not, however, 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 SAS 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, however, the remaining statements *do not execute* before the procedure ends.

See "Example 2: Displaying Contents, Changing Names, and Changing a Description " on page 198.

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, 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;
```

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, however, because it is in a different RUN group.

CAUTION:

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, PROC CATALOG generates an error. You can supply an entry type in one of four ways, as shown in the following table:

Table 9.1 *Supplying an Entry Type*

Entry Type	Example

Entry name	`delete`
	`test1.program`
	`test1.log test2.log;`

ET= in parentheses	`delete` `test1 (et=program);`
ET= after a slash *	`delete test1 (et=program)` `test1 test2 / et=log;`
ENTRYTYPE= without a slash **	`proc catalog catalog=mycat et=log;` `delete test1 test2;`

* in a subordinate statement

** in the PROC CATALOG or the COPY statement

Note: All statements, except the CONTENTS statement, accept the ENTRYTYPE= (alias ET=) option.

Why Use the ENTRYTYPE= Option?

ENTRYTYPE= 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 ENTRYTYPE= in the PROC CATALOG statement. To set the default for only the current statement, use ENTRYTYPE= in a subordinate statement.

You can have many entries of one type and a few of other types. You can use ENTRYTYPE= 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= specifications contradict each other:

```
/* THIS IS INCORRECT CODE. */
proc catalog cat=sample et=help;
   delete a b c / et=program;
run;
```

The ENTRYTYPE= Option

The ENTRYTYPE= option is available in every statement in the CATALOG procedure except CONTENTS.

ENTRYTYPE=*etype*

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. The alias is ET=. If you omit ENTRYTYPE=, PROC CATALOG processes all entries in the catalog.

Alias: ET=

Default: If you omit ENTRYTYPE=, PROC CATALOG processes all entries in the catalog.

Interactions:

If you specify ENTRYTYPE= in the PROC CATALOG statement, do not specify either ENTRYTYPE= or (ENTRYTYPE=) in a subordinate statement.

(ENTRYTYPE=etype) in parentheses immediately following an entry name overrides ENTRYTYPE= in that same statement.

Tips:

On all statements except the PROC CATALOG and COPY statements, this option follows a slash.

To process multiple entry types in a single PROC CATALOG step, use ENTRYTYPE= in a subordinate statement, not in the PROC CATALOG statement.

See: "Specifying an Entry Type" on page 181 and "Example 1: Copying, Deleting, and Moving Catalog Entries from Multiple Catalogs" on page 194

(ENTRYTYPE=*etype*)

in parentheses, identifies the type of the entry just preceding it.

Alias: (ET=)

Restriction: (ENTRYTYPE=*etype*) 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: (ENTRYTYPE=*etype*) immediately following an entry name overrides ENTRYTYPE= in that same statement.

Tips:

This form is useful mainly for specifying exceptions to an ENTRYTYPE= option 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 (ENTRYTYPE=) in parentheses only once for each pair of names following the second name in the pair as shown in the following example:

```
change old1=new1 (et=log)
       old1=new2 (et=help);
```

See: "Specifying an Entry Type" on page 181 , "Example 1: Copying, Deleting, and Moving Catalog Entries from Multiple Catalogs" on page 194 and "Example 2: Displaying Contents, Changing Names, and Changing a Description " on page 198

Catalog Concatenation

About Catalog Concatenation

There are two types of CATALOG concatenation. The first is specified by the LIBNAME statement and the second 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:

1. If the input catalog is a concatenation and if the output catalog exists in any level of the input concatenation, the copy is not allowed.
2. If the output catalog is a concatenation and if the input catalog exists in the first level of the output concatenation, the copy is not allowed.

For example, the following code demonstrates these two rules, and the copy fails:

```
LIBNAME first 'path-name1';
LIBNAME second 'path-name2';
/* 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.

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

Syntax: CATALOG Procedure

Tips: Supports RUN-group processing.

You can perform similar functions with the SAS Explorer window and with dictionary tables in the SQL procedure. For information about the Explorer window, see the online Help. For information about PROC SQL, see *SAS SQL Procedure User's Guide*.

See: CATALOG Procedure under Windows, UNIX, z/OS

PROC CATALOG CATALOG=<*libref.*>*catalog* <ENTRYTYPE=*etype*> <FORCE> <KILL>;
CONTENTS <OUT=*SAS-data-set*> <FILE=*fileref*>;
COPY OUT=<*libref.*>*catalog* <*options*>;
SELECT *entry-1* <*...entry-n*> </ ENTRYTYPE=*etype*>;
EXCLUDE *entry-1* <*...entry-n*> </ ENTRYTYPE=*etype*>;
CHANGE *old-name-1=new-name-1*
<*…old-name-n=new-name-n*>
</ ENTRYTYPE=*etype*>;
EXCHANGE *name-1=other-name-1*
<*…name-n=other-name-n*>
</ ENTRYTYPE=*etype*>;
DELETE *entry-1* <*...entry-n*> </ ENTRYTYPE=*etype*>;
MODIFY *entry* (DESCRIPTION=<<'>*entry-description*<'>>)</ ENTRYTYPE=*etype*>;
SAVE *entry-1* <*...entry-n*> </ ENTRYTYPE=*etype*>;

Statement	Task	Example
"PROC CATALOG Statement"	Copy entries from one SAS catalog to another	Ex. 1, Ex. 2, Ex. 3
"CHANGE Statement"	Change the names of catalog entries	Ex. 2

Statement	Task	Example
"CONTENTS Statement"	Print the contents of a catalog	Ex. 2
"COPY Statement"	Copy some or all of the entries in one catalog to another catalog	Ex. 1
"DELETE Statement"	Delete specified entries	Ex. 1
"EXCHANGE Statement"	Switch the names of two catalog entries	
"EXCLUDE Statement"	Exclude entries from being copied	Ex. 1
"MODIFY Statement"	Change the description of a catalog entry	Ex. 2
"SAVE Statement"	Delete all except the entries specified	
"SELECT Statement"	Copy only selected entries	Ex. 1

PROC CATALOG Statement

Copies entries from one SAS catalog to another.

Syntax

PROC CATALOG CATALOG=<*libref.*>*catalog* <ENTRYTYPE=*etype*> <FORCE> <KILL>;

Required Argument

CATALOG=<*libref.*>*catalog*
specifies the SAS catalog to process.

Alias: CAT=, C=

Default: If ENTRYTYPE= is not specified, PROC CATALOG processes all entries in the catalog.

Example: "Example 3: Using the FORCE Option with the KILL Option" on page 200

Optional Arguments

ENTRYTYPE=*etype*
restricts processing of the current PROC CATALOG step to one entry type.

Alias: ET=

Default: If you omit ENTRYTYPE=, PROC CATALOG processes all entries in a catalog.

Interactions:

The specified entry type applies to any one-level entry names used in a subordinate statement. You cannot override this specification in a subordinate statement.

ENTRYTYPE= does not restrict the effects of the KILL option.

Tip: In order to process multiple entry types in a single PROC CATALOG step, use ENTRYTYPE= in a subordinate statement, not in the PROC CATALOG statement.

See: "Specifying an Entry Type" on page 181

Examples:

"Example 1: Copying, Deleting, and Moving Catalog Entries from Multiple Catalogs" on page 194

"Example 2: Displaying Contents, Changing Names, and Changing a Description " on page 198

FORCE

forces statements to execute on a catalog that is opened by another resource environment.

Some CATALOG statements 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 the catalogs that are affected by FORCE:

KILL
affects the specified catalog.

COPY
affects the OUT= catalog.

COPY MOVE
affects the IN= and the OUT= catalogs.

SAVE
affects the specified catalog.

Tip: Use FORCE to execute the statement, even if exclusive access cannot be obtained.

Example: "Example 3: Using the FORCE Option with the KILL Option" on page 200

KILL

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 ENTRYTYPE= 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: KILL 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 200

CHANGE Statement

Renames one or more catalog entries.

Tip: You can change multiple names in a single CHANGE statement or use multiple CHANGE statements.

Example: "Example 2: Displaying Contents, Changing Names, and Changing a Description " on page 198

Syntax

CHANGE *old-name-1=new-name-1*
<*…old-name-n=new-name-n*>
</ ENTRYTYPE=*etype*>;

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 (*ename.etype*) or with the ENTRYTYPE= option.

Optional Argument

ENTRYTYPE=*etype*
restricts processing to one entry type.

See:
"The ENTRYTYPE= Option" on page 182
"Specifying an Entry Type" on page 181

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: ENTRYTYPE= (ET=) option is not available for the CONTENTS statement.

Example: "Example 2: Displaying Contents, Changing Names, and Changing a Description " on page 198

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.

Alias: CAT=, C=

Default: None

FILE=*fileref*
sends the contents to an external file, 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 on the SAS log reports that a data set has been created. The data set contains six variables in the following order:

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 SELECT or EXCLUDE statements, 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 this statement.

Example: "Example 1: Copying, Deleting, and Moving Catalog Entries from Multiple Catalogs" on page 194

Syntax

COPY OUT=<*libref*.>*catalog* <*options*>;

Summary of Optional Arguments

ENTRYTYPE=*etype*
: restricts processing to one type of entry.

IN=<*libref*.>*catalog*
: copies from a different catalog in the same step.

LOCKCAT=EXCLUSIVE | SHARE
: enables concurrent users to copy to the same catalog at the same time.

MOVE
: moves (copy and then delete) a catalog entry.

NEW
: copies entries to a new catalog (overwriting the catalog if it already exists).

NOEDIT
: protects several types of SAS/AF entries from being edited with PROC BUILD.

NOSOURCE
: does not copy source lines from a PROGRAM, FRAME, or SCL entry.

Required Argument

OUT=<*libref*.>*catalog*
names the catalog to which entries are copied.

Optional Arguments

ENTRYTYPE=*etype*
restricts processing to one entry type for the current COPY statement and any subsequent SELECT or EXCLUDE statements.

See:
"The ENTRYTYPE= Option" on page 182
"Specifying an Entry Type" on page 181

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 194

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 MOVE removes all entries from a catalog, the procedure deletes the catalog from the library.

NEW
overwrites the destination (specified by OUT=) if it already exists. If you omit NEW, PROC CATALOG updates the destination.

See: For information about using the NEW option with concatenated catalogs, see "Catalog Concatenation" on page 183.

NOEDIT
prevents the copied version of the following SAS/AF entry types from being edited by the BUILD procedure:

CBT	PROGRAM
FRAME	SCL
HELP	SYSTEM
MENU	

Restriction: If you specify the NOEDIT option for an entry that is not one of these types, it is ignored.

Tip: When creating SAS/AF applications for other users, use NOEDIT 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 194

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, it is ignored.

DELETE Statement

Deletes entries from a SAS catalog.

Tips: Use DELETE to delete only a few entries; use SAVE 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 193

Example: "Example 1: Copying, Deleting, and Moving Catalog Entries from Multiple Catalogs" on page 194

Syntax

DELETE *entry-1 <...entry-n>* </ ENTRYTYPE=*etype*>;

Required Argument

entry-1 <...entry-n>
specifies the name of one or more SAS catalog entries.

Restriction: You must designate the type of the entry, either with the name (*ename.etype*) or with the ENTRYTYPE= option.

Optional Argument

ENTRYTYPE=*etype*
restricts processing to one entry type.

See:
"The ENTRYTYPE= Option" on page 182
"Specifying an Entry Type" on page 181

EXCHANGE Statement

Switches the name of two catalog entries.

Restriction: The catalog entries must be of the same type.

Syntax

EXCHANGE *name-1=other-name-1*
<…*name-n=other-name-n*>
</ ENTRYTYPE=*etype*>;

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 181

Optional Argument

ENTRYTYPE=*etype*
restricts processing to one entry type.

See:
"The ENTRYTYPE= Option" on page 182
"Specifying an Entry Type" on page 181

EXCLUDE Statement

Specifies entries that the COPY statement does not copy.

Restrictions: 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 188 and SELECT Statement on page 193

Example: "Example 1: Copying, Deleting, and Moving Catalog Entries from Multiple Catalogs" on page 194

Syntax

EXCLUDE *entry-1 <...entry-n>* </ ENTRYTYPE=*etype*>;

Required Argument

entry-1 <...entry-n>
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 (*ename.etype*) or with the ENTRYTYPE= option.

See: "Specifying an Entry Type" on page 181

Optional Argument

ENTRYTYPE=*etype*
restricts processing to one entry type.

See:
"The ENTRYTYPE= Option" on page 182
"Specifying an Entry Type" on page 181

MODIFY Statement

Changes the description of a catalog entry.

Example: "Example 2: Displaying Contents, Changing Names, and Changing a Description " on page 198

Syntax

MODIFY *entry* (DESCRIPTION=<<'>*entry-description*<'>>) </ ENTRYTYPE=*etype*>;

Required Arguments

entry
specifies the name of one SAS catalog entry. You can specify the entry type with the name.

Restriction: You must designate the type of the entry, either when you specify the name (*ename.etype*) or with the ENTRYTYPE= option.

See: "Specifying an Entry Type" on page 181

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 quotes.

Alias: DESC

Tip: Use DESCRIPTION= with no text to remove the current description.

Optional Argument

ENTRYTYPE=*etype*
restricts processing to one entry type.

See:
"The ENTRYTYPE= Option" on page 182
"Specifying an Entry Type" on page 181

SAVE Statement

Specifies entries not to delete from a SAS catalog.

Restriction: Cannot limit the effects of the KILL option.

Tips: Use SAVE to delete all but a few entries in a catalog. Use DELETE 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 190

Syntax

SAVE *entry-1 <...entry-n>* </ ENTRYTYPE=*etype*>;

Required Argument

entry <...entry-n>
specifies the name of one or more SAS catalog entries.

Restriction: You must designate the type of the entry, either with the name (*ename.etype*) or with the ENTRYTYPE= option.

Optional Argument

ENTRYTYPE=*etype*
restricts processing to one entry type.

See:
"The ENTRYTYPE= Option" on page 182
"Specifying an Entry Type" on page 181

SELECT Statement

Specifies entries that the COPY statement copies.

Restrictions: Requires the COPY 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 188 and EXCLUDE Statement on page 191

Example: "Example 1: Copying, Deleting, and Moving Catalog Entries from Multiple Catalogs" on page 194

Syntax

SELECT *entry-1 <...entry-n. </ ENTRYTYPE=etype>*;

Required Argument

entry-1 <...entry-n>
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 (*ename.etype*) or with the ENTRYTYPE= option.

Optional Argument

ENTRYTYPE=*etype*
restricts processing to one entry type.

See:
"The ENTRYTYPE= Option" on page 182.
"Specifying an Entry Type" on page 181.

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 *SAS Output Delivery System: User's Guide*.

Table 9.2 *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

CATALOG= argument

COPY statement options
- IN=
- MOVE
- NOEDIT

DELETE statement options
- ENTRYTYPE= or ET=

EXCLUDE statement options
- ENTRYTYPE= or ET=
- (ENTRYTYPE=) or (ET=)

SELECT statement options:
- ENTRYTYPE= or ET=

Other statements
- QUIT statement
- RUN statement

Details

This example demonstrates all the following actions:

- 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

Program

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
```

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 'C:\My Documents\proccatalogs';
```

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=tcatall;
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 five 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;
quit;
```

SAS Log

```
1    libname perm 'C:\My Documents\proccatalogs';
NOTE: Libref PERM was successfully assigned as follows:
      Engine:        V9
      Physical Name: C:\My Documents\proccatalogs
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=tcatall;
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

CHANGE statement options
(ENTRYTYPE=) or (ET=)

CONTENTS statement options
FILE=

Other statements
MODIFY statement
RUN statement
QUIT statement

Details

This example demonstrates the following actions:

- 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 'C:\My Documents\proccatalogs';

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 'C:\My Documents\proccatalogs';
```

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;
```

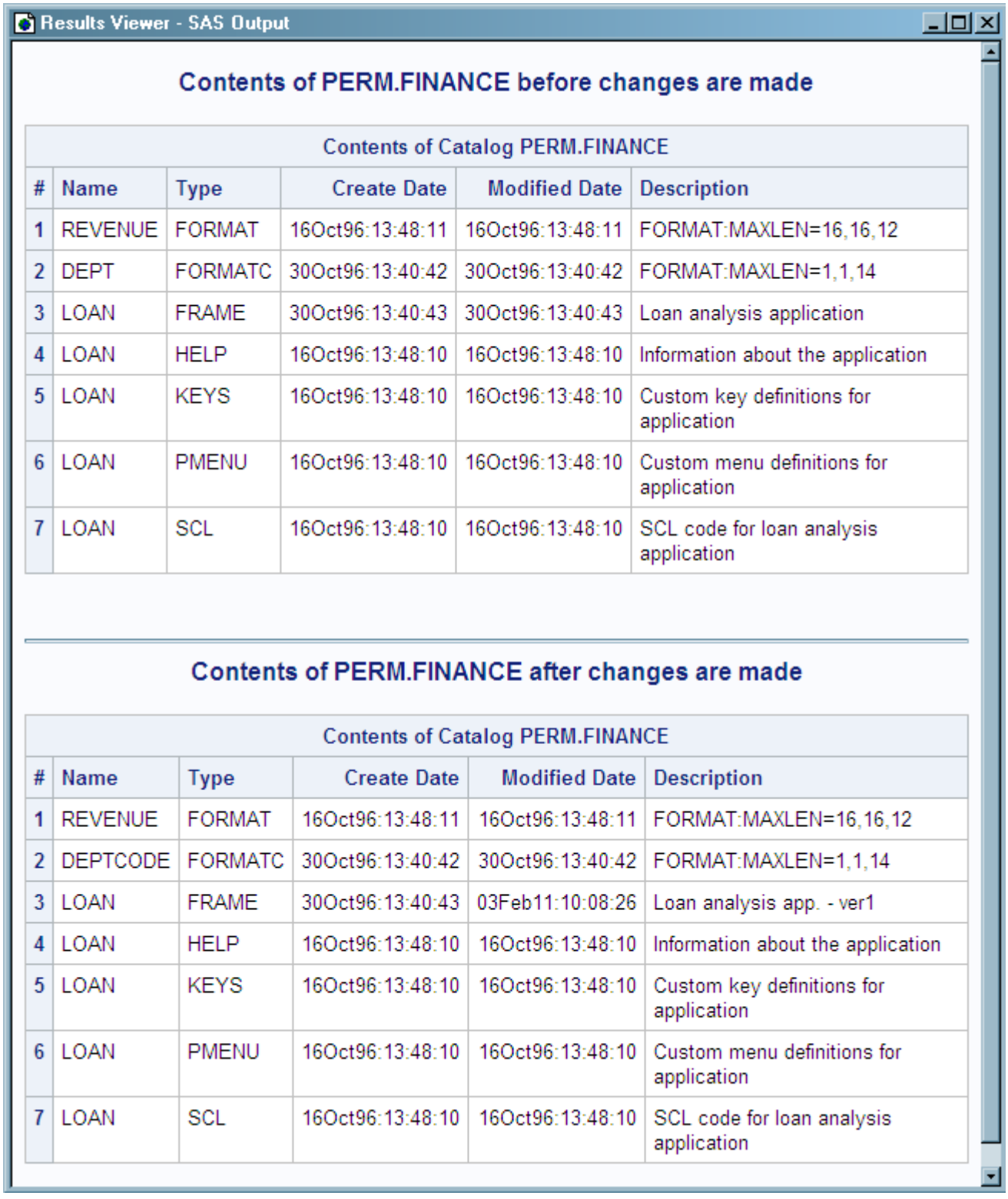
Change entry names. The CHANGE statement changes the name of an entry that contains a user-written character format. (ET=) specifies the entry type.

```
   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



Results Viewer - SAS Output

Contents of PERM.FINANCE before changes are made

Contents of Catalog PERM.FINANCE

#	Name	Type	Create Date	Modified Date	Description
1	REVENUE	FORMAT	16Oct96:13:48:11	16Oct96:13:48:11	FORMAT:MAXLEN=16,16,12
2	DEPT	FORMATC	30Oct96:13:40:42	30Oct96:13:40:42	FORMAT:MAXLEN=1,1,14
3	LOAN	FRAME	30Oct96:13:40:43	30Oct96:13:40:43	Loan analysis application
4	LOAN	HELP	16Oct96:13:48:10	16Oct96:13:48:10	Information about the application
5	LOAN	KEYS	16Oct96:13:48:10	16Oct96:13:48:10	Custom key definitions for application
6	LOAN	PMENU	16Oct96:13:48:10	16Oct96:13:48:10	Custom menu definitions for application
7	LOAN	SCL	16Oct96:13:48:10	16Oct96:13:48:10	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	16Oct96:13:48:11	16Oct96:13:48:11	FORMAT:MAXLEN=16,16,12
2	DEPTCODE	FORMATC	30Oct96:13:40:42	30Oct96:13:40:42	FORMAT:MAXLEN=1,1,14
3	LOAN	FRAME	30Oct96:13:40:43	03Feb11:10:08:26	Loan analysis app. - ver1
4	LOAN	HELP	16Oct96:13:48:10	16Oct96:13:48:10	Information about the application
5	LOAN	KEYS	16Oct96:13:48:10	16Oct96:13:48:10	Custom key definitions for application
6	LOAN	PMENU	16Oct96:13:48:10	16Oct96:13:48:10	Custom menu definitions for application
7	LOAN	SCL	16Oct96:13:48:10	16Oct96:13:48:10	SCL code for loan analysis application

Example 3: Using the FORCE Option with the KILL Option

Features: PROC CATALOG statement:

CATALOG= argument
FORCE option
KILL option
QUIT statement
RUN statement

Details

This example

- 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;
```

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;
```

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
```

Program

```
proc catalog c=work.sasmacr kill force;
run;
quit;
```

Program Description

Add the FORCE option to the KILL option to delete the catalog entries.

```
proc catalog c=work.sasmacr kill force;
run;
quit;
```

SAS Log

```
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
```

Chapter 10

CHART Procedure

Overview: CHART Procedure

What Does the CHART Procedure Do?

The CHART procedure produces vertical and horizontal bar charts, block charts, pie charts, and star charts. These types of charts graphically display values of a variable or a statistic associated with those values. The charted variable can be numeric or character.

PROC CHART is a useful tool that lets you visualize data quickly, but if you need to produce presentation-quality graphics that include color and various fonts, then use

SAS/GRAPH software. The GCHART procedure in SAS/GRAPH software produces the same types of charts as PROC CHART does. In addition, PROC GCHART can produce donut charts.

What Types of Charts Can PROC CHART Create?

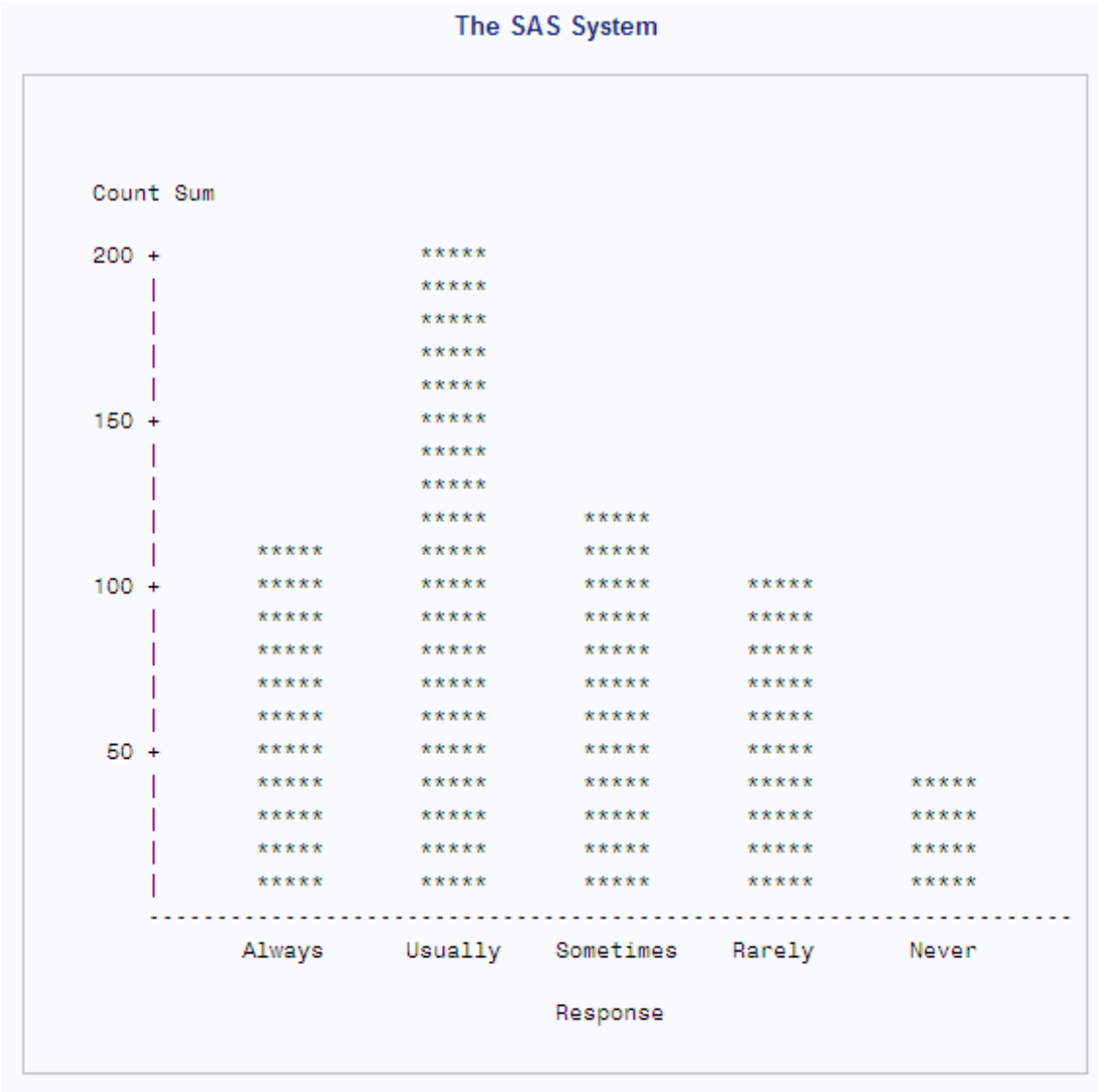
Bar Charts

Horizontal and vertical bar charts display the magnitude of data with bars, each of which represents a category of data. The length or height of the bars represents the value of the chart statistic for each category.

The following output shows a vertical bar chart that displays the number of responses for the five categories from the survey data. The following statements produce the output:

```
proc chart data=survey;
   vbar response / sumvar=count
   axis=0 to 200 by 50
   midpoints='Always' 'Usually'
             'Sometimes' 'Rarely' 'Never';
run;
```

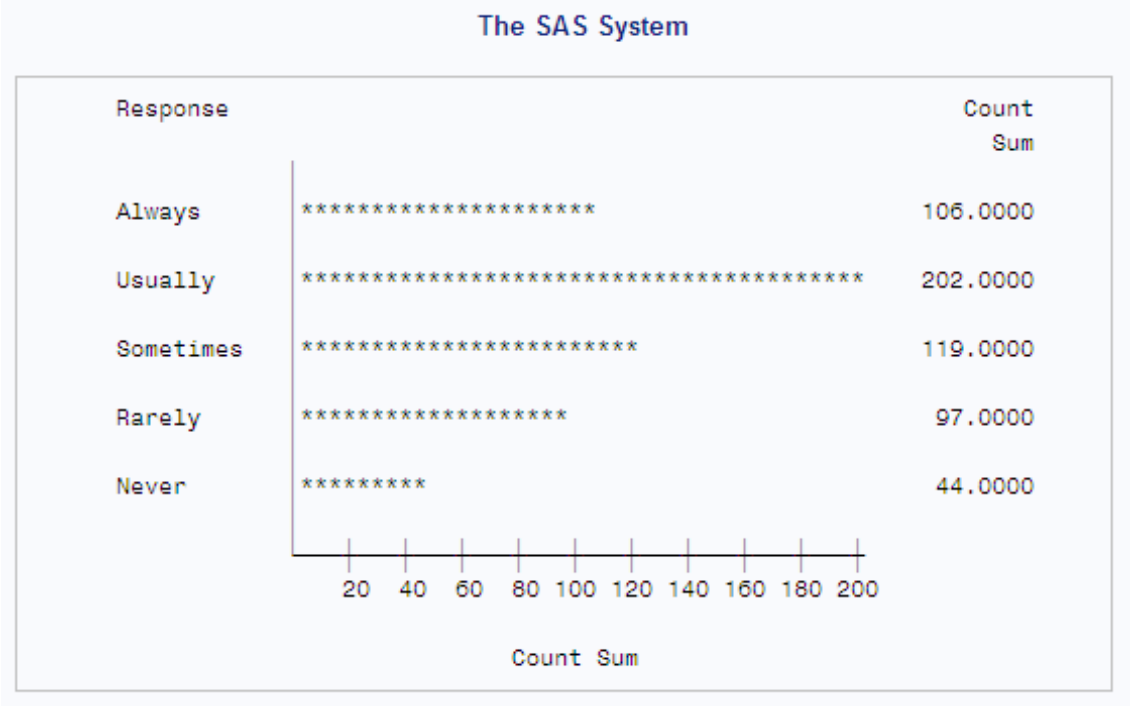
Output 10.1 *Vertical Bar Chart*



The following output shows the same data presented in a horizontal bar chart. The two types of bar charts have essentially the same characteristics, except that horizontal bar charts by default display a table of statistic values to the right of the bars. The following statements produce the output:

```
proc chart data=survey;
   hbar response / sumvar=count
        midpoints='Always' 'Usually'
           'Sometimes' 'Rarely' 'Never';
run;
```

Output 10.2 *Horizontal Bar Chart*

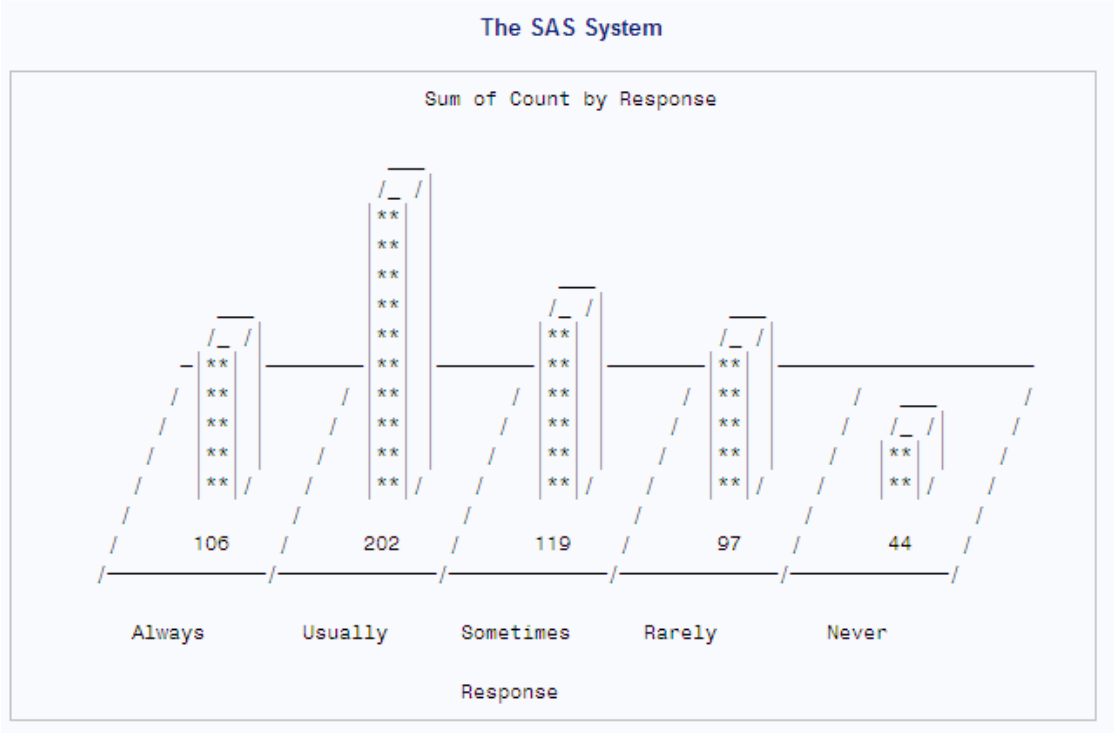


Block Charts

Block charts display the relative magnitude of data by using blocks of varying height, each set in a square that represents a category of data. The following output shows the number of each survey response in the form of a block chart.

```
proc chart data=survey;
    block response / sumvar=count
    midpoints='Always' 'Usually'
              'Sometimes' 'Rarely' 'Never';
run;
```

Output 10.3 *Block Chart*

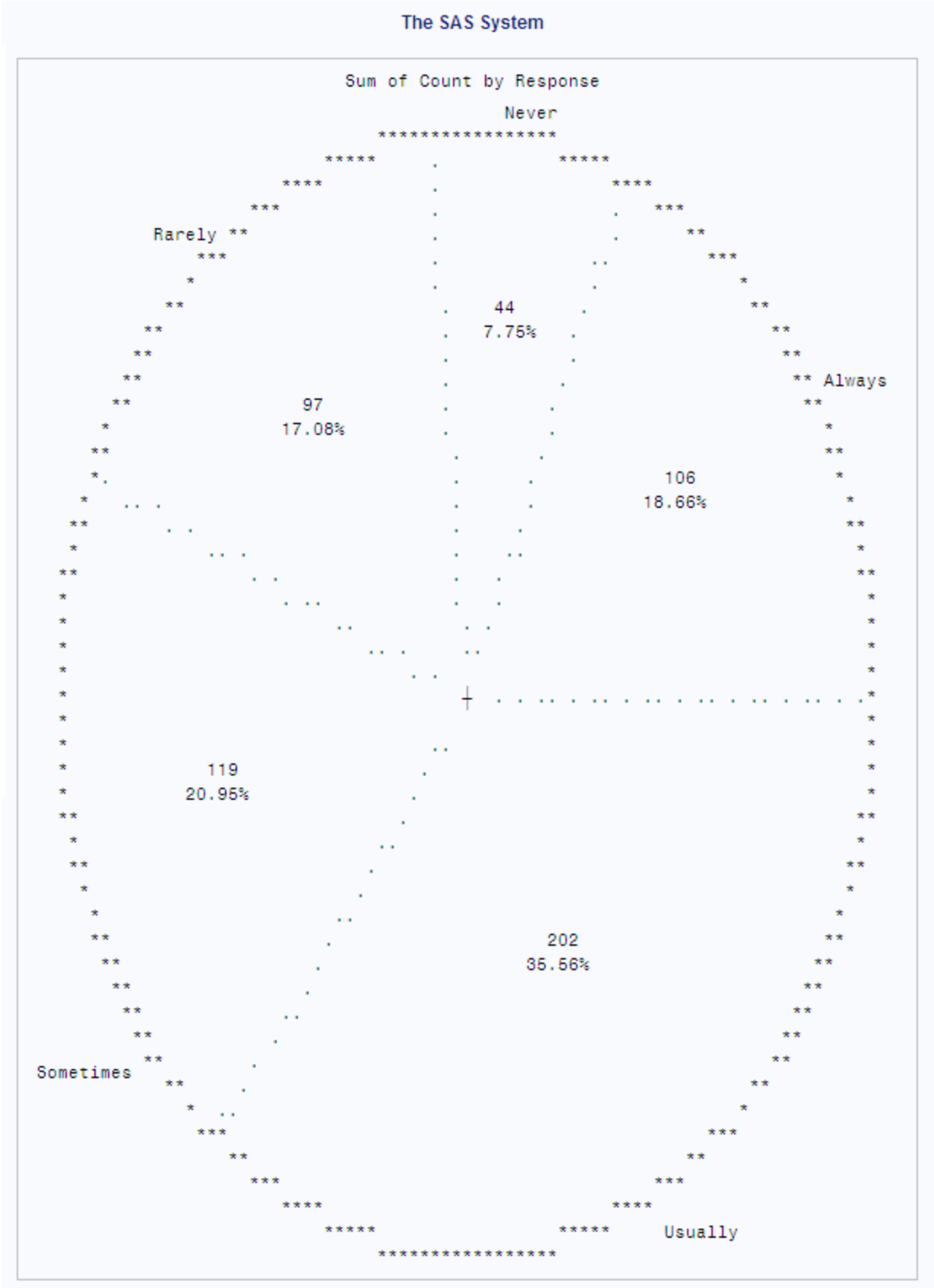


Pie Charts

Pie charts represent the relative contribution of parts to the whole by displaying data as wedge-shaped slices of a circle. Each slice represents a category of the data. The following output shows the survey results divided by response into five pie slices. The following statements produce the output:

```
proc chart data=survey;
   pie response / sumvar=count;
run;
```

Output 10.4 *Pie Chart*

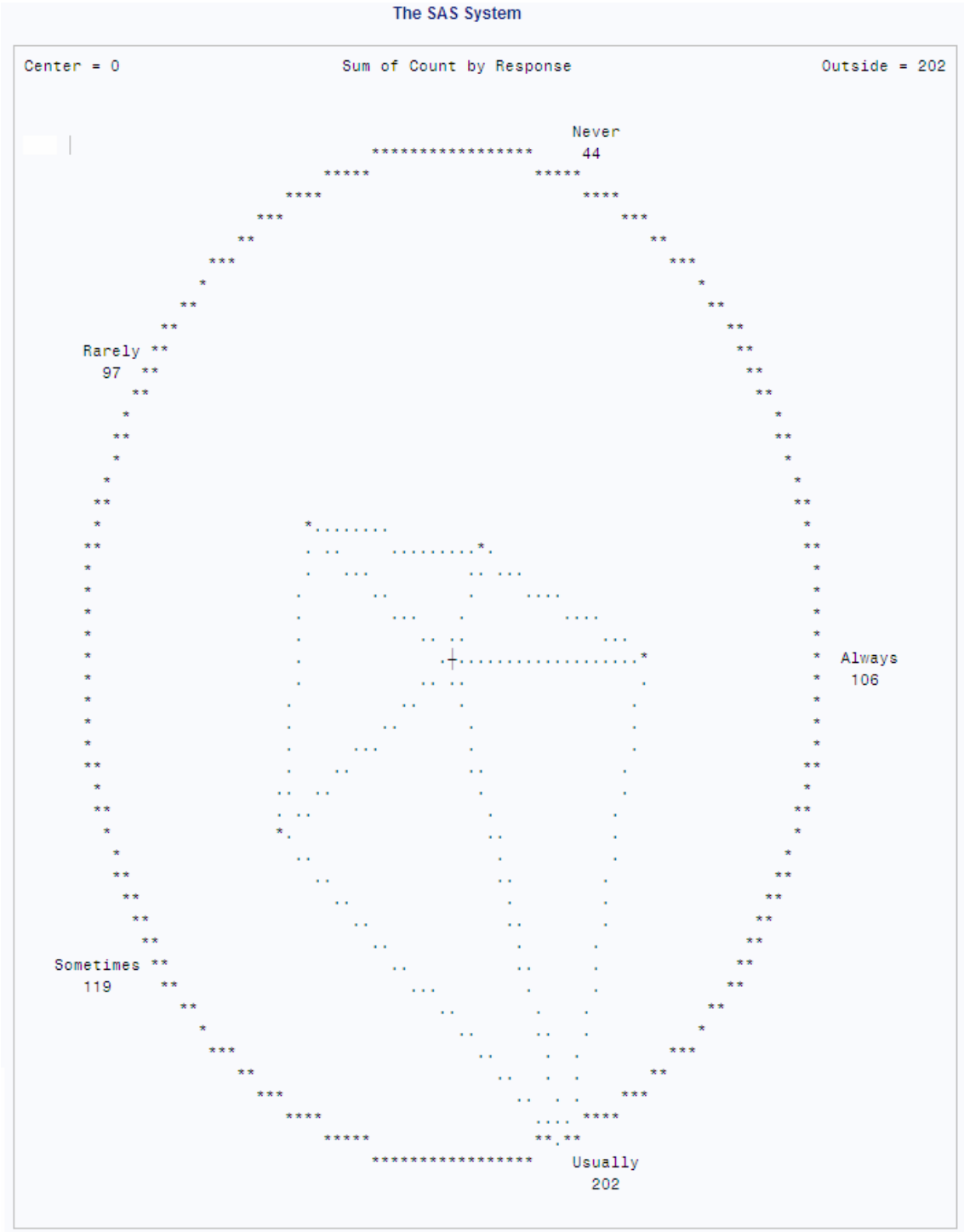


Star Charts

With PROC CHART, you can produce star charts that show group frequencies, totals, or mean values. A star chart is similar to a vertical bar chart, but the bars on a star chart radiate from a center point, like spokes in a wheel. Star charts are commonly used for cyclical data, such as measures taken every month or day or hour. They are also used for data in which the categories have an inherent order ("always" meaning more frequent than "usually," which means more frequent than "sometimes"). The following output shows the survey data displayed in a star chart. The following statements produce the output:

```
proc chart data=survey;
   star response / sumvar=count;
run;
```

Output 10.5 *Star Chart*



Concepts: CHART Procedure

The following are variable characteristics for the CHART procedure:

- Character variables and formats cannot exceed a length of 16.

- For continuous numeric variables, PROC CHART automatically selects display intervals, although you can define interval midpoints.
- For character variables and discrete numeric variables, which contain several distinct values rather than a continuous range, the data values themselves define the intervals.

Syntax: CHART Procedure

Requirement: You must use at least one of the chart-producing statements.

Tips: Supports the Output Delivery System. For details, see Chapter 3, "Output Delivery System: Basic Concepts," in *SAS Output Delivery System: User's Guide*.

You can use the ATTRIB, FORMAT, LABEL, and WHERE statements. For more information, see "Statements with the Same Function in Multiple Procedures" on page 35. You can also use any global statement. For a list, see "Global Statements" in Chapter 1 of *SAS Statements: Reference*.

PROC CHART <*option(s)*>;
BLOCK *variable(s)* </ *option(s)*>;
BY <DESCENDING> *variable-1*
<…<DESCENDING>*variable-n*>
<NOTSORTED>;
HBAR *variable(s)* </ *option(s)*>;
PIE *variable(s)* </ *option(s)*>;
STAR *variable(s)* </ *option(s)*>;
VBAR *variable(s)* </ *option(s)*>;

Statement	Task	Example
"PROC CHART Statement"	Produce a chart	
"BLOCK Statement"	Produce a block chart	Ex. 6
"BY Statement"	Produce a separate chart for each BY group	Ex. 6
"HBAR Statement"	Produce a horizontal bar chart	Ex. 5
"PIE Statement"	Produce a PIE chart	
"STAR Statement"	Produce a STAR chart	
"VBAR Statement"	Produce a vertical bar chart	Ex. 1, Ex. 2, Ex. 3, Ex. 4

PROC CHART Statement

Produces vertical and horizontal bar charts, block charts, pie charts, and star charts.

Syntax

PROC CHART <*option(s)*>;

Optional Arguments

DATA=*SAS-data-set*
identifies the input SAS data set.

Restriction: You cannot use PROC CHART 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 20

FORMCHAR <(*position(s)*)>='*formatting-character(s)*'
defines the characters to use for constructing the horizontal and vertical axes, reference lines, and other structural parts of a chart. It also defines the symbols to use to create the bars, blocks, or sections in the output.

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.

Note: PROC CHART uses 6 of the 20 formatting characters that SAS provides. Table 10.1 on page 212 shows the formatting characters that PROC CHART uses. Figure 10.1 on page 213 illustrates the use of formatting characters commonly used in PROC CHART.

formatting-character(s)
lists the characters to use for the specified positions. PROC CHART assigns characters in *formatting-character(s)* to *position(s)*, in the order which they are listed. For example, the following option assigns the asterisk (*) to the second formatting character, the pound sign (#) to the seventh character, and does not alter the remaining characters:

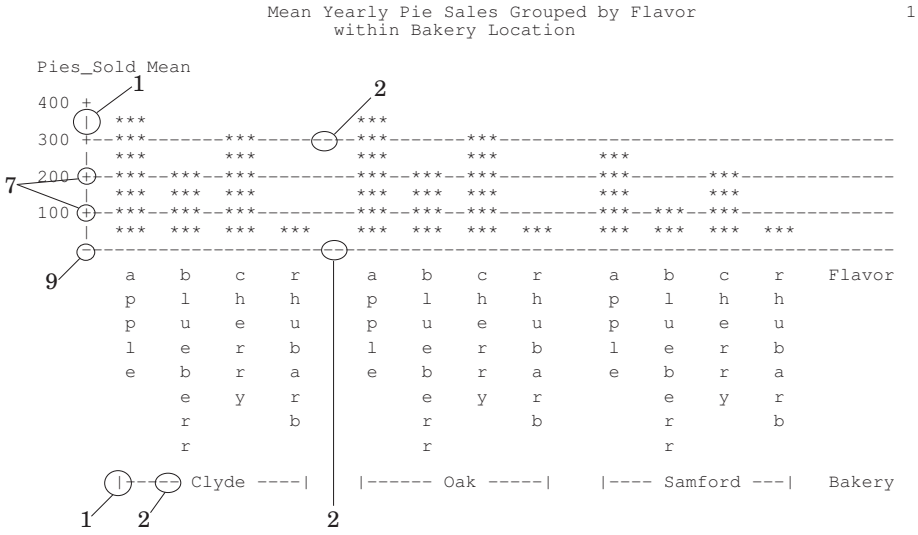
```
formchar(2,7)='*#'
```

Table 10.1 *Formatting Characters Used by PROC CHART*

Position	Default	Used to Draw
1	|	Vertical axes in bar charts, the sides of the blocks in block charts, and reference lines in horizontal bar charts. In side-by-side bar charts, the first and second formatting characters appear around each value of the group variable (below the chart) to indicate the width of each group.
2	-	Horizontal axes in bar charts, the horizontal lines that separate the blocks in a block chart, and reference lines in vertical bar charts. In side-by-side bar charts, the first and second formatting characters appear around each value of the group variable (below the chart) to indicate the width of each group.

Position	Default	Used to Draw
7	+	Tick marks in bar charts and the centers in pie and star charts.
9	-	Intersection of axes in bar charts.
16	/	Ends of blocks and the diagonal lines that separate blocks in a block chart.
20	*	Circles in pie and star charts.

Figure 10.1 *Formatting Characters Commonly Used in PROC CHART Output*



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.

Tip: You can use any character in *formatting-characters*, including hexadecimal characters. If you use hexadecimal characters, then you must put an **x** after the closing quotation mark. For example the following option assigns the hexadecimal character 2-D to the second formatting character, the hexadecimal character 7C to the seventh character, and does not alter the remaining characters:

```
formchar(2,7)='2D7C'x
```

See: For information about which hexadecimal codes to use for which characters, consult the documentation for your hardware.

LPI=*value*

specifies the proportions of PIE and STAR charts. The *value* is determined by

(*lines per inch* / *columns per inch*) * 10

For example, if you have a printer with 8 lines per inch and 12 columns per inch, then specify LPI=6.6667.

Default: 6

BLOCK Statement

Produces a block chart.

Example: "Example 6: Producing Block Charts for BY Groups" on page 246

Syntax

BLOCK *variable(s)* </ *option(s)*>;

Required Argument

variable(s)
specifies the variables for which PROC CHART produces a block chart, one chart for each variable.

Optional Arguments

AXIS=*value-expression*
specifies the values for the response axis, where *value-expression* is a list of individual values, each separated by a space, or a range with a uniform interval for the values. For example, the following range specifies tick marks on a bar chart from 0 to 100 at intervals of 10: `hbar x / axis=0 to 100 by 10;`

Restrictions:
Values must be uniformly spaced, even if you specify them individually.

For frequency charts, values must be integers.

Interactions:
For BLOCK charts, AXIS= sets the scale of the tallest block. To set the scale, PROC CHART uses the maximum value from the AXIS= list. If no value is greater than 0, then PROC CHART ignores the AXIS= option.

If you use AXIS= and the BY statement, then PROC CHART produces uniform axes over BY groups.

CAUTION: Values in value-expression override the range of the data. For example, if the data range is 1 to 10 and you specify a range of 3 to 5, then only the data in the range 3 to 5 appears on the chart. Values out of range produce a warning message in the SAS log.

FREQ=*variable*
specifies a data set variable that represents a frequency count for each observation. Normally, each observation contributes a value of one to the frequency counts. With FREQ=, each observation contributes its value of the FREQ= value.

Restriction: If the FREQ= values are not integers, then PROC CHART truncates them.

Interaction: If you use SUMVAR=, then PROC CHART multiplies the sums by the FREQ= value.

GROUP=*variable*
produces side-by-side charts, with each chart representing the observations that have a common value for the GROUP= variable. The GROUP= variable can be character or numeric and is assumed to be discrete. For example, the following statement produces a frequency bar chart for men and women in each department:

```
vbar gender / group=dept;
```

Missing values for a GROUP= variable are treated as valid levels.

Examples:

"Example 4: Producing Side-by-Side Bar Charts" on page 242

"Example 5: Producing a Horizontal Bar Chart for a Subset of the Data" on page 245

"Example 6: Producing Block Charts for BY Groups" on page 246

G100

specifies that the sum of percentages for each group equals 100. By default, PROC CHART uses 100% as the total sum. For example, if you produce a bar chart that separates males and females into three age categories, then the six bars, by default, add to 100%. However, with G100, the three bars for females add to 100%, and the three bars for males add to 100%.

Interaction: PROC CHART ignores G100 if you omit GROUP=.

LEVELS=*number-of-midpoints*

specifies the number of bars that represent each chart variable when the variables are continuous.

MIDPOINTS=*midpoint-specification* | OLD

defines the range of values that each bar, block, or section represents by specifying the range midpoints.

The value for MIDPOINTS= is one of the following:

midpoint-specification

specifies midpoints, either individually, or across a range at a uniform interval. For example, the following statement produces a chart with five bars. The first bar represents the range of values of X with a midpoint of 10. The second bar represents the range with a midpoint of 20, and so on:

```
vbar x / midpoints=10 20 30 40 50;
```

Here is an example of a midpoint specification for a character variable:

```
vbar x / midpoints='JAN' 'FEB' 'MAR';
```

Here is an example of specifying midpoints across a range at a uniform interval:

```
vbar x / midpoints=10 to 100 by 5;
```

Example:

OLD

specifies an algorithm that PROC CHART used in previous versions of SAS to choose midpoints for continuous variables. The old algorithm was based on the work of Nelder (1976). The current algorithm that PROC CHART uses if you omit OLD is based on the work of Terrell and Scott (1985).

Default: Without MIDPOINTS=, PROC CHART displays the values in the SAS System's normal sorted order.

MISSING

specifies that missing values are valid levels for the chart variable.

NOHEADER

suppresses the default header line printed at the top of a chart.

Alias: NOHEADING

Example: "Example 6: Producing Block Charts for BY Groups" on page 246

NOSYMBOL

suppresses printing of the subgroup symbol or legend table.

Alias: NOLEGEND

Interaction: PROC CHART ignores NOSYMBOL if you omit SUBGROUP=.

SUBGROUP=*variable*
subdivides each bar or block into characters that show the contribution of the values of *variable* to that bar or block. PROC CHART uses the first character of each value to fill in the portion of the bar or block that corresponds to that value, unless more than one value begins with the same first character. In that case, PROC CHART uses the letters A, B, C, and so on, to fill in the bars or blocks. If the variable is formatted, then PROC CHART uses the first character of the formatted value.

The characters used in the chart and the values that they represent are given in a legend at the bottom of the chart. The subgroup symbols are ordered A through Z and 0 through 9 with the characters in ascending order.

PROC CHART calculates the height of a bar or block for each subgroup individually and then rounds the percentage of the total bar up or down. So the total height of the bar can be higher or lower than the same bar without the SUBGROUP= option.

Interaction: If you use both TYPE=MEAN and SUBGROUP=, then PROC CHART first calculates the mean for each variable that is listed in the SUMVAR= option. It then subdivides the bar into the percentages that each subgroup contributes.

Example: "Example 3: Subdividing the Bars into Categories" on page 238

SUMVAR=*variable*
specifies the variable for which either values or means (depending on the value of TYPE=) PROC CHART displays in the chart.

Interaction: If you use SUMVAR= and you use TYPE= with a value other than MEAN or SUM, then TYPE=SUM overrides the specified TYPE= value.

Tip: Both HBAR and VBAR charts can print labels for SUMVAR= variables if you use a LABEL statement.

Examples:
"Example 3: Subdividing the Bars into Categories" on page 238
"Example 4: Producing Side-by-Side Bar Charts" on page 242
"Example 5: Producing a Horizontal Bar Chart for a Subset of the Data" on page 245
"Example 6: Producing Block Charts for BY Groups" on page 246

SYMBOL=*character(s)*
specifies the character or characters that PROC CHART uses in the bars or blocks of the chart when you do not use the SUBGROUP= option.

Default: asterisk (*)

Interaction: If the SAS system option OVP is in effect and if your printing device supports overprinting, then you can specify up to three characters to produce overprinted charts.

Example: "Example 6: Producing Block Charts for BY Groups" on page 246

TYPE=*statistic*
specifies what the bars or sections in the chart represent. The *statistic* is one of the following:

CFREQ
specifies that each bar, block, or section represent the cumulative frequency.

CPERCENT
specifies that each bar, block, or section represent the cumulative percentage.

Alias: CPCT

FREQ
specifies that each bar, block, or section represent the frequency with which a value or range occurs for the chart variable in the data.

MEAN
specifies that each bar, block, or section represent the mean of the SUMVAR= variable across all observations that belong to that bar, block, or section.

Interaction: With TYPE=MEAN, you can compute only MEAN and FREQ statistics.

Example: "Example 4: Producing Side-by-Side Bar Charts" on page 242

PERCENT
specifies that each bar, block, or section represent the percentage of observations that have a given value or that fall into a given range of the chart variable.

Alias: PCT

Example: "Example 2: Producing a Percentage Bar Chart" on page 236

SUM
specifies that each bar, block, or section represent the sum of the SUMVAR= variable for the observations that correspond to each bar, block, or section.

Default: FREQ (unless you use SUMVAR=, which causes a default of SUM)

Interaction: With TYPE=SUM, you can compute only SUM and FREQ statistics.

Details

Statement Results

Because each block chart must fit on one output page, you might have to adjust the SAS system options LINESIZE= and PAGESIZE= if you have a large number of charted values for the BLOCK variable and for the variable specified in the GROUP= option.

The following table shows the maximum number of charted values of BLOCK variables for selected LINESIZE= (LS=) specifications that can fit on a 66-line page.

Table 10.2 *Maximum Number of Bars of BLOCK Variables*

GROUP= Value	LS= 132	LS= 120	LS= 105	LS= 90	LS= 76	LS= 64
0,1	9	8	7	6	5	4
2	8	8	7	6	5	4
3	8	7	6	5	4	3
4	7	7	6	5	4	3
5,6	7	6	5	4	3	2

If the value of any GROUP= level is longer than three characters, then the maximum number of charted values for the BLOCK variable that can fit might be reduced by one. BLOCK level values truncate to 12 characters. If you exceed these limits, then PROC CHART produces a horizontal bar chart instead.

BY Statement

Produces a separate chart for each BY group.

See: "BY" on page 36

Example: "Example 6: Producing Block Charts for BY Groups" on page 246

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 the observations in the data set must either 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.

HBAR Statement

Produces a horizontal bar chart.

Tip: HBAR charts can print either the name or the label of the chart variable.

See: "Example 5: Producing a Horizontal Bar Chart for a Subset of the Data" on page 245

Syntax

HBAR *variable(s)* </ *option(s)*>;

Required Argument

variable(s)
specifies the variables for which PROC CHART produces a horizontal bar chart, one chart for each variable.

Optional Arguments

ASCENDING
prints the bars and any associated statistics in ascending order of size within groups.

Alias: ASC

AXIS=*value-expression*
specifies the values for the response axis, where *value-expression* is a list of individual values, each separated by a space, or a range with a uniform interval for the values. For example, the following range specifies tick marks on a bar chart from 0 to 100 at intervals of 10: `hbar x / axis=0 to 100 by 10;`

Restrictions:
Values must be uniformly spaced, even if you specify them individually.

For frequency charts, values must be integers.

Interactions:
For HBAR and VBAR charts, AXIS= determines tick marks on the response axis. If the AXIS= specification contains only one value, then the value determines the minimum tick mark if the value is less than 0, or determines the maximum tick mark if the value is greater than 0.

If you use AXIS= and the BY statement, then PROC CHART produces uniform axes over BY groups.

CAUTION: Values in value-expression override the range of the data. For example, if the data range is 1 to 10 and you specify a range of 3 to 5, then only the data in the range 3 to 5 appears on the chart. Values out of range produce a warning message in the SAS log.

CFREQ
prints the cumulative frequency.

Restriction: Available only in the HBAR statement

CPERCENT
prints the cumulative percentages.

Restriction: Available only in the HBAR statement

DISCRETE
specifies that a numeric chart variable is discrete rather than continuous. Without DISCRETE, PROC CHART assumes that all numeric variables are continuous and automatically chooses intervals for them unless you use MIDPOINTS= or LEVELS=.

FREQ
prints the frequency of each bar to the side of the chart.

Restriction: Available only in the HBAR statement

FREQ=*variable*
specifies a data set variable that represents a frequency count for each observation. Normally, each observation contributes a value of one to the frequency counts. With FREQ=, each observation contributes its value of the FREQ= value.

Restriction: If the FREQ= values are not integers, then PROC CHART truncates them.

Interaction: If you use SUMVAR=, then PROC CHART multiplies the sums by the FREQ= value.

GROUP=*variable*
produces side-by-side charts, with each chart representing the observations that have a common value for the GROUP= variable. The GROUP= variable can be character or numeric and is assumed to be discrete. For example, the following statement produces a frequency bar chart for men and women in each department:

```
vbar gender / group=dept;
```

Missing values for a GROUP= variable are treated as valid levels.

Examples:
"Example 4: Producing Side-by-Side Bar Charts" on page 242
"Example 5: Producing a Horizontal Bar Chart for a Subset of the Data" on page 245
"Example 6: Producing Block Charts for BY Groups" on page 246

GSPACE=*n*
specifies the amount of extra space between groups of bars. Use GSPACE=0 to leave no extra space between adjacent groups of bars.

Interaction: PROC CHART ignores GSPACE= if you omit GROUP=

G100
specifies that the sum of percentages for each group equals 100. By default, PROC CHART uses 100% as the total sum. For example, if you produce a bar chart that separates males and females into three age categories, then the six bars, by default, add to 100%. However, with G100, the three bars for females add to 100%, and the three bars for males add to 100%.

Interaction: PROC CHART ignores G100 if you omit GROUP=.

LEVELS=*number-of-midpoints*
specifies the number of bars that represent each chart variable when the variables are continuous.

MEAN
prints the mean of the observations represented by each bar.

Restrictions:
Available only when you use SUMVAR= and TYPE=
Not available when TYPE=CFREQ, CPERCENT, FREQ, or PERCENT

MISSING
specifies that missing values are valid levels for the chart variable.

NOSTATS
suppresses the statistics on a horizontal bar chart.

Alias: NOSTAT

NOSYMBOL
suppresses printing of the subgroup symbol or legend table.

Alias: NOLEGEND

Interaction: PROC CHART ignores NOSYMBOL if you omit SUBGROUP=.

NOZEROS
suppresses any bar with zero frequency.

PERCENT
prints the percentages of observations having a given value for the chart variable.

REF=*value(s)*
draws reference lines on the response axis at the specified positions.

Tip: The REF= values should correspond to values of the TYPE= statistic.

Example: "Example 4: Producing Side-by-Side Bar Charts" on page 242

SPACE=*n*
specifies the amount of space between individual bars.

Tips:
Use SPACE=0 to leave no space between adjacent bars.

Use the GSPACE= option to specify the amount of space between the bars within each group.

SUBGROUP=*variable*
subdivides each bar or block into characters that show the contribution of the values of *variable* to that bar or block. PROC CHART uses the first character of each value to fill in the portion of the bar or block that corresponds to that value, unless more than one value begins with the same first character. In that case, PROC CHART uses the letters A, B, C, and so on, to fill in the bars or blocks. If the variable is formatted, then PROC CHART uses the first character of the formatted value.

The characters used in the chart and the values that they represent are given in a legend at the bottom of the chart. The subgroup symbols are ordered A through Z and 0 through 9 with the characters in ascending order.

PROC CHART calculates the height of a bar or block for each subgroup individually and then rounds the percentage of the total bar up or down. So the total height of the bar can be higher or lower than the same bar without the SUBGROUP= option.

Interaction: If you use both TYPE=MEAN and SUBGROUP=, then PROC CHART first calculates the mean for each variable that is listed in the SUMVAR= option. It then subdivides the bar into the percentages that each subgroup contributes.

Example: "Example 3: Subdividing the Bars into Categories" on page 238

SUM
prints the total number of observations that each bar represents.

Restrictions:
Available only when you use both SUMVAR= and TYPE=

Not available when TYPE=CFREQ, CPERCENT, FREQ, or PERCENT

SUMVAR=*variable*
specifies the variable for which either values or means (depending on the value of TYPE=) PROC CHART displays in the chart.

Interaction: If you use SUMVAR= and you use TYPE= with a value other than MEAN or SUM, then TYPE=SUM overrides the specified TYPE= value.

Tip: HBAR charts can print labels for SUMVAR= variables if you use a LABEL statement.

Examples:
"Example 3: Subdividing the Bars into Categories" on page 238

"Example 4: Producing Side-by-Side Bar Charts" on page 242

"Example 5: Producing a Horizontal Bar Chart for a Subset of the Data" on page 245

"Example 6: Producing Block Charts for BY Groups" on page 246

SYMBOL=*character(s)*
specifies the character or characters that PROC CHART uses in the bars or blocks of the chart when you do not use the SUBGROUP= option.

Default: asterisk (*)

Interaction: If the SAS system option OVP is in effect and if your printing device supports overprinting, then you can specify up to three characters to produce overprinted charts.

Example: "Example 6: Producing Block Charts for BY Groups" on page 246

TYPE=*statistic*

specifies what the bars or sections in the chart represent. The *statistic* is one of the following:

CFREQ

specifies that each bar, block, or section represent the cumulative frequency.

CPERCENT

specifies that each bar, block, or section represent the cumulative percentage.

Alias: CPCT

FREQ

specifies that each bar, block, or section represent the frequency with which a value or range occurs for the chart variable in the data.

MEAN

specifies that each bar, block, or section represent the mean of the SUMVAR= variable across all observations that belong to that bar, block, or section.

Interaction: With TYPE=MEAN, you can compute only MEAN and FREQ statistics.

Example: "Example 4: Producing Side-by-Side Bar Charts" on page 242

PERCENT

specifies that each bar, block, or section represent the percentage of observations that have a given value or that fall into a given range of the chart variable.

Alias: PCT

Example: "Example 2: Producing a Percentage Bar Chart" on page 236

SUM

specifies that each bar, block, or section represent the sum of the SUMVAR= variable for the observations that correspond to each bar, block, or section.

Default: FREQ (unless you use SUMVAR=, which causes a default of SUM)

Interaction: With TYPE=SUM, you can compute only SUM and FREQ statistics.

WIDTH=*n*

specifies the width of the bars on bar charts.

Details

Statement Results

Each chart occupies one or more output pages, depending on the number of bars; each bar occupies one line, by default.

By default, for horizontal bar charts of TYPE=FREQ, CFREQ, PCT, or CPCT, PROC CHART prints the following statistics: frequency, cumulative frequency, percentage, and cumulative percentage. If you use one or more of the statistics options, then PROC CHART prints only the statistics that you request, plus the frequency.

PIE Statement

Produces a pie chart.

Syntax

PIE *variable(s)* </ *option(s)*>;

Required Argument

variable(s)
specifies the variables for which PROC CHART produces a pie chart, one chart for each variable.

Optional Arguments

FREQ=*variable*
specifies a data set variable that represents a frequency count for each observation. Normally, each observation contributes a value of one to the frequency counts. With FREQ=, each observation contributes its value of the FREQ= value.

Restriction: If the FREQ= values are not integers, then PROC CHART truncates them.

Interaction: If you use SUMVAR=, then PROC CHART multiplies the sums by the FREQ= value.

LEVELS=*number-of-midpoints*
specifies the number of bars that represent each chart variable when the variables are continuous.

MIDPOINTS=*midpoint-specification* | OLD
defines the range of values that each bar, block, or section represents by specifying the range midpoints.

The value for MIDPOINTS= is one of the following:

midpoint-specification
specifies midpoints, either individually, or across a range at a uniform interval. For example, the following statement produces a chart with five bars. The first bar represents the range of values of X with a midpoint of 10. The second bar represents the range with a midpoint of 20, and so on:

```
vbar x / midpoints=10 20 30 40 50;
```

Here is an example of a midpoint specification for a character variable:

```
vbar x / midpoints='JAN' 'FEB' 'MAR';
```

Here is an example of specifying midpoints across a range at a uniform interval:

```
vbar x / midpoints=10 to 100 by 5;
```

Example:

OLD
specifies an algorithm that PROC CHART used in previous versions of SAS to choose midpoints for continuous variables. The old algorithm was based on the work of Nelder (1976). The current algorithm that PROC CHART uses if you omit OLD is based on the work of Terrell and Scott (1985).

Default: Without MIDPOINTS=, PROC CHART displays the values in the SAS System's normal sorted order.

MISSING
specifies that missing values are valid levels for the chart variable.

NOHEADER
suppresses the default header line printed at the top of a chart.

Alias: NOHEADING

Example: "Example 6: Producing Block Charts for BY Groups" on page 246

SUMVAR=*variable*
specifies the variable for which either values or means (depending on the value of TYPE=) PROC CHART displays in the chart.

Interaction: If you use SUMVAR= and you use TYPE= with a value other than MEAN or SUM, then TYPE=SUM overrides the specified TYPE= value.

Tip: Both HBAR and VBAR charts can print labels for SUMVAR= variables if you use a LABEL statement.

Examples:
"Example 3: Subdividing the Bars into Categories" on page 238
"Example 4: Producing Side-by-Side Bar Charts" on page 242
"Example 5: Producing a Horizontal Bar Chart for a Subset of the Data" on page 245
"Example 6: Producing Block Charts for BY Groups" on page 246

TYPE=*statistic*
specifies what the bars or sections in the chart represent. The *statistic* is one of the following:

CFREQ
specifies that each bar, block, or section represent the cumulative frequency.

CPERCENT
specifies that each bar, block, or section represent the cumulative percentage.

Alias: CPCT

FREQ
specifies that each bar, block, or section represent the frequency with which a value or range occurs for the chart variable in the data.

MEAN
specifies that each bar, block, or section represent the mean of the SUMVAR= variable across all observations that belong to that bar, block, or section.

Interaction: With TYPE=MEAN, you can compute only MEAN and FREQ statistics.

Example: "Example 4: Producing Side-by-Side Bar Charts" on page 242

PERCENT
specifies that each bar, block, or section represent the percentage of observations that have a given value or that fall into a given range of the chart variable.

Alias: PCT

Example: "Example 2: Producing a Percentage Bar Chart" on page 236

SUM
specifies that each bar, block, or section represent the sum of the SUMVAR= variable for the observations that correspond to each bar, block, or section.

Default: FREQ (unless you use SUMVAR=, which causes a default of SUM)

Interaction: With TYPE=SUM, you can compute only SUM and FREQ statistics.

Details

Statement Results

PROC CHART determines the number of slices for the pie in the same way that it determines the number of bars for vertical bar charts. Any slices of the pie accounting for less than three print positions are grouped together into an "OTHER" category.

The pie's size is determined only by the SAS system options LINESIZE= and PAGESIZE=. By default, the pie looks elliptical if your printer does not print 6 lines per inch and 10 columns per inch. To make a circular pie chart on a printer that does not print 6 lines and 10 columns per inch, use the LPI= option in the PROC CHART statement. See the description of "LPI=value" on page 213 for the formula that gives you the proper LPI= value for your printer.

If you try to create a PIE chart for a variable with more than 50 levels, then PROC CHART produces a horizontal bar chart instead.

STAR Statement

Produces a star chart.

Syntax

STAR *variable(s)* </ *option(s)*>;

Required Argument

variable(s)
specifies the variables for which PROC CHART produces a star chart, one chart for each variable.

Optional Arguments

AXIS=*value-expression*
specifies the values for the response axis, where *value-expression* is a list of individual values, each separated by a space, or a range with a uniform interval for the values. For example, the following range specifies tick marks on a bar chart from 0 to 100 at intervals of 10: `hbar x / axis=0 to 100 by 10;`

Restrictions:
Values must be uniformly spaced, even if you specify them individually.

For frequency charts, values must be integers.

Interactions:
For STAR charts, a single AXIS= value sets the minimum (the center of the chart) if the value is less than zero, or sets the maximum (the outside circle) if the value is greater than zero. If the AXIS= specification contains more than one value, then PROC CHART uses the minimum and maximum values from the list.

If you use AXIS= and the BY statement, then PROC CHART produces uniform axes over BY groups.

CAUTION: Values in value-expression override the range of the data. For example, if the data range is 1 to 10 and you specify a range of 3 to 5, then only the data in the range 3 to 5 appears on the chart. Values out of range produce a warning message in the SAS log.

FREQ=*variable*
specifies a data set variable that represents a frequency count for each observation. Normally, each observation contributes a value of one to the frequency counts. With FREQ=, each observation contributes its value of the FREQ= value.

Restriction: If the FREQ= values are not integers, then PROC CHART truncates them.

Interaction: If you use SUMVAR=, then PROC CHART multiplies the sums by the FREQ= value.

LEVELS=*number-of-midpoints*
specifies the number of bars that represent each chart variable when the variables are continuous.

MIDPOINTS=*midpoint-specification* | OLD
defines the range of values that each bar, block, or section represents by specifying the range midpoints.

The value for MIDPOINTS= is one of the following:

midpoint-specification
specifies midpoints, either individually, or across a range at a uniform interval. For example, the following statement produces a chart with five bars. The first bar represents the range of values of X with a midpoint of 10. The second bar represents the range with a midpoint of 20, and so on:

```
vbar x / midpoints=10 20 30 40 50;
```

Here is an example of a midpoint specification for a character variable:

```
vbar x / midpoints='JAN' 'FEB' 'MAR';
```

Here is an example of specifying midpoints across a range at a uniform interval:

```
vbar x / midpoints=10 to 100 by 5;
```

Example:

OLD
specifies an algorithm that PROC CHART used in previous versions of SAS to choose midpoints for continuous variables. The old algorithm was based on the work of Nelder (1976). The current algorithm that PROC CHART uses if you omit OLD is based on the work of Terrell and Scott (1985).

Default: Without MIDPOINTS=, PROC CHART displays the values in the SAS System's normal sorted order.

MISSING
specifies that missing values are valid levels for the chart variable.

NOHEADER
suppresses the default header line printed at the top of a chart.

Alias: NOHEADING

Example: “Example 6: Producing Block Charts for BY Groups” on page 246

SUMVAR=*variable*
specifies the variable for which either values or means (depending on the value of TYPE=) PROC CHART displays in the chart.

Interaction: If you use SUMVAR= and you use TYPE= with a value other than MEAN or SUM, then TYPE=SUM overrides the specified TYPE= value.

Tip: Both HBAR and VBAR charts can print labels for SUMVAR= variables if you use a LABEL statement.

Examples:

"Example 3: Subdividing the Bars into Categories" on page 238

"Example 4: Producing Side-by-Side Bar Charts" on page 242

"Example 5: Producing a Horizontal Bar Chart for a Subset of the Data" on page 245

"Example 6: Producing Block Charts for BY Groups" on page 246

TYPE=*statistic*

specifies what the bars or sections in the chart represent. The *statistic* is one of the following:

CFREQ

specifies that each bar, block, or section represent the cumulative frequency.

CPERCENT

specifies that each bar, block, or section represent the cumulative percentage.

Alias: CPCT

FREQ

specifies that each bar, block, or section represent the frequency with which a value or range occurs for the chart variable in the data.

MEAN

specifies that each bar, block, or section represent the mean of the SUMVAR= variable across all observations that belong to that bar, block, or section.

Interaction: With TYPE=MEAN, you can compute only MEAN and FREQ statistics.

Example: "Example 4: Producing Side-by-Side Bar Charts" on page 242

PERCENT

specifies that each bar, block, or section represent the percentage of observations that have a given value or that fall into a given range of the chart variable.

Alias: PCT

Example: "Example 2: Producing a Percentage Bar Chart" on page 236

SUM

specifies that each bar, block, or section represent the sum of the SUMVAR= variable for the observations that correspond to each bar, block, or section.

Default: FREQ (unless you use SUMVAR=, which causes a default of SUM)

Interaction: With TYPE=SUM, you can compute only SUM and FREQ statistics.

Details

Statement Results

The number of points in the star is determined in the same way as the number of bars for vertical bar charts.

If all the data values are positive, then the center of the star represents zero and the outside circle represents the maximum value. If any data values are negative, then the center represents the minimum. See the description of the AXIS=*value expression* for more information about how to specify maximum and minimum values. For information

about how to specify the proportion of the chart, see the description of the "LPI=value" on page 213.

If you try to create a star chart for a variable with more than 24 levels, then PROC CHART produces a horizontal bar chart instead.

VBAR Statement

Produces a vertical bar chart.

Examples: "Example 1: Producing a Simple Frequency Count" on page 233
"Example 2: Producing a Percentage Bar Chart" on page 236
"Example 3: Subdividing the Bars into Categories" on page 238
"Example 4: Producing Side-by-Side Bar Charts" on page 242

Syntax

VBAR *variable(s)* </ *option(s)*>;

Required Argument

variable(s)
specifies the variables for which PROC CHART produces a vertical bar chart, one chart for each variable.

Optional Arguments

ASCENDING
prints the bars and any associated statistics in ascending order of size within groups.

Alias: ASC

AXIS=*value-expression*
specifies the values for the response axis, where *value-expression* is a list of individual values, each separated by a space, or a range with a uniform interval for the values. For example, the following range specifies tick marks on a bar chart from 0 to 100 at intervals of 10: `hbar x / axis=0 to 100 by 10;`

Restrictions:
Values must be uniformly spaced, even if you specify them individually.

For frequency charts, values must be integers.

Interactions:
For HBAR and VBAR charts, AXIS= determines tick marks on the response axis. If the AXIS= specification contains only one value, then the value determines the minimum tick mark if the value is less than 0, or determines the maximum tick mark if the value is greater than 0.

If you use AXIS= and the BY statement, then PROC CHART produces uniform axes over BY groups.

CAUTION: Values in value-expression override the range of the data. For example, if the data range is 1 to 10 and you specify a range of 3 to 5, then only the data in the range 3 to 5 appears on the chart. Values out of range produce a warning message in the SAS log.

DISCRETE
specifies that a numeric chart variable is discrete rather than continuous. Without DISCRETE, PROC CHART assumes that all numeric variables are continuous and automatically chooses intervals for them unless you use MIDPOINTS= or LEVELS=.

FREQ=*variable*
specifies a data set variable that represents a frequency count for each observation. Normally, each observation contributes a value of one to the frequency counts. With FREQ=, each observation contributes its value of the FREQ= value.

Restriction: If the FREQ= values are not integers, then PROC CHART truncates them.

Interaction: If you use SUMVAR=, then PROC CHART multiplies the sums by the FREQ= value.

GROUP=*variable*
produces side-by-side charts, with each chart representing the observations that have a common value for the GROUP= variable. The GROUP= variable can be character or numeric and is assumed to be discrete. For example, the following statement produces a frequency bar chart for men and women in each department:

```
vbar gender / group=dept;
```

Missing values for a GROUP= variable are treated as valid levels.

Examples:
"Example 4: Producing Side-by-Side Bar Charts" on page 242

"Example 5: Producing a Horizontal Bar Chart for a Subset of the Data" on page 245

"Example 6: Producing Block Charts for BY Groups" on page 246

GSPACE=*n*
specifies the amount of extra space between groups of bars. Use GSPACE=0 to leave no extra space between adjacent groups of bars.

Interaction: PROC CHART ignores GSPACE= if you omit GROUP=

G100
specifies that the sum of percentages for each group equals 100. By default, PROC CHART uses 100% as the total sum. For example, if you produce a bar chart that separates males and females into three age categories, then the six bars, by default, add to 100%. However, with G100, the three bars for females add to 100%, and the three bars for males add to 100%.

Interaction: PROC CHART ignores G100 if you omit GROUP=.

LEVELS=*number-of-midpoints*
specifies the number of bars that represent each chart variable when the variables are continuous.

MIDPOINTS=*midpoint-specification* | OLD
defines the range of values that each bar, block, or section represents by specifying the range midpoints.

The value for MIDPOINTS= is one of the following:

midpoint-specification
specifies midpoints, either individually, or across a range at a uniform interval. For example, the following statement produces a chart with five bars; the first bar represents the range of values of X with a midpoint of 10, the second bar represents the range with a midpoint of 20, and so on:

```
vbar x / midpoints=10 20 30 40 50;
```

Here is an example of a midpoint specification for a character variable:

```
vbar x / midpoints='JAN' 'FEB' 'MAR';
```

Here is an example of specifying midpoints across a range at a uniform interval:

```
vbar x / midpoints=10 to 100 by 5;
```

OLD
specifies an algorithm that PROC CHART used in previous versions of SAS to choose midpoints for continuous variables. The old algorithm was based on the work of Nelder (1976). The current algorithm that PROC CHART uses if you omit OLD is based on the work of Terrell and Scott (1985).

Default: Without MIDPOINTS=, PROC CHART displays the values in the SAS System's normal sorted order.

Restriction: When the VBAR variables are numeric, the midpoints must be given in ascending order.

MISSING
specifies that missing values are valid levels for the chart variable.

NOSYMBOL
suppresses printing of the subgroup symbol or legend table.

Alias: NOLEGEND

Interaction: PROC CHART ignores NOSYMBOL if you omit SUBGROUP=.

NOZEROS
suppresses any bar with zero frequency.

REF=*value(s)*
draws reference lines on the response axis at the specified positions.

Tip: The REF= values should correspond to values of the TYPE= statistic.

Example: "Example 4: Producing Side-by-Side Bar Charts" on page 242

SPACE=*n*
specifies the amount of space between individual bars.

Tips:
Use SPACE=0 to leave no space between adjacent bars.

Use the GSPACE= option to specify the amount of space between the bars within each group.

SUBGROUP=*variable*
subdivides each bar or block into characters that show the contribution of the values of *variable* to that bar or block. PROC CHART uses the first character of each value to fill in the portion of the bar or block that corresponds to that value, unless more than one value begins with the same first character. In that case, PROC CHART uses the letters A, B, C, and so on, to fill in the bars or blocks. If the variable is formatted, then PROC CHART uses the first character of the formatted value.

The characters used in the chart and the values that they represent are given in a legend at the bottom of the chart. The subgroup symbols are ordered A through Z and 0 through 9 with the characters in ascending order.

PROC CHART calculates the height of a bar or block for each subgroup individually and then rounds the percentage of the total bar up or down. So the total height of the bar can be higher or lower than the same bar without the SUBGROUP= option.

Interaction: If you use both TYPE=MEAN and SUBGROUP=, then PROC CHART first calculates the mean for each variable that is listed in the

SUMVAR= option. It then subdivides the bar into the percentages that each subgroup contributes.

Example: "Example 3: Subdividing the Bars into Categories" on page 238

SUMVAR=*variable*
specifies the variable for which either values or means (depending on the value of TYPE=) PROC CHART displays in the chart.

Interaction: If you use SUMVAR= and you use TYPE= with a value other than MEAN or SUM, then TYPE=SUM overrides the specified TYPE= value.

Tip: VBAR charts can print labels for SUMVAR= variables if you use a LABEL statement.

Examples:
"Example 3: Subdividing the Bars into Categories" on page 238
"Example 4: Producing Side-by-Side Bar Charts" on page 242
"Example 5: Producing a Horizontal Bar Chart for a Subset of the Data" on page 245
"Example 6: Producing Block Charts for BY Groups" on page 246

SYMBOL=*character(s)*
specifies the character or characters that PROC CHART uses in the bars or blocks of the chart when you do not use the SUBGROUP= option.

Default: asterisk (*)

Interaction: If the SAS system option OVP is in effect and if your printing device supports overprinting, then you can specify up to three characters to produce overprinted charts.

Example: "Example 6: Producing Block Charts for BY Groups" on page 246

TYPE=*statistic*
specifies what the bars or sections in the chart represent. The *statistic* is one of the following:

CFREQ
specifies that each bar, block, or section represent the cumulative frequency.

CPERCENT
specifies that each bar, block, or section represent the cumulative percentage.

Alias: CPCT

FREQ
specifies that each bar, block, or section represent the frequency with which a value or range occurs for the chart variable in the data.

MEAN
specifies that each bar, block, or section represent the mean of the SUMVAR= variable across all observations that belong to that bar, block, or section.

Interaction: With TYPE=MEAN, you can compute only MEAN and FREQ statistics.

Example: "Example 4: Producing Side-by-Side Bar Charts" on page 242

PERCENT
specifies that each bar, block, or section represent the percentage of observations that have a given value or that fall into a given range of the chart variable.

Alias: PCT

Example: "Example 2: Producing a Percentage Bar Chart" on page 236

SUM
specifies that each bar, block, or section represent the sum of the SUMVAR= variable for the observations that correspond to each bar, block, or section.

Default: FREQ (unless you use SUMVAR=, which causes a default of SUM)

Interaction: With TYPE=SUM, you can compute only SUM and FREQ statistics.

WIDTH=*n*
specifies the width of the bars on bar charts.

Details

Statement Results

PROC CHART prints one page per chart. Along the vertical axis, PROC CHART describes the chart frequency, the cumulative frequency, the chart percentage, the cumulative percentage, the sum, or the mean. At the bottom of each bar, PROC CHART prints a value according to the value of the TYPE= option, if specified. For character variables or discrete numeric variables, this value is the actual value represented by the bar. For continuous numeric variables, the value gives the midpoint of the interval represented by the bar.

PROC CHART can automatically scale the vertical axis, determine the bar width, and choose spacing between the bars. However, by using options, you can choose bar intervals and the number of bars, include missing values in the chart, produce side-by-side charts, and subdivide the bars. If the number of characters per line (LINESIZE=) is not sufficient to display all vertical bars, then PROC CHART produces a horizontal bar chart instead.

Results: CHART Procedure

Missing Values

PROC CHART follows these rules when handling missing values:

- Missing values are not considered as valid levels for the chart variable when you use the MISSING option.
- Missing values for a GROUP= or SUBGROUP= variable are treated as valid levels.
- PROC CHART ignores missing values for the FREQ= option and the SUMVAR= option.
- If the value of the FREQ= variable is missing, zero, or negative, then the observation is excluded from the calculation of the chart statistic.
- If the value of the SUMVAR= variable is missing, then the observation is excluded from the calculation of the chart statistic.

ODS Table Names

The CHART 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 10.3 *ODS Tables Produced by the CHART Procedure*

Name	Description	Statement Used
BLOCK	A block chart	BLOCK
HBAR	A horizontal bar chart	HBAR
PIE	A pie chart	PIE
STAR	A star chart	STAR
VBAR	A vertical bar chart	VBAR

Portability of ODS Output with PROC CHART

Under certain circumstances, using PROC CHART with the Output Delivery System produces files that are not portable. If the SAS system option FORMCHAR= in your SAS session uses nonstandard line-drawing characters, then the output might include strange characters instead of lines in operating environments in which the SAS Monospace font is not installed. To avoid this problem, specify the following OPTIONS statement before executing PROC CHART:

```
options formchar="|----|+|---+=|-/\<>*";
```

Examples: CHART Procedure

Example 1: Producing a Simple Frequency Count

Features: VBAR statement

Details

This example produces a vertical bar chart that shows a frequency count for the values of the chart variable.

Program

```
data shirts;
   input Size $ @@;
   datalines;
medium    large
large     large
large     medium
```

```
medium    small
small     medium
medium    large
small     medium
large     large
large     small
medium    medium
medium    medium
medium    large
small     small
;

proc chart data=shirts;
   vbar size;

   title 'Number of Each Shirt Size Sold';
run;
```

Program Description

Create the SHIRTS data set. SHIRTS contains the sizes of a particular shirt that is sold during a week at a clothing store, with one observation for each shirt that is sold.

```
data shirts;
   input Size $ @@;
   datalines;
medium    large
large     large
large     medium
medium    small
small     medium
medium    large
small     medium
large     large
large     small
medium    medium
medium    medium
medium    large
small     small
;
```

Create a vertical bar chart with frequency counts. The VBAR statement produces a vertical bar chart for the frequency counts of the Size values.

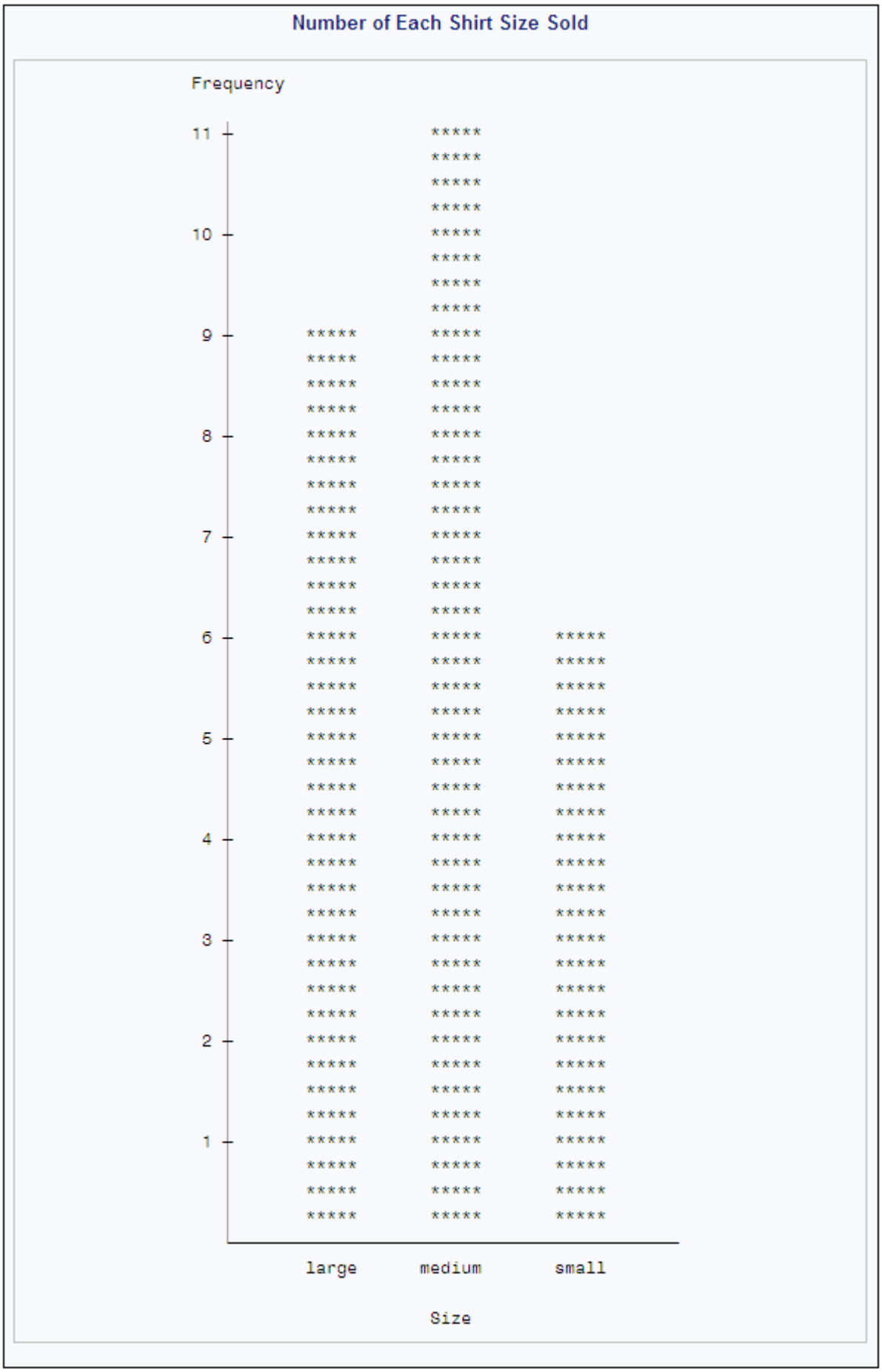
```
proc chart data=shirts;
   vbar size;
```

Specify the title.

```
   title 'Number of Each Shirt Size Sold';
run;
```

Output: HTML

The following frequency chart shows the store's sales of each shirt size for the week: 9 large shirts, 11 medium shirts, and 6 small shirts.


Number of Each Shirt Size Sold
Frequency
11
10
9
8
7
6
5
4
3
2
1
large
medium
small
Size

Example 2: Producing a Percentage Bar Chart

Features: VBAR statement option
TYPE=

Data set: SHIRTS

Details

This example produces a vertical bar chart. The chart statistic is the percentage for each category of the total number of shirts sold.

Program

```
proc chart data=shirts;
   vbar size / type=percent;

   title 'Percentage of Total Sales for Each Shirt Size';
run;
```

Program Description

Create a vertical bar chart with percentages. The VBAR statement produces a vertical bar chart. TYPE= specifies percentage as the chart statistic for the variable Size.

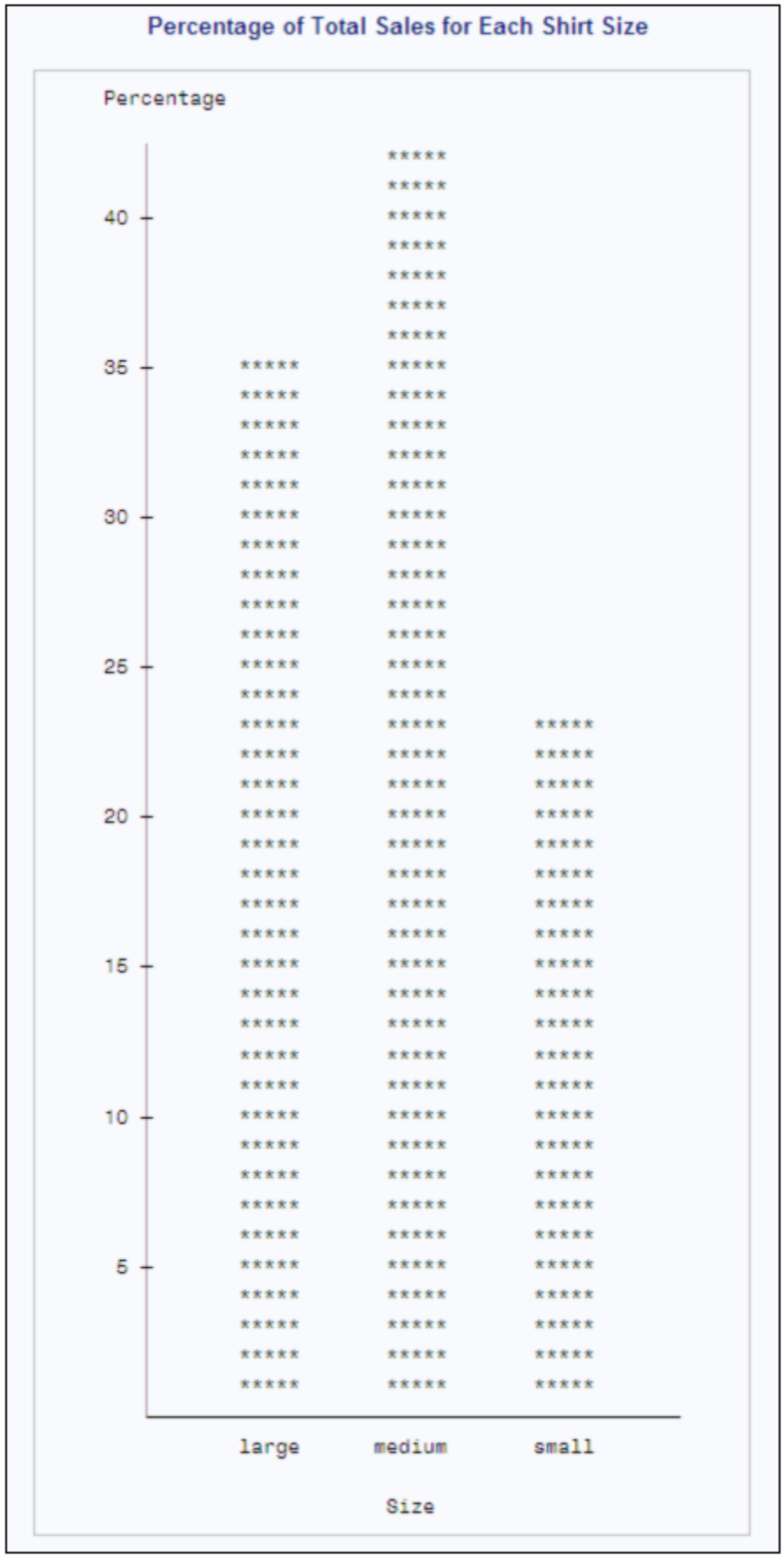
```
proc chart data=shirts;
   vbar size / type=percent;
```

Specify the title.

```
   title 'Percentage of Total Sales for Each Shirt Size';
run;
```

Output: HTML

The following chart shows the percentage of total sales for each shirt size. Of all the shirts sold, about 42.3 percent were medium, 34.6 were large, and 23.1 were small.


Percentage of Total Sales for Each Shirt Size
Percentage
40
35
30
25
20
15
10
5
large
medium
small
Size

Example 3: Subdividing the Bars into Categories

Features: VBAR statement options
SUBGROUP=
SUMVAR=

Details

This example does the following:

- produces a vertical bar chart for categories of one variable with bar lengths that represent the values of another variable
- subdivides each bar into categories based on the values of a third variable

Program

```
data piesales;
   input Bakery $ Flavor $ Year Pies_Sold;
   datalines;
Samford  apple      2005  234
Samford  apple      2006  288
Samford  blueberry  2005  103
Samford  blueberry  2006  143
Samford  cherry     2005  173
Samford  cherry     2006  195
Samford  rhubarb    2005   26
Samford  rhubarb    2006   28
Oak      apple      2005  219
Oak      apple      2006  371
Oak      blueberry  2005  174
Oak      blueberry  2006  206
Oak      cherry     2005  226
Oak      cherry     2006  311
Oak      rhubarb    2005   51
Oak      rhubarb    2006   56
Clyde    apple      2005  213
Clyde    apple      2006  415
Clyde    blueberry  2005  177
Clyde    blueberry  2006  201
Clyde    cherry     2005  230
Clyde    cherry     2006  328
Clyde    rhubarb    2005   60
Clyde    rhubarb    2006   59
;

proc chart data=piesales;
   vbar flavor / subgroup=bakery

                 sumvar=pies_sold;

   title 'Pie Sales by Flavor Subdivided by Bakery Location';
run;
```

Program Description

Create the PIESALES data set. PIESALES contains the number of each flavor of pie that is sold for two years at three bakeries that are owned by the same company. One bakery is on Samford Avenue, one on Oak Street, and one on Clyde Drive.

```
data piesales;
   input Bakery $ Flavor $ Year Pies_Sold;
   datalines;
Samford  apple      2005  234
Samford  apple      2006  288
Samford  blueberry  2005  103
Samford  blueberry  2006  143
Samford  cherry     2005  173
Samford  cherry     2006  195
Samford  rhubarb    2005   26
Samford  rhubarb    2006   28
Oak      apple      2005  219
Oak      apple      2006  371
Oak      blueberry  2005  174
Oak      blueberry  2006  206
Oak      cherry     2005  226
Oak      cherry     2006  311
Oak      rhubarb    2005   51
Oak      rhubarb    2006   56
Clyde    apple      2005  213
Clyde    apple      2006  415
Clyde    blueberry  2005  177
Clyde    blueberry  2006  201
Clyde    cherry     2005  230
Clyde    cherry     2006  328
Clyde    rhubarb    2005   60
Clyde    rhubarb    2006   59
;
```

Create a vertical bar chart with the bars that are subdivided into categories. The VBAR statement produces a vertical bar chart with one bar for each pie flavor. SUBGROUP= divides each bar into sales for each bakery.

```
proc chart data=piesales;
   vbar flavor / subgroup=bakery
```

Specify the bar length variable. SUMVAR= specifies Pies_Sold as the variable whose values are represented by the lengths of the bars.

```
                 sumvar=pies_sold;
```

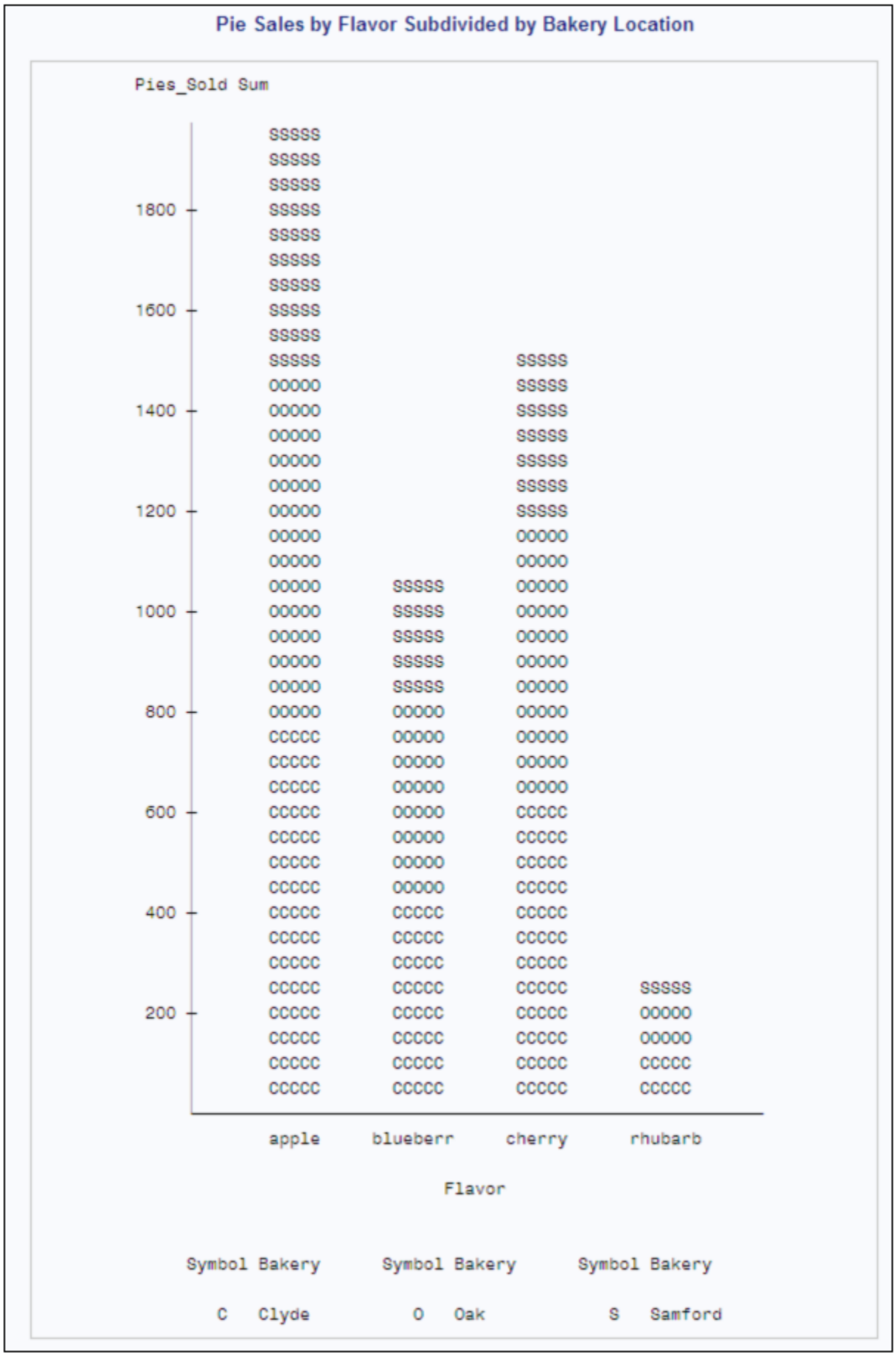
Specify the title.

```
   title 'Pie Sales by Flavor Subdivided by Bakery Location';
run;
```

Output: HTML

In the following output, the bar that represents the sales of apple pies, for example, shows 1,940 total pies across both years and all three bakeries. The symbol for the Samford Avenue bakery represents the 522 pies at the top. The symbol for the Oak

Street bakery represents the 690 pies in the middle. The symbol for the Clyde Drive bakery represents the 728 pies at the bottom of the bar for apple pies. By default, the labels along the horizontal axis are truncated to eight characters.


Pie Sales by Flavor Subdivided by Bakery Location
Pies_Sold Sum
1800
1600
1400
1200
1000
800
600
400
200
apple
blueberr
cherry
rhubarb
Flavor
Symbol Bakery
C Clyde
Symbol Bakery
O Oak
Symbol Bakery
S Samford

Example 4: Producing Side-by-Side Bar Charts

Features: VBAR statement options
GROUP=
REF=
SUMVAR=
TYPE=

Data set: PIESALES

Details

This example does the following:

- charts the mean values of a variable for the categories of another variable
- creates side-by-side bar charts for the categories of a third variable
- draws reference lines across the charts

Program

```
proc chart data=piesales;
   vbar flavor / group=bakery
                 ref=100 200 300
                 sumvar=pies_sold
                 type=mean;
   title  'Mean Yearly Pie Sales Grouped by Flavor';
   title2 'within Bakery Location';
run;
```

Program Description

Create a side-by-side vertical bar chart. The VBAR statement produces a side-by-side vertical bar chart to compare the sales across values of Bakery, specified by GROUP=. Each Bakery group contains a bar for each Flavor value.

```
proc chart data=piesales;
   vbar flavor / group=bakery
```

Create reference lines. REF= draws reference lines to mark pie sales at 100, 200, and 300.

```
                 ref=100 200 300
```

Specify the bar length variable. SUMVAR= specifies Pies_Sold as the variable that is represented by the lengths of the bars.

```
                 sumvar=pies_sold
```

Specify the statistical variable. TYPE= averages the sales for 2005 and 2006 for each combination of bakery and flavor.

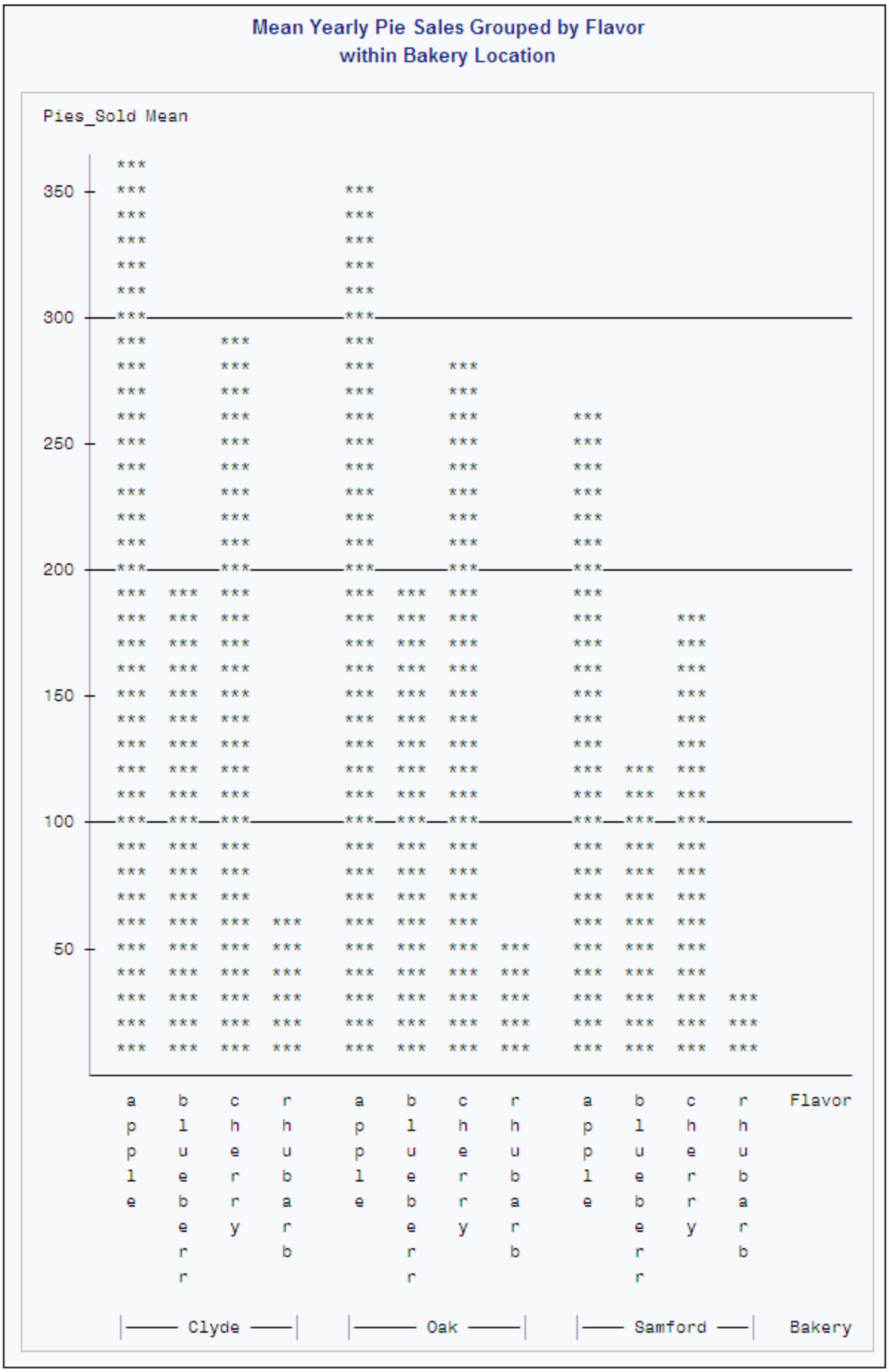
```
                  type=mean;
```

Specify the titles.

```
      title  'Mean Yearly Pie Sales Grouped by Flavor';
      title2 'within Bakery Location';
run;
```

Output: HTML

The following side-by-side bar charts compare the sales of apple pies, for example, across bakeries. The mean for the Clyde Drive bakery is 364, the mean for the Oak Street bakery is 345, and the mean for the Samford Avenue bakery is 261.


Mean Yearly Pie Sales Grouped by Flavor
within Bakery Location
Pies_Sold Mean
350
300
250
200
150
100
50
apple
blueberr
cherry
rhubarb
Flavor
Clyde
Oak
Samford
Bakery

Example 5: Producing a Horizontal Bar Chart for a Subset of the Data

Features: HBAR statement options
GROUP=
SUMVAR=

Other features: WHERE= data set option

Data set: PIESALES

Details

This example does the following:

- produces horizontal bar charts only for observations with a common value
- charts the values of a variable for the categories of another variable
- creates side-by-side bar charts for the categories of a third variable

Program

```
proc chart data=piesales(where=(year=2005));
   hbar bakery / group=flavor
   sumvar=pies_sold;
   title '2005 Pie Sales for Each Bakery According to Flavor';
run;
```

Program Description

Specify the variable value limitation for the horizontal bar chart. WHERE= limits the chart to only the 2005 sales totals.

```
proc chart data=piesales(where=(year=2005));
```

Create a side-by-side horizontal bar chart. The HBAR statement produces a side-by-side horizontal bar chart to compare sales across values of Flavor, specified by GROUP=. Each Flavor group contains a bar for each Bakery value.

```
   hbar bakery / group=flavor
```

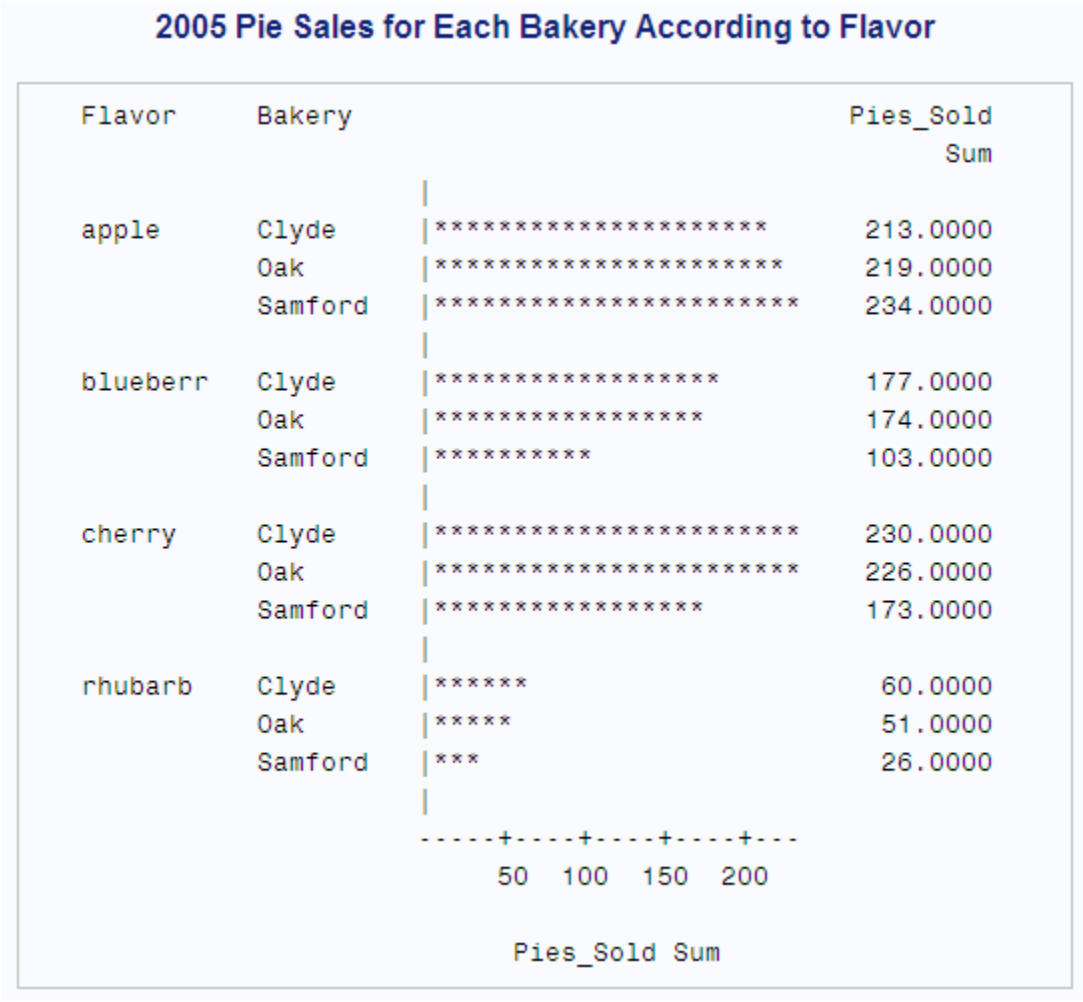
Specify the bar length variable. SUMVAR= specifies Pies_Sold as the variable whose values are represented by the lengths of the bars.

```
   sumvar=pies_sold;
```

Specify the title.

```
   title '2005 Pie Sales for Each Bakery According to Flavor';
run;
```

Output: HTML



Example 6: Producing Block Charts for BY Groups

Features: BLOCK statement options
- GROUP=
- NOHEADER=
- SUMVAR=
- SYMBOL=

BY statement

Other features: PROC SORT

SAS system options
- NOBYLINE
- OVP

TITLE statement
- #BYVAL specification

Data set: PIESALES

Details

This example does the following:

- sorts the data set
- produces a block chart for each BY group
- organizes the blocks into a three-dimensional chart
- prints BY group-specific titles

Program

```
proc sort data=piesales out=sorted_piesales;
   by year;
run;

options nobyline;

proc chart data=sorted_piesales;
   by year;

   block bakery / group=flavor

                  sumvar=pies_sold

                  noheader

                  symbol='OX';

   title  'Pie Sales for Each Bakery and Flavor';
   title2 '#byval(year)';
run;

options byline;
```

Program Description

Sort the input data set PIESALES. PROC SORT sorts PIESALES by year. Sorting is required to produce a separate chart for each year.

```
proc sort data=piesales out=sorted_piesales;
   by year;
run;
```

Suppress BY lines and allow overprinted characters in the block charts. NOBYLINE suppresses the usual BY lines in the output.

```
options nobyline;
```

Specify the BY group for multiple block charts. The BY statement produces one chart for 2005 sales and one for 2006 sales.

```
proc chart data=sorted_piesales;
   by year;
```

Create a block chart. The BLOCK statement produces a block chart for each year. Each chart contains a grid (Bakery values along the bottom, Flavor values along the side) of cells that contain the blocks.

```
   block bakery / group=flavor
```

Specify the bar length variable. SUMVAR= specifies Pies_Sold as the variable whose values are represented by the lengths of the blocks.

```
sumvar=pies_sold
```

Suppress the default header line. NOHEADER suppresses the default header line.

```
noheader
```

Specify the block symbols. SYMBOL= specifies the symbols in the blocks.

```
symbol='OX';
```

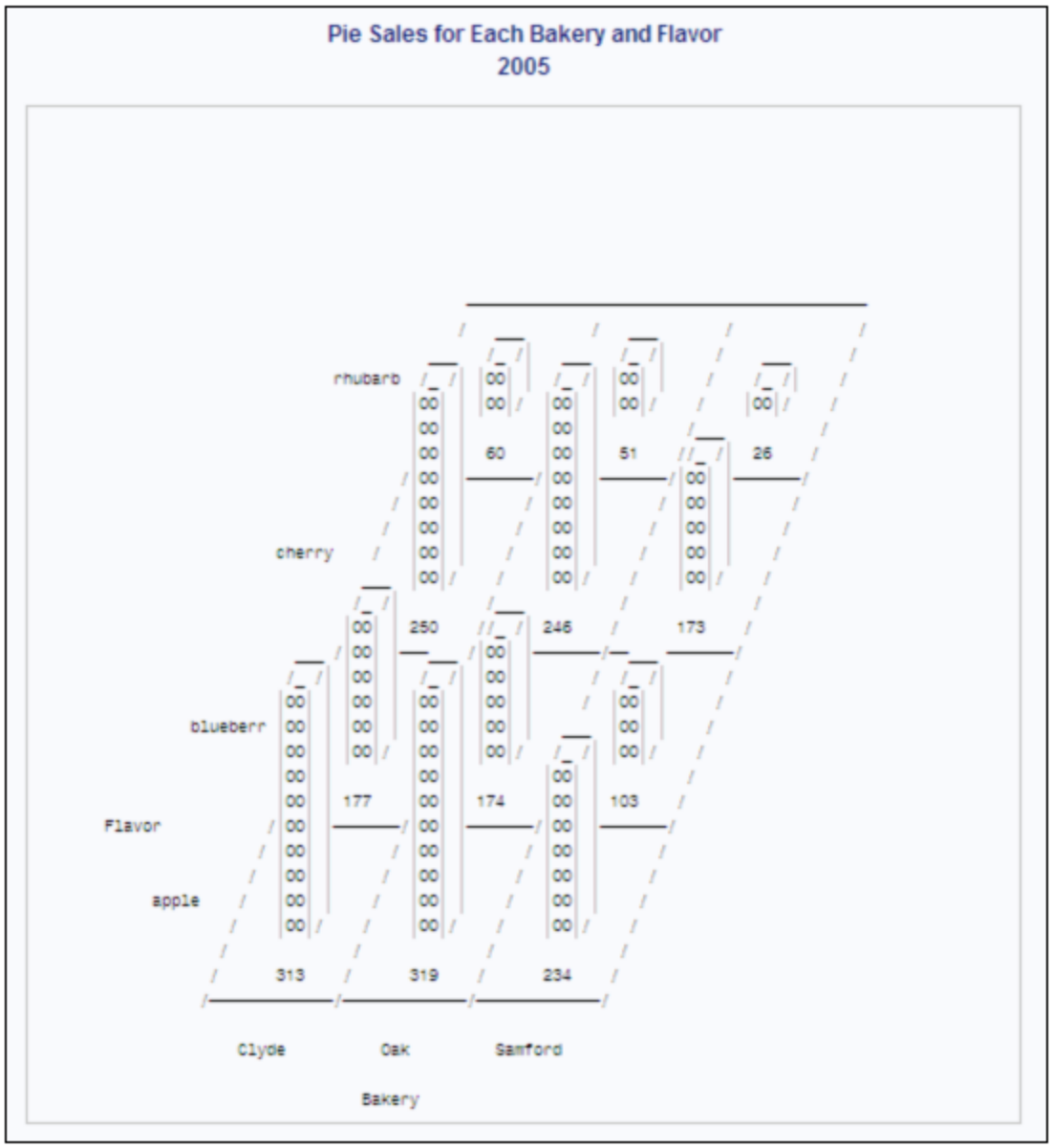
Specify the titles. The #BYVAL specification inserts the year into the second line of the title.

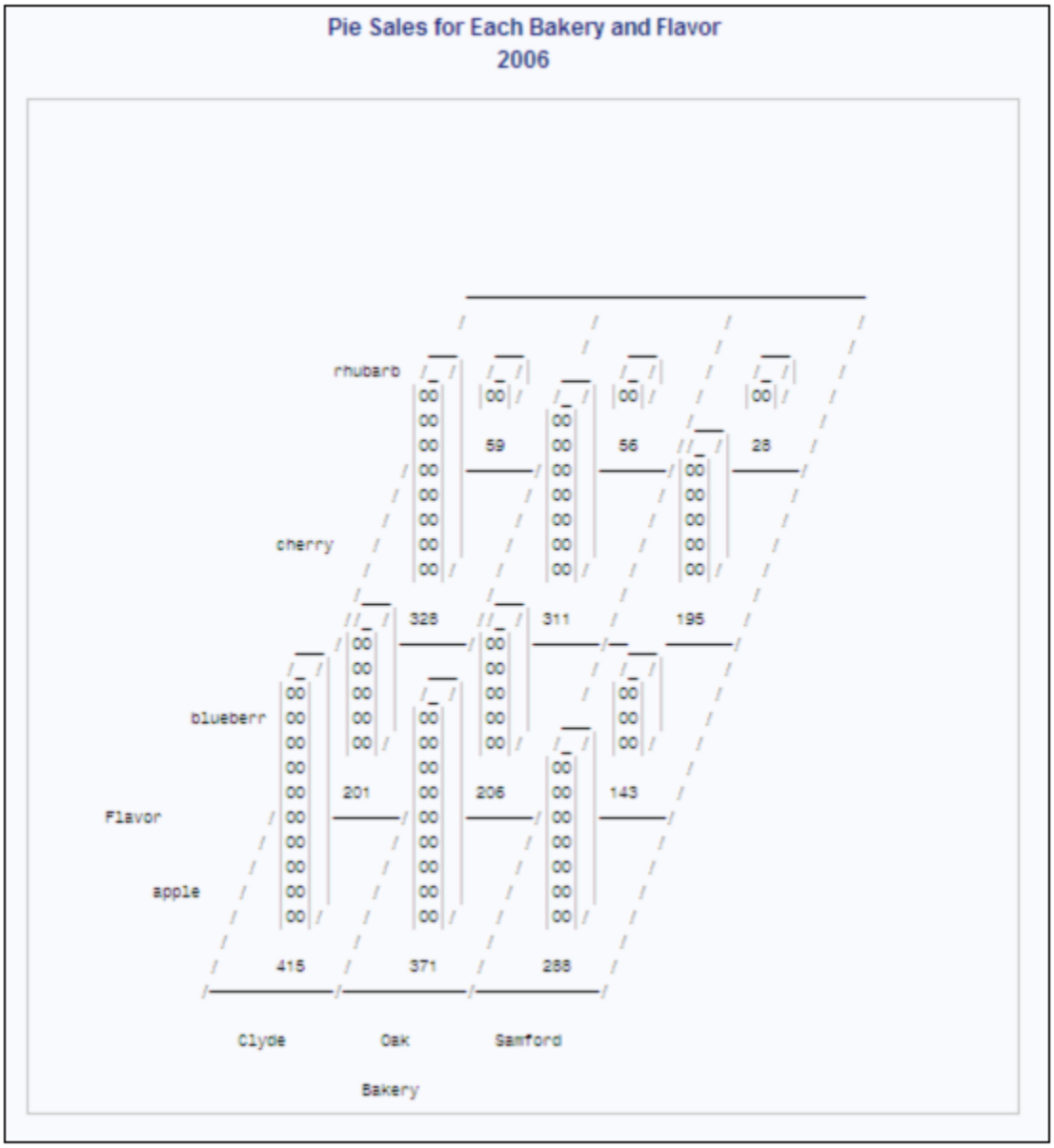
```
   title  'Pie Sales for Each Bakery and Flavor';
   title2 '#byval(year)';
run;
```

Reset the printing of the default BY line. The SAS system option BYLINE resets the printing of the default BY line.

```
options byline;
```

Output: HTML


Pie Sales for Each Bakery and Flavor
2005
rhubarb
cherry
blueberr
apple
Flavor
60
51
26
250
246
173
177
174
103
313
319
234
Clyde
Oak
Sanford
Bakery



References

Nelder, J.A. “A Simple Algorithm for Scaling Graphs.” 1976. *Applied Statistics* 25 (1), London: The Royal Statistical Society: 94–96.

Terrell, G.R. and Scott, D.W. “Oversmoothed Nonparametric Density Estimates.” 1985. *Journal of the American Statistical Association* 80 (389): 209–214.

Chapter 11

CIMPORT Procedure

Overview: CIMPORT Procedure

What Does the CIMPORT Procedure Do?

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 6 to SAS®9) or between the same versions (for example, from one SAS 9 operating environment to another SAS 9 operating environment). PROC CIMPORT automatically converts the transport file as it imports it.

However, PROC CPORT and PROC CIMPORT do not allow file transport from a later version to an earlier version, which is known as regressing (for example, from SAS® 9 to SAS 6).

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 via communications software or a magnetic medium.
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: 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: CIMPORT Procedure under Windows, UNIX, z/OS

PROC CIMPORT *destination=libref* | *<libref.>member-name <option(s)>*;
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 Statement"	Import a transport file	Ex. 1, Ex. 2, Ex. 3
"EXCLUDE Statement"	Exclude one or more specified files from the import process	
"SELECT Statement"	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 264
"Example 2: Importing Individual Catalog Entries" on page 265
"Example 3: Importing a Single Indexed SAS Data Set" on page 266

Syntax

PROC CIMPORT *destination=libref | <libref.>member-name<option(s)>*;

Summary of Optional Arguments

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.

Control the contents of the transport file

EXTENDSN=YES | NO
: specifies whether to extend by 1 byte the length of short numerics (less than 8 bytes) when you import them.

ISFILEUTF8=YES | NO ISFILEUTF8=TRUE | FALSE
: specifies whether the file is encoded in UTF-8 format.

MEMTYPE=*mtype*
: specifies that only data sets, only catalogs, or both, be moved when a library is imported.

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.

Select files to import

EET=(*etype(s)*)
: excludes specified entry types from the import process.

ET=(*etype(s)*)
: specifies entry types to import.

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.

TIP Refer to "Names in the SAS Language" in Chapter 3 of *SAS Language Reference: Concepts* for naming conventions that you can use for names and member names.

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 | CAT | C

DATA | DS | D

LIBRARY | LIB | L

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*.

Optional Arguments

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.

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.

ISFILEUTF8=YES | NO ISFILEUTF8=TRUE | FALSE

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 | Y | yes | y | TRUE | true | T | t
specifies that the data set in the transport file is encoded as UTF-8.

NO | N | no | n | FALSE | false | F | f
specifies that the data set in the transport file is not encoded as UTF-8. NO is the default.

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. **`Windows`** for example.
- SAS release. **`SAS 9.2`** 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 258

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 264

MEMTYPE=*mtype*
specifies that only data sets, only catalogs, or both, be imported from the transport file. Values for *mtype* can be as follows:

ALL	both catalogs and data sets
CATALOG | CAT	catalogs

DATA | DS SAS data sets

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.

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 355 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 "Shortcuts for Specifying Lists of Variable Names" on page 26 .

TIP Refer to "Names in the SAS Language" in Chapter 3 of *SAS Language Reference: Concepts* for naming conventions that you can use for names and member names.

Optional Arguments

ENTRYTYPE=*entry-type*
specifies a single entry type for one or more catalog entries that are listed in the EXCLUDE statement. See *SAS Language Reference: Concepts* for a complete list of catalog entry types.

Alias: 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	both catalogs and data sets
CATALOG	catalogs
DATA	SAS data sets

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.

Alias: 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 265

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 "Shortcuts for Specifying Lists of Variable Names" on page 26 .

TIP Refer to "Names in the SAS Language" in Chapter 3 of *SAS Language Reference: Concepts* for naming conventions that you can use for names and member names.

Optional Arguments

ENTRYTYPE=*entry-type*
specifies a single entry type for one or more catalog entries that are listed in the SELECT statement. See *SAS Language Reference: Concepts* for a complete list of catalog entry types.

Alias: 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.

Alias: MTYPE=, MT=

Default: ALL

Restriction: MEMTYPE= is valid only when you import a SAS library.

CIMPORT Problems: Importing Transport Files

About Transport Files and Encodings

The character data in a transport file is created in either of two types of encodings:

- the UTF-8 encoding of the SAS session in which the transport file is created

- the Windows encoding that is associated with the locale of the SAS session in which the transport file is created

These examples show how SAS applies an encoding to a transport file:

Table 11.1 *Assignment of Encodings to Transport Files*

Encoding Value of the Transport File	Example of Applying an Encoding in a SAS Invocation	Explanation
UTF-8	sas9 -encoding utf8;	A SAS session is invoked using the UTF-8 encoding. The session encoding is applied to the transport file.
wlatin2	sas9 -locale pl_PL;	A SAS session is invoked using the default UNIX 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 11.2 *Compatible Encodings*

Source SAS Session			Target SAS Session	
Locale	UNIX SAS Session Encoding	Transport File Encoding	Locale	Windows 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 Windows 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 cannot be successfully imported. Here is an example of incompatible encodings:

Table 11.3 Incompatible Encodings

Source SAS Session			Target SAS Session	
Locale	UNIX SAS Session Encoding	Transport File Encoding	Locale	z/OS Encoding
cs_CZ (Czech Czechoslovakia)	latin2	wlatin2	de_DE (German Germany)	open_ed-1141

The encodings of the source and target SAS sessions are incompatible because the Windows default encoding for the cs_CZ locale is wlatin2 and the encoding of the target SAS session is open_ed-1141. A transport file cannot be imported between these locales.

When importing transport files, you will be alerted to compatibility problems via warnings and error messages.

Problems with Transport Files Created Using a SAS Release Prior to 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 254. 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 260
- “Warning: Transport File Encoding Is Unknown” on page 261

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.

The following is an example of the UTF-8 transport file, which was created using a SAS release before 9.2, and UTF-8 target SAS session:

```
filename importin 'transport-file';
libname target 'SAS-data-library';
proc cimport isfileutf8=yes infile=importin library=target memtype=data;
run;
```

For syntax details, see the ISFILEUTF8= option on page 254.

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, which was created using a Polish Poland locale, was created in a source SAS session using a SAS release before 9.2. 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-data-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.

Problems with Transport Files Created Using SAS Versions 9.2 and Later

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 such as UTF-8 encoded transport files cannot be imported into SAS sessions that do not use the UTF-8 encoding. For example, a UTF-8 transport file cannot be imported into a SAS session that uses the Wlatin2 encoding.

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 will report a warning only, but will allow the import procedure to continue.

Here are the warning and error messages with recovery actions:

Error: Target Session Uses UTF-8: Transport File Is Not UTF-8

The error message provides this 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.

The encoding of the target SAS session cannot be UTF–8. Also, 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 national language that is used by the character data in the transport file.

   For example, if you know that Polish is the national language, specify the **pl_PL** (Polish Poland) locale in a new target SAS session. Here are the encoding values that are associated with the **pl_PL** locale:

   Table 11.4 *LOCALE= Value for the Polish Language*

Posix Locale	Windows Encoding	UNIX Encoding	z/OS Encoding
pl_PL (Polish Poland)	wlatin2	latin2	open_ed-870

   Here is an example of specifying the **pl_PL** locale in a new SAS session:

   ```
   sas9 -locale pl_PL;
   ```

   For complete details, see the Locale Table in *SAS National Language Support (NLS): Reference Guide*.

   Note: Verify that you do not have a SAS invocation command that already contains the specification of the UTF-8 encoding. **sas9 -encoding utf8;** for example. 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.

Error: Target Session Does Not Use UTF-8: Transport File Is UTF-8

The error message provides this information:

- The target session uses an identified encoding.
- The transport file is encoded as UTF-8. The encodings of the transport file and the target SAS session are incompatible.

The encoding of the target SAS session must be changed to UTF-8.

Here is an example of a SAS 9.2 UTF-8 transport file and Wlatin1 target SAS session:

1. To recover, in the target SAS session, start a new SAS session and change the session encoding to UTF-8. Here is an example:

   ```
   sas9 -encoding utf8;
   ```

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. Although the wlatin2 Windows encoding that is assigned to the transport file in the source SAS session is incompatible with the open_ed-1141 encoding of the target SAS session, a warning is displayed and the import will continue.

Table 11.5 *Encoding Values for the Czech and German Locales*

SAS Session	Posix Locale	Windows Encoding	UNIX Encoding	z/OS Encoding
Source SAS Session	cs_CZ (Czech Czechoslovakia)	wlatin2	latin2	open_ed-870
Target SAS Session	de_DE (German Germany)	wlatin1	latin9	open_ed-1141

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.

   Here is an example of specifying the **cs_CZ** locale in a new SAS session:

   ```
   sas9 -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-data-library';
   proc cimport infile=importin library=target memtype=data;
   run;
   ```

   PROC CIMPORT should succeed.

Problems with Loss of Numeric Precision

PROC CPORT and PROC CIMPORT can lose precision on numeric values that are extremely small and large. Refer to “Loss of Numeric Precision and Magnitude” in Chapter 1 of *SAS/CONNECT User's Guide* for details.

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 by means of communications software or magnetic medium. PROC CIMPORT imports the transport file to a SAS library, called NEWLIB, in the new operating environment.

Program

```
libname newlib 'SAS-data-library';
filename tranfile 'transport-file'

host-option(s)-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 CPORT created and enables you to specify any operating environment options for file characteristics.

```
libname newlib 'SAS-data-library';
filename tranfile 'transport-file'

host-option(s)-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;
```

SAS Log

```
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-data-library';
filename trans2 'transport-file'

host-option(s)-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 CPORT created and enables you to specify any operating environment options for file characteristics.

```
libname newlib 'SAS-data-library';
filename trans2 'transport-file'

host-option(s)-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;
```

SAS Log

```
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-data-library';
filename trans3 'transport-file'

host-option(s)-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-data-library';
filename trans3 'transport-file'

host-option(s)-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 CPORT exported the data set NEWDATA.TIMES in the transport file TRANS3.

```
proc cimport data=newdata.times infile=trans3;
run;
```

SAS Log

```
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
```

Chapter 12

COMPARE Procedure

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

Further, PROC COMPARE creates two kinds 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 1970 NC 85 87
1042 1971 MD 92 92
1095 1969 PA 78 72
1187 1970 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 1970 NC 84 87 Math
1042 1971 MA 92 92 History
1095 1969 PA 79 73 Physics
1187 1970 MD 87 74 Dance
1204 1971 NC 82 96 French
;
```

How Can PROC COMPARE Output Be Customized?

PROC COMPARE produces lengthy output. You can use one or more options to determine the kinds 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;
```

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
```

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.
```

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 293 shows the default output for these two data sets. "Example 1: Producing a Complete Report of the Differences" on page 303 shows the complete output for these two data sets.

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 12.1 *Comparison by the Positions of Observations*

Data Set ONE

IDNUM	NAME	GENDER	GPA
2998	Bagwell	f	3.722
9866	Metcalf	m	3.342
2118	Gray	f	3.177
3847	Baglione	f	4.000
2342	Hall	m	3.574

Data Set TWO

IDNUM	NAME	GENDER	GPA	YEAR
2998	Bagwell	f	3.722	2
9866	Metcalf	m	3.342	2
2118	Gray	f	3.177	3
3847	Baglione	f	4.000	4
2342	Hall	m	3.574	4
7565	Gold	f	3.609	2
1755	Syme	f	3.883	3

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, GENDER, and GPA.

The procedure does not report on the values of the last two observations or the variable YEAR 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 12.2 *Comparison by the Value of the ID Variable*

Data Set ONE

IDNUM	NAME	GENDER	GPA
2998	Bagwell	f	3.722
9866	Metcalf	m	3.342
2118	Gray	f	3.177
3847	Baglione	f	4.000
2342	Hall	m	3.574

Data Set TWO

IDNUM	NAME	GENDER	GPA	YEAR
2998	Bagwell	f	3.722	2
9866	Metcalf	m	3.342	2
2118	Gray	f	3.177	3
3847	Baglione	f	4.000	4
2342	Hall	m	3.574	4
7565	Gold	f	3.609	2
1755	Syme	f	3.883	3

The data sets contain three matching variables: NAME, GENDER, and GPA. They also contain five matching observations: the observations with values of `2998`, `9866`, `2118`, `3847`, and `2342` for IDNUM.

Data Set TWO contains two observations (IDNUM=`7565` and IDNUM=`1755`) for which data set ONE contains no matching observations. Similarly, no variable in data set ONE matches the variable YEAR in data set TWO.

See "Example 5: Comparing Observations with an ID Variable" on page 316 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

  $$100(ABS(y - x) / ABS(x)) > \gamma \quad for \quad x \neq 0$$

  or

  $$y \neq 0 \quad for \quad x = 0$$

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, $1-\varepsilon<1<1+\varepsilon$. 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=δγ. However, when either x or y is much larger than δ in absolute value, the comparison is like METHOD=RELATIVE CRITERION=γ. For moderate values of x and y, METHOD=RELATIVE(δ) CRITERION=γ is, in effect, a compromise between METHOD=ABSOLUTE CRITERION=δ γ and METHOD=RELATIVE CRITERION=γ.

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 = $(y - x) / x * 100 \quad for \quad x \neq 0$
- 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.

Syntax: COMPARE Procedure

Restriction: You must use the VAR statement when you use the WITH statement.

Tips: Supports the Output Delivery System. For details see Chapter 3, "Output Delivery System: Basic Concepts," in *SAS Output Delivery System: User's Guide*.

You can use the LABEL, ATTRIB, FORMAT, and WHERE statements.

PROC COMPARE <*option(s)*>;
BY <DESCENDING> *variable-1*
<…<DESCENDING>*variable-n*>
<NOTSORTED>;
ID <DESCENDING> *variable-1*
<…<DESCENDING>*variable-n*>
<NOTSORTED>;
VAR *variable(s)*;
WITH *variable(s)*;

Statement	Task	Example
"PROC COMPARE Statement"	Compare the contents of SAS data sets, or compare two variables	Ex. 1, Ex. 2, Ex. 4, Ex. 6, Ex. 7
"BY Statement"	Produce a separate comparison for each BY group	
"ID Statement"	Identify variables to use to match observations	Ex. 5
"VAR Statement"	Restrict the comparison to values of specific variables	Ex. 2, Ex. 3, Ex. 4
"WITH Statement"	Compare variables of different names	Ex. 2, Ex. 3, Ex. 4
"WITH Statement"	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.

Restrictions: If you omit COMPARE=, then you must use the WITH and VAR statements.

PROC COMPARE reports errors differently if one or both of the compared data sets are not RADIX addressable. Version 6 compressed files are not RADIX addressable, while, beginning with Version 7, compressed files are RADIX addressable. (The integrity of the data is not compromised; the procedure simply numbers the observations differently.)

Tip: You can use data set options with the BASE= and COMPARE= options.

Examples: "Example 1: Producing a Complete Report of the Differences" on page 303

"Example 2: Comparing Variables in Different Data Sets" on page 310

"Example 4: Comparing Variables That Are in the Same Data Set" on page 314

"Example 6: Comparing Values of Observations Using an Output Data Set (OUT=)" on page 322

"Example 7: Creating an Output Data Set of Statistics (OUTSTATS=)" on page 325

Syntax

PROC COMPARE <*option(s)*>;

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 print.

NODATE
: suppresses the print 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.

TRANSPOSE
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 one 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=γ
: 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 297 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.

Alias: DATA=

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 314

COMPARE=*SAS-data-set*
specifies the data set to use as the comparison data set.

Alias: 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 statement.

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.

Default: 0.00001

See: "The Equality Criterion" on page 276

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

- 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.

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.

LISTCOMPVAR
lists all variables that are found in the comparison data set but not in the base data set.

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.

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 print, where

total
is the maximum total number of differences to print. 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 print 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=.

See: "The Equality Criterion" on page 276

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 master 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 the chapter on SAS language statements in *SAS System Options: Reference*.

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 master 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 the chapter on SAS language statements in *SAS System Options: Reference*.

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 may 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 322

NOSUMMARY
suppresses the data set, variable, observation, and values comparison summary reports.

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 310

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.

Example: "Overview: COMPARE Procedure" on page 270

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 301

Example: "Example 6: Comparing Values of Observations Using an Output Data Set (OUT=)" on page 322

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 301

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 301

Example: "Example 6: Comparing Values of Observations Using an Output Data Set (OUT=)" on page 322

OUTCOMP
writes an observation to the output data set for each observation in the comparison data set, creating observations in which _TYPE_=COMP.

See: "Output Data Set (OUT=)" on page 301

Example: "Example 6: Comparing Values of Observations Using an Output Data Set (OUT=)" on page 322

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.

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 301

Example: "Example 6: Comparing Values of Observations Using an Output Data Set (OUT=)" on page 322

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.

See: "Output Data Set (OUT=)" on page 301

Example: "Example 6: Comparing Values of Observations Using an Output Data Set (OUT=)" on page 322

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 301

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 302

"Table of Summary Statistics" on page 297

Example: "Example 7: Creating an Output Data Set of Statistics (OUTSTATS=)" on page 325

PRINTALL
invokes the following options: ALLVARS, ALLOBS, ALLSTATS, LISTALL, and WARNING.

Example: "Example 1: Producing a Complete Report of the Differences" on page 303

STATS
prints a table of summary statistics for all pairs of matching numeric variables that are judged unequal.

See: "Table of Summary Statistics" on page 297 for information about the statistics produced.

TRANSPOSE
prints the reports of value differences by observation instead of by variable.

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 299.

WARNING
displays a warning message in the SAS log when differences are found.

Interaction: The ERROR option overrides the WARNING option.

BY Statement

Produces a separate comparison for each BY group.

See: "BY" on page 36

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 the observations in the data set must be sorted by all the variables that you specify. 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).

Note: The requirement for ordering observations according to the values of BY variables is suspended for BY-group processing when you use the NOTSORTED option. 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.

Details

BY Processing with PROC COMPARE

To use a BY statement with PROC COMPARE, you must sort both the base and comparison data sets by the BY variables. The nature of the comparison depends on whether all BY variables are in the comparison data set and, if they are, whether their attributes match the ones of the BY variables in the base data set. The following table shows how PROC COMPARE behaves under different circumstances:

Condition	Behavior of PROC COMPARE
All BY variables are in the comparison data set and all attributes match exactly	Compares corresponding BY groups

Condition	Behavior of PROC COMPARE
None of the BY variables are in the comparison data set	Compares each BY group in the base data set with the entire comparison data set
Some BY variables are not in the comparison data set	Writes an error message to the SAS log and terminates
Some BY variables have different types in the two data sets	Writes an error message to the SAS log and terminates

ID Statement

Lists variables to use to match observations.

See: "A Comparison with an ID Variable" on page 275

Example: "Example 5: Comparing Observations with an ID Variable" on page 316

Syntax

ID <DESCENDING> *variable-1*
<…<DESCENDING>*variable-n*>
<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 289

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.

VAR Statement

Restricts the comparison of the values of variables to the ones named in the VAR statement.

Examples: "Example 2: Comparing Variables in Different Data Sets" on page 310
"Example 3: Comparing a Variable Multiple Times" on page 312
"Example 4: Comparing Variables That Are in the Same Data Set" on page 314

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.

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 310
"Example 3: Comparing a Variable Multiple Times" on page 312
"Example 4: Comparing Variables That Are in the Same Data Set" on page 314

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.

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 12.1 *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

Bit	Condition	Code	Hexadecimal	Description
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
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;
```

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_;
   if &rc='1......'b then
```

```
      put 'Observations in Base but not
           in Comparison Data Set';
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 270. 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

The view the Data Set Summary, see Display 12.1 on page 294.

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.

Display 12.1 *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
                     WORK.TWO  Char       8          Home State
           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.

Display 12.2 *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.

Display 12.3 *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. When comparing character values, PROC COMPARE displays only the first 20 characters. When you use the TRANSPOSE option, it displays only the first 12 characters. 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.

Display 12.4 *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.

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.

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.

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":

```
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;
```

Display 12.5 *Partial Output*

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
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		

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;
```

Display 12.6 *Partial Output*

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 12.2 *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

Table Name	Description	Conditions When Table Is Generated
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

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. 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.

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 lengths of ID and VAR variables, for which it uses the longer length regardless of which data set 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, while 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 322.

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 compared (see “Table of Summary Statistics” on page 297). 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 325 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;
```

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 297.

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
```

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
                  WORK.TWO  Char       8          Home State
        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.
```

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
```

Comparing Two Data Sets: Full Report

The COMPARE Procedure
Comparison of WORK.ONE with WORK.TWO
(Method=EXACT)

Value Comparison Results for Variables

Obs	Year of Birth Base Value year	Compare Value year
1	1970	1970
2	1971	1971
3	1969	1969
4	1970	1970

Obs	Home State Base Value state	Compare Value state
1	NC	NC
2	MD	MA
3	PA	PA
4	MA	MD

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		

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

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.

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

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;
```

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 322.

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.
```

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
_____________________________________________________
```

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


_____________________________________________________
       ||       Base    Compare
 idnum ||     salary     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
OUTBASE
OUTCOMP
OUTDIF
OUTNOEQUAL

Other features: 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 316, 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 output data set RESULT 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 output data set RESULT 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;
```

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.

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.XXXXXXXXXXXXX	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

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.XXXXXXXXXX.....	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 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;
```

The variables are described in "Output Statistics Data Set (OUTSTATS=)" on page 302.

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	.	.

Chapter 13

CONTENTS Procedure

Overview: CONTENTS Procedure

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

See Concepts for the CONTENTS Statement on page 371.

Syntax: CONTENTS Procedure

Notes: The links in the table below are to the DATASETS procedure documentation, which explains these options.

You cannot use the WHERE option to affect the output because PROC CONTENTS does not process any observations.

Tips: You can use the ATTRIB, FORMAT, and LABEL statements.

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.

Complete documentation the CONTENTS procedure is in CONTENTS Statement on page 399.

See: CONTENTS Procedure under Windows, UNIX, z/OS

PROC CONTENTS <*option-1* <*...option-n*>>;

Statement	Task	Example
"CONTENTS Statement"	List the contents of one or more SAS data sets and print the directory of the SAS library	Ex. 3
"CENTILES"	Print centiles information for indexed variables	
"DATA=SAS-file-specification"	Specify the input data set	Ex. 3
"DETAILS | NODETAILS"	Include information in the output about the number of observations, number of variables, number of indexes, and data set labels	
"DIRECTORY"	Print a list of the SAS files in the SAS library	
"FMTLEN"	Print the length of a variable's informat or format	
"MEMTYPE=(mtype-1 <...mtype-n>)"	Restrict processing to one or more types of SAS files	
"NODS"	Suppress the printing of individual files	
"NOPRINT"	Suppress the printing of the output	
"ORDER= COLLATE | CASECOLLATE | IGNORECASE | VARNUM"	Print a list of variables in various order. Note that the ORDER option does not affect the order of the OUT= or OUT2= data sets	Ex. 2
"OUT=SAS-data-set"	Specify the name for an output data set	
"OUT2=SAS-data-set "	Specify the name of an output data set to contain information about indexes and integrity constraints	
"SHORT"	Print abbreviated output	
"VARNUM"	Print a list of the variables by their position in the data set. By default, the CONTENTS statement lists the variables alphabetically	Ex. 2

Examples: CONTENTS Procedure

Example 1: Using PROC CONTENTS to Extract Only Attributes from Data Sets

Features: CONTENTS Procedure
PRINT Procedure

Details

Using SAS Output Delivery System (ODS), you can extract the attributes from a data set without listing the variables. There are different destinations that can be used other than RTF. The code creates a data set that contains all the attributes information from each of the data sets in the library specified, since we have used the _ALL_ keyword. For more information, see *SAS Output Delivery System: User's Guide*.

Program

```
options ls=79 nodate nocenter;
title;

data work.one;x=1;
run;
data work.two;x=2;
run;

ods listing close;
ods output attributes=attout;


proc contents data=work._all_;
run;


ods rtf file='cinfo.rtf';


proc print data=attout;
run;

ods rtf close;
ods listing;
```

Program Description

```
options ls=79 nodate nocenter;
title;
```

```
data work.one;x=1;
run;
data work.two;x=2;
run;

ods listing close;
```

Close the listing destination.

```
ods output attributes=attout;
```

Specify the attributes output object.

```
proc contents data=work._all_;
run;
```

Get the attributes for all data sets in the WORK library.

```
ods rtf file='cinfo.rtf';
```

Create RTF.

```
proc print data=attout;
run;

ods rtf close;
ods listing;
```

Results

Output 13.1 *PROC CONTENTS ODS Output*

1

Obs	Member	Label1	cValue1	nValue1	Label2	cValue2	nValue2
1	WORK.ONE	Data Set Name	WORK.ONE	.	Observations	1	1.000000
2	WORK.ONE	Member Type	DATA	.	Variables	1	1.000000
3	WORK.ONE	Engine	V9	.	Indexes	0	0
4	WORK.ONE	Created	Monday, December 13, 2010 08:52:15 AM	1607849535	Observation Length	8	8.000000
5	WORK.ONE	Last Modified	Monday, December 13, 2010 08:52:15 AM	1607849535	Deleted Observations	0	0
6	WORK.ONE	Protection		.	Compressed	NO	.
7	WORK.ONE	Data Set Type		.	Sorted	NO	.
8	WORK.ONE	Label		.			0
9	WORK.ONE	Data Representation	WINDOWS_32	.			0
10	WORK.ONE	Encoding	wlatin1 Western (Windows)	.			0
11	WORK.TWO	Data Set Name	WORK.TWO	.	Observations	1	1.000000
12	WORK.TWO	Member Type	DATA	.	Variables	1	1.000000
13	WORK.TWO	Engine	V9	.	Indexes	0	0
14	WORK.TWO	Created	Monday, December 13, 2010 08:52:24 AM	1607849545	Observation Length	8	8.000000
15	WORK.TWO	Last Modified	Monday, December 13, 2010 08:52:24 AM	1607849545	Deleted Observations	0	0
16	WORK.TWO	Protection		.	Compressed	NO	.
17	WORK.TWO	Data Set Type		.	Sorted	NO	.
18	WORK.TWO	Label		.			0
19	WORK.TWO	Data Representation	WINDOWS_32	.			0
20	WORK.TWO	Encoding	wlatin1 Western (Windows)	.			0

Example 2: Using the ORDER= Option with the CONTENTS Statement

Features: CONTENTS statement options:

- ORDER=
- COLLATE
- CASECOLLATE
- IGNORECASE
- VARNUM

Details

The ORDER= options prints a list of variables in a specified order.

Program

```
options nonotes nodate nonumber nocenter formdlim  ='-';

libname contents 'c:\proccontents';

data contents.test;
   d=2;
```

```
   b001 =1;
   b002 =2;
   b003 =3;
   b001z=1;
   B001a=2;
   CaSeSeNsItIvE2=9;
   CASESENSITIVE3=9;
   D=2;
   casesensitive1=9;
   CaSeSeNsItIvE1a=9;
   d001z=1;
   CASESENSITIVE1C=9;
   D001a=2;
   casesensitive1b=9;
   A =1;
   a002 =2;
   a =3;
   a001z=1;
   A001a=2;

run;

%let mydata=contents.test;

ods output Variables=var1(keep=Num Variable);
ods listing close;

proc contents data=&mydata;
run;

ods listing;
    title "Default options";

proc print data=var1 noobs;
run;

ods output Variables=var2(keep=Num Variable);
ods listing close;

proc contents order=collate data=&mydata;
run;

ods listing;
    title "order=collate option";

proc print data=var2 noobs;
run;

ods output Variables=var3(keep=Num Variable);
ods listing close;

proc contents order=casecollate data=&mydata;
run;

ods listing;
    title "order=casecollate option";
```

```
proc print data=var3 noobs;
run;

ods output Variables=var4(keep=Num Variable);
ods listing close;

proc contents order=ignorecase data=&mydata;
run;

ods listing;
    title "order=ignorecase option";

proc print data=var4 noobs;
run;


ods output Position=var5(keep=Num Variable);
ods listing close;

proc contents data=&mydata varnum;
run;

ods listing;
    title "varnum option";

proc print data=var5 noobs;
run;
```

Program Description

Set up the data set.

```
options nonotes nodate nonumber nocenter formdlim  ='-';

libname contents 'c:\proccontents';

data contents.test;
   d=2;
   b001 =1;
   b002 =2;
   b003 =3;
   b001z=1;
   B001a=2;
   CaSeSeNsItIvE2=9;
   CASESENSITIVE3=9;
   D=2;
   casesensitive1=9;
   CaSeSeNsItIvE1a=9;
   d001z=1;
   CASESENSITIVE1C=9;
   D001a=2;
   casesensitive1b=9;
   A =1;
   a002 =2;
   a =3;
```

```
   a001z=1;
   A001a=2;

run;
```

To produce PROC CONTENTS output for a data set of your choice, change data set name to MYDATA.

```
%let mydata=contents.test;

ods output Variables=var1(keep=Num Variable);
ods listing close;

proc contents data=&mydata;
run;

ods listing;
    title "Default options";

proc print data=var1 noobs;
run;

ods output Variables=var2(keep=Num Variable);
ods listing close;

proc contents order=collate data=&mydata;
run;

ods listing;
    title "order=collate option";

proc print data=var2 noobs;
run;

ods output Variables=var3(keep=Num Variable);
ods listing close;

proc contents order=casecollate data=&mydata;
run;

ods listing;
    title "order=casecollate option";

proc print data=var3 noobs;
run;

ods output Variables=var4(keep=Num Variable);
ods listing close;

proc contents order=ignorecase data=&mydata;
run;

ods listing;
    title "order=ignorecase option";
```

```
proc print data=var4 noobs;
run;
```

The name of the ODS output object is different when the varnum option is used.

```
ods output Position=var5(keep=Num Variable);
ods listing close;

proc contents data=&mydata varnum;
run;

ods listing;
    title "varnum option";

proc print data=var5 noobs;
run;
```

Using the COLLATE and CASECOLLATE Options

The following output shows the results of the ORDER= default, the COLLATE option, and the CASECOLLATE option.

Default options

Num	Variable
15	A
18	A001a
6	B001a
8	CASESENSITIVE3
12	CASESENSITIVE1C
7	CaSeSeNsItIvE2
10	CaSeSeNsItIvE1a
13	D001a
16	a002
17	a001z
2	b001
3	b002
4	b003
5	b001z
9	casesensitive1
14	casesensitive1b
1	d
11	d001z

order=collate option

Num	Variable
15	A
18	A001a
6	B001a
12	CASESENSITIVE1C
8	CASESENSITIVE3
10	CaSeSeNsItIvE1a
7	CaSeSeNsItIvE2
13	D001a
17	a001z
16	a002
2	b001
5	b001z
3	b002
4	b003
9	casesensitive1
14	casesensitive1b
1	d
11	d001z

order=casecollate option

Num	Variable
15	A
18	A001a
17	a001z
16	a002
2	b001
6	B001a
5	b001z
3	b002
4	b003
9	casesensitive1
10	CaSeSeNsItIvE1a
14	casesensitive1b
12	CASESENSITIVE1C
7	CaSeSeNsItIvE2
8	CASESENSITIVE3
1	d
13	D001a
11	d001z

The following output shows the results of the ORDER= default, IGNORECASE option, and VARNUM option.

Output 13.2 *Results of Using the IGNORECASE and VARNUM Options*

Default options

Num	Variable
15	A
18	A001a
6	B001a
8	CASESENSITIVE3
12	CASESENSITIVE1C
7	CaSeSeNsItIvE2
10	CaSeSeNsItIvE1a
13	D001a
16	a002
17	a001z
2	b001
3	b002
4	b003
5	b001z
9	casesensitive1
14	casesensitive1b
1	d
11	d001z

order=ignorecase option

Num	Variable
15	A
16	a002
18	A001a
17	a001z
2	b001
3	b002
4	b003
6	B001a
5	b001z
9	casesensitive1
7	CaSeSeNsItIvE2
8	CASESENSITIVE3
10	CaSeSeNsItIvE1a
14	casesensitive1b
12	CASESENSITIVE1C
1	d
13	D001a
11	d001z

varnum option

Num	Variable
1	d
2	b001
3	b002
4	b003
5	b001z
6	B001a
7	CaSeSeNsItIvE2
8	CASESENSITIVE3
9	casesensitive1
10	CaSeSeNsItIvE1a
11	d001z
12	CASESENSITIVE1C
13	D001a
14	casesensitive1b
15	A
16	a002
17	a001z
18	A001a

Example 3: Describing a SAS Data Set

Features: CONTENTS statement option
DATA=

Other features: SAS data set option
READ=

This example shows 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 468.

```
options pagesize=40 linesize=80 nodate pageno=1;

LIBNAME health
'SAS-library';
```

Specify HEALTH as the procedure input library, and suppress the directory listing.

```
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" on page 447.

```
   contents data=group (read=green) out=grpout;
   title  'The Contents of the GROUP Data Set';
run;
```

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	Tuesday, March 22, 2005 09:00:23 AM	Observation Length	96
Last Modified	Tuesday, March 22, 2005 09:00:23 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	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
Filename	c:\procdatasets\group.sas7bdat
Release Created	9.0201B0
Host Created	XP_PRO

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		

Chapter 14

COPY Procedure

Overview: COPY Procedure

The COPY procedure copies one or more SAS files from a SAS library.

Generally, the COPY procedure functions the same as the COPY statement in the DATASETS procedure. The two 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.

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 842.

Concepts

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 343.

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 information, see Chapter 15, "CPORT Procedure," on page 349 and Chapter 11, "CIMPORT Procedure," on page 251.

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`**.

See the Details on page 410 section of the CONTENT statement in PROC DATASETS for more information.

Syntax: COPY Procedure

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.

Tips: See "Statements with the Same Function in Multiple Procedures" for details. You can also use any global statements. See "Global Statements" for a list.

Complete documentation for the COPY procedure is in COPY Statement on page 405.

PROC COPY OUT=*libref-1* IN=*libref-2*
<CLONE |NOCLONE>
<CONSTRAINT=YES |NO>
<DATECOPY>
<INDEX=YES | NO>
<MEMTYPE=(*mtype-1* <...*mtype-n*>)>
<MOVE <ALTER=*alter-password*>>;
<OVERRIDE *ds_option-1=value-1*<...*ds_option-n=value-n*>>

EXCLUDE *SAS-file-1* <...*SAS-file-n*> </ MEMTYPE=*mtype*>;
SELECT *SAS-file-1* <...*SAS-file-n*> </ <MEMTYPE=*mtype*><ALTER=*alter-password*>>;

Statement	Task	Example
"COPY Statement"	Copy one or more files	Ex. 1
"EXCLUDE Statement"	Exclude files or memtypes	
"CLONE | NOCLONE"	Specify whether to copy data set attributes	Ex. 2, Ex. 3
"IN=libref-2"	Name of source library	Ex. 1
"OUT=libref-1"	Name of destination library	Ex. 1, Ex. 3
"SELECT Statement"	Select files or memtypes	Ex. 1, Ex. 2
"OVERRIDE=(ds_option-1=value-1 <...ds_option-n=value-n>)"	Overrides specified output data set options copied from the input data set	

Examples: COPY Procedure

Example 1: Copying SAS Data Sets between Hosts

Features: PROC COPY statement options:
IN=
MEMTYPE=
OUT=

Other features: XPORT engine

Details

This example illustrates 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;
```

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;
```

```
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
```

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 (for example, by using FTP in binary mode to the Windows host), 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;
```

```
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:
- NOCLONE
- IN=
- OUT=

Other features: CVP engine

Details

This example illustrates 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 about the "Compatible and Incompatible Encodings," see *SAS National Language Support (NLS): Reference Guide*.

```
LIBNAME inlib cvp 'SAS-library';
LIBNAME outlib 'SAS-library' outencoding="encoding value for output";
proc copy noclone in=inlib out=outlib;
   select car;
run;
```

SAS 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: Using PROC COPY to Migrate from a 32-bit to a 64-bit Machine

Features: PROC COPY statement options:
- IN=
- OUT=
- NOCLONE
- SELECT

Other features: OUTREP=

Details

Use PROC COPY to migrate from a 32-bit to a 64-bit environment. PROC MIGRATE does not support item stores when you migrate from a 32-bit to a 64-bit environment.

Program

```
libname source 'path\SAS-data-library';
libname target 'path\SAS-data-library'
    outrep=windows_64;


proc copy in=source out=target NOCLONE;
select dsname;
run;
```

Program Description

Assign library resources. Use the OUTREP= option when changing from a 32-bit to a 64-bit machine.

```
libname source 'path\SAS-data-library';
libname target 'path\SAS-data-library'
    outrep=windows_64;
```

Copy data set from a 32-bit to a 64-bit machine.

```
proc copy in=source out=target NOCLONE;
select dsname;
run;
```

Chapter 15

CPORT Procedure

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 to the target computer via communications software or a magnetic medium.
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: CPORT Procedure under Windows, UNIX , z/OS

PROC CPORT *source-type=libref* | <*libref.*>*member-name*<*option(s)*>;
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*
<*option(s)*>;

Statement	Task	Example
"PROC CPORT Statement"	Create a transport file	Ex. 1, Ex. 2, Ex. 3, Ex. 4, Ex. 5
"EXCLUDE Statement"	Exclude one or more specified files from the transport file	
"SELECT Statement"	Specify one or more files or entries to include in the transport file	Ex. 2
"TRANTAB Statement"	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 360
"Example 2: Exporting Individual Catalog Entries" on page 361
"Example 3: Exporting a Single SAS Data Set" on page 362
"Example 4: Applying a Translation Table" on page 363
"Example 5: Exporting Entries Based on Modification Date" on page 364

Syntax

PROC CPORT *source-type=libref* | <*libref.*>*member-name*<*option(s)*>;

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=*DBCS-type*
: 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=*mtype*
: 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.

TIP Refer to "Names in the SAS Language" in Chapter 3 of *SAS Language Reference: Concepts* for naming conventions that you can use for names and member names.

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 | CAT | C

DATA | DS | D

LIBRARY | LIB | L

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 359.

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.

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 364

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 the section on SAS files in *SAS Language Reference: Concepts*.

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 when the SAS file was created and the date and time when 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.

Tip: 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 429.

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 about generation data sets, see *SAS Language Reference: Concepts*.

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=*DBCS-type*
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 | HITAC | FACOM	for z/OS
IBM	for VSE
DEC | SJIS	for OpenVMS
PCIBM | SJIS	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=*mtype*
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 | CAT
catalogs

DATA | DS
SAS data sets

Alias: MT=

Default: ALL

Example: "Example 1: Exporting Multiple Catalogs" on page 360

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 Argument

SAS file(s) | catalog entry(s)
specifies one or more SAS files or one or more catalog entries 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 "Shortcuts for Specifying Lists of Variable Names" on page 26.

TIP Refer to "Names in the SAS Language" in Chapter 3 of *SAS Language Reference: Concepts* for naming conventions that you can use for names and member names.

Optional Arguments

ENTRYTYPE=*entry-type*
specifies a single entry type for the catalog entries listed in the EXCLUDE statement. See *SAS Language Reference: Concepts* for a complete list of catalog entry types.

Alias: 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 or CAT, 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 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 CPORT statement:

Alias: 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.

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.

Example: "Example 2: Exporting Individual Catalog Entries" on page 361

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 "Shortcuts for Specifying Lists of Variable Names" on page 26.

TIP Refer to "Names in the SAS Language" in Chapter 3 of *SAS Language Reference: Concepts* for naming conventions that you can use for names and member names.

Optional Arguments

ENTRYTYPE=*entry-type*
specifies a single entry type for the catalog entries listed in the SELECT statement. See *SAS Language Reference: Concepts* for a complete list of catalog entry types.

Alias: 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 or CAT, 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 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 CPORT statement.

Alias: 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.

TRANTAB Statement

Specifies translation tables for characters in catalog entries that you export.

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 363

Syntax

TRANTAB NAME=*translation-table-name*
<*option(s)*>;

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 will be 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 42, "PWENCODE Procedure," on page 1133.

Example 2: Encoded Password

```
proc cport
data=source.grades(read={sas003}6EDB396015B96DBD9E80F0913A543819A8E5)
   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 on 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 Language Reference: Concepts*.

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

Data Control Blocks Characteristics

A common problem when you create or import a transport file under the z/OS environment is a failure to specify the correct Data Control Block (DCB) characteristics. When you reference a transport file, you must specify the following DCB characteristics:

- LRECL=80
- BLKSIZE=8000
- RECFM=FB
- DSORG=PS

Another common problem can occur if you use communications software to move files from another environment to z/OS. In some cases, the transport file does not have the proper DCB characteristics when it arrives on z/OS. If the communications software does not allow you to specify file characteristics, try the following approach for z/OS:

1. Create a file under z/OS with the correct DCB characteristics and initialize the file.
2. Move the transport file from the other environment to the newly created file under z/OS using binary transfer.

Loss of Numeric Precision

PROC CPORT and PROC CIMPORT can lose precision on numeric values that are extremely small and large. Refer to “Loss of Numeric Precision and Magnitude” in Chapter 1 of *SAS/CONNECT User's Guide* for details.

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-data-library';
filename tranfile 'transport-file'

host-option(s)-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-data-library';
filename tranfile 'transport-file'

host-option(s)-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;
```

SAS Log

```
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-data-library';
filename tranfile 'transport-file'

host-option(s)-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-data-library';
filename tranfile 'transport-file'

host-option(s)-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;
```

SAS Log

```
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-data-library';
filename tranfile 'transport-file'
```

```
host-option(s)-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-data-library';
filename tranfile 'transport-file'

host-option(s)-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;
```

SAS Log

```
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-data-library';
filename tranfile 'transport-file'

host-option(s)-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-data-library';
filename tranfile 'transport-file'

host-option(s)-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;
```

SAS Log

```
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-data-library';
filename tranfile 'transport-file'

host-option(s)-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-data-library';
filename tranfile 'transport-file'

host-option(s)-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;
```

SAS Log

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.

```
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.
```

Chapter 16

DATASETS Procedure

Overview: DATASETS Procedure

What Does the DATASETS Procedure Do?

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, such as:
  - the date when 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

Sample PROC DATASETS Output

The following 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;
```

Log 16.1 *Log from PROC DATASETS*

```
744  proc datasets library=health memtype=data details;
                                Directory

Libref         HEALTH
Engine         V9
Physical Name  c:\Documents and Settings\myfile\My Documents\procdatasets\health
Filename       c:\Documents and Settings\myfile\My Documents\procdatasets\health


             Member  Obs, Entries                                  File
 #  Name     Type     or Indexes   Vars  Label                     Size  Last Modified

 1  ALL      DATA         23        17                            13312  12Sep07:10:57:50
 2  BODYFAT  DATA         83        13   California Results       13312  12Sep07:10:57:54
 3  CONFOUND DATA          8         4                             5120  12Sep07:10:57:50
 4  CORONARY DATA         39         4                             5120  12Sep07:10:57:50
 5  DRUG1    DATA          6         2   JAN2005 DATA              5120  12Sep07:10:57:50
 6  DRUG2    DATA         13         2   MAY2005 DATA              5120  12Sep07:10:57:50
 7  DRUG3    DATA         11         2   JUL2005 DATA              5120  12Sep07:10:57:50
 8  DRUG4    DATA          7         2   JAN2002 DATA              5120  12Sep07:10:57:50
 9  DRUG5    DATA          1         2   JUL2002 DATA              5120  12Sep07:10:57:50
10  GROUP    DATA        148        11   Test Subjects            33792  12Sep07:13:01:16
    GROUP    INDEX         1                                      9216  12Sep07:13:01:16
11  GRPOUT   DATA         11        40                            17408  13Dec10:11:40:24
12  GRPOUT1  DATA         11        40                            17408  13Dec10:10:28:32
13  INFANT   DATA        149         6                            17408  12Sep07:10:57:52
14  MLSCL    DATA         32         4   Multiple Sclerosis Data   5120  12Sep07:10:57:52
15  NAMES    DATA          7         4                             5120  12Sep07:10:57:52
16  OXYGEN   DATA         31         7                            17408  12Sep07:13:01:16
17  PERSONL  DATA        148        11                            25600  12Sep07:10:57:52
18  PHARM    DATA          6         3   Sugar Study               5120  12Sep07:10:57:52
19  POINTS   DATA          6         6                             5120  12Sep07:10:57:52
20  RESULTS  DATA         10         5                             5120  12Sep07:10:57:54
21  SLEEP    DATA        108         6                             9216  12Sep07:10:57:54
22  TEST2    DATA         15         5                             5120  12Sep07:10:57:54
23  TRAIN    DATA          7         2                             5120  12Sep07:10:57:54
24  VISION   DATA         16         3                             5120  12Sep07:10:57:54
25  WEIGHT   DATA          1         2                             5120  12Sep07:10:57:50
26  WGHT     DATA         83        13                            13312  12Sep07:10:57:54
745     delete syndrome;
746     change prenat=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;
```

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*. If you are unfamiliar with SAS files and SAS libraries, refer to “SAS Files Concepts” in Chapter 1 of *SAS Language Reference: Concepts*.
- PROC DATASETS cannot work with sequential data libraries.
- You cannot change the length of a variable using the LENGTH statement or the LENGTH= option on 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.

Concepts

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. For example, 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, the statements that perform those tasks must appear in that order (that is, CONTENTS, COPY, CONTENTS).

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 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 that 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.

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, refer to "Assigning Passwords" in Chapter 34 of *SAS Language Reference: Concepts*.

When you are working with password-protected SAS files in the AGE, CHANGE, DELETE, EXCHANGE, REPAIR, or SELECT statement, you can specify 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. Refer to COPY statement on page 405 and the MODIFY statement on page 429.

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 only refers 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.

[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.

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;
   ```

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 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. Refer to CONTENTS statement on page 399, COPY Statement on page 405, and MODIFY statement on page 429 for more information.

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. (Refer to Table 16.1 on page 376 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 412.

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

MDDB
: 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 16.1 *Subordinate Statements and Appropriate Member Types*

Statement	Appropriate Member Types	Default Member Type
AGE	ACCESS, CATALOG, DATA, FDB, MDDB, PROGRAM, VIEW	DATA
CHANGE	ACCESS, ALL, CATALOG, DATA, FDB, MDDB, PROGRAM, VIEW	ALL
CONTENTS	ALL, DATA, VIEW	DATA *
COPY	ACCESS, ALL, CATALOG, DATA, FDB, MDDB, PROGRAM, VIEW	ALL
DELETE	ACCESS, ALL, CATALOG, DATA, FDB, MDDB, PROGRAM, VIEW	DATA
EXCHANGE	ACCESS, ALL, CATALOG, DATA, FDB, MDDB, PROGRAM, VIEW	ALL
EXCLUDE	ACCESS, ALL, CATALOG, DATA, FDB, MDDB, PROGRAM, VIEW	ALL
MODIFY	ACCESS, DATA, VIEW	DATA
REPAIR	ALL, CATALOG, DATA	ALL **
SAVE	ACCESS, ALL, CATALOG, DATA, FDB, MDDB, PROGRAM, VIEW	ALL
SELECT	ACCESS, ALL, CATALOG, DATA, FDB, MDDB, PROGRAM, VIEW	ALL

* When DATA=_ALL_ in the CONTENTS statement, the default is ALL. ALL includes only DATA and VIEW.

** ALL includes only DATA and CATALOG.

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, refer to "Generation Data Sets" in "Understanding Generation Data Sets" in Chapter 26 of *SAS Language Reference: Concepts*.

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 429.

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.

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 only refers 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, 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);
   ```

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.

Syntax: DATASETS Procedure

Tips: Supports RUN-group processing.

Supports the Output Delivery System. For details, see Chapter 3, “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 35. You can also use any global statement. For a list, see “Global Statements” in Chapter 1 of *SAS Statements: Reference*.

DATASETS Procedure under Windows, UNIX, z/OS

```
PROC DATASETS <<option-1 <...option-n>>>;
  AGE current-name related-SAS-file-1 <...current-name related-SAS-file-n>
    </ <ALTER=alter-password><MEMTYPE=mtype>>;
  APPEND BASE=<libref.>SAS-data-set
    <APPENDVER=V6>
    <DATA=<libref.>SAS-data-set>
    <FORCE>
    <GETSORT>
    <NOWARN>;
  AUDIT SAS-file <(SAS-password)<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-1 <...variable-n>>;
  CHANGE old-name-1=new-name-1
    <…old-name-n=new-name-n>
    </ <ALTER=alter-password><GENNUM=ALL | integer><MEMTYPE=mtype>>;
  CONTENTS <option-1 <...option-n>>;
  COPY OUT=libref-1
    <CLONE | NOCLONE>
    <CONSTRAINT=YES | NO>
    <DATECOPY>
    <FORCE>
    <IN=libref-2>
    <INDEX=YES | NO>
    <MEMTYPE=(mtype-1 <...mtype-n>)>
    <MOVE <ALTER=alter-password>>
    <OVERRIDE ds_option-1=value-1<...ds_option-n=value-n>>;
      EXCLUDE SAS-file-1 <...SAS-file-n> < / MEMTYPE=mtype>;
      SELECT SAS-file-1 <...SAS-file-n>
        </ <ALTER=alter-password>
        <MEMTYPE= mtype>>;
  DELETE SAS-file-1 <...SAS-file-n>
    </ <ALTER=alter-password><GENNUM=ALL | HIST | REVERT | integer>
    <MEMTYPE=mtype>>;
  EXCHANGE name-1=other-name-1
    <…name-n=other-name-n>
    </ <ALTER=alter-password><MEMTYPE=mtype>>;
```

MODIFY *SAS-file* <(*option-1* <...*option-n*>)>
</ <CORRECTENCODING=*encoding-value*><DTC=*SAS-date-time*>
<GENNUM=*integer*><MEMTYPE=*mtype*>>;

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

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

IC CREATE <*constraint-name*=> *constraint*
<MESSAGE='*message-string*' <MSGTYPE=USER>>;

IC DELETE *constraint-name-1* <...*constraint-name-n*> | _ALL_;

IC REACTIVATE *foreign-key-name* REFERENCES *libref*;

INDEX CENTILES *index-1* <...*index-n*>
</ <REFRESH><UPDATECENTILES= ALWAYS | NEVER | *integer*>>;

INDEX CREATE *index-specification-1* <...*index-specification-n*>
</ <NOMISS><UNIQUE><UPDATECENTILES=ALWAYS | NEVER | *integer*>>;

INDEX DELETE *index-1* <...*index-n*> | _ALL_;

INFORMAT *variable-1* <*informat-1*>
<…*variable-n*<*informat-n*>>;

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

RENAME *old-name-1*=*new-name-1*
<…*old-name-n*=*new-name-n*>;

REBUILD *SAS-file* </ ALTER=*password* GENNUM=*n* MEMTYPE=*mtype* NOINDEX>;

REPAIR *SAS-file-1* <...*SAS-file-n*>
</ <ALTER=*alter-password*><GENNUM=*integer*><MEMTYPE=*mtype*>>;

SAVE *SAS-file-1* <...*SAS-file-n*> </ MEMTYPE=*mtype*>;

Statement	Task	Example
"PROC DATASETS Statement"	Manage SAS files	
"AGE Statement"	Rename a group of related SAS files	Ex. 7
"APPEND Statement"	Add observations from one SAS data set to the end of another SAS data set	Ex. 8, Ex. 6, Ex. 9
"ATTRIB Statement"	Associates a format, informat, or label with variables in the SAS data set specified in the MODIFY statement	Ex. 1
"AUDIT Statement"	Initiate, control, suspend, resume, or terminate event logging to an audit file	
"CHANGE Statement"	Rename one or more SAS files	Ex. 2
"CONTENTS Statement"	Describe the contents of one or more SAS data sets and prints a directory of the SAS library	Ex. 9, Ex. 5, Ex. 10
"COPY Statement"	Copy all or some of the SAS files	Ex. 2

Statement	Task	Example
"DELETE Statement"	Delete SAS files	Ex. 2
"EXCHANGE Statement"	Exchange the names of two SAS files	Ex. 2
"EXCLUDE Statement"	Exclude SAS files from copying	Ex. 2
"FORMAT Statement"	Permanently assign, change, and remove variable formats	Ex. 4
"IC CREATE Statement"	Create an integrity constraint	
"IC DELETE Statement"	Delete an integrity constraint	
"IC REACTIVATE Statement"	Reactivate a foreign key integrity constraint	
"INDEX CENTILES Statement"	Update centiles statistics for indexed variables	
"INDEX CREATE Statement"	Create simple or composite indexes	Ex. 4
"INDEX DELETE Statement"	Delete one or more indexes	
"INFORMAT Statement"	Permanently assign, change, and remove variable informats	Ex. 4
"LABEL Statement"	Assign, change, and remove variable labels	Ex. 4
"MODIFY Statement"	Change the attributes of a SAS file and the attributes of variables	Ex. 4
"REBUILD Statement"	Specifies whether to restore or delete the disabled indexes and integrity constraints	
"RENAME Statement"	Rename variables in the SAS data set	Ex. 4
"REPAIR Statement"	Attempt to restore damaged SAS data sets or catalogs	
"SAVE Statement"	Delete all the SAS files except the ones listed in the SAVE statement	Ex. 3

Statement	Task	Example
"SELECT Statement"	Select SAS files for copying	Ex. 2

PROC DATASETS Statement

Manages SAS files.

Syntax

PROC DATASETS <*option-1* <*...option-n*>>;

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.

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=(*mtype(s)*)
: restricts processing to a certain type of SAS file.

NODETAILS
: see the description of DETAILS | NODETAILS.

NOLIST
: suppresses the printing of the directory.

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 373

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.

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.

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.

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 460

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 371 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). Refer to APPEND statement on page 386 for more information about the FORCE option.

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 376

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*.

Alias: 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.

See: For information about the WORK and USER libraries, see "One-level SAS Data Set Names" in Chapter 25 of *SAS Language Reference: Concepts*.

Example: "Example 2: Manipulating SAS Files" on page 460

MEMTYPE=(*mtype(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;
```

Alias: MTYPE=, MT=

Default: ALL

See: "Restricting Member Types for Processing" on page 374

NODETAILS
See "DETAILS | NODETAILS" on page 382.

NOLIST
suppresses the printing of the directory of the SAS files in the SAS log.

Note: If you specify the ODS RTF destination, 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.

Example: "Example 4: Modifying SAS Data Sets" on page 468

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 373

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 373

AGE Statement

Renames a group of related SAS files in a library.

Example: "Example 7: Aging SAS Data Sets" on page 475

Syntax

AGE *current-name related-SAS-file-1* <*...current-namerelated-SAS-file-n*> </ <ALTER=*alter-password*><MEMTYPE=*mtype*>>;

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-1* <*...related-SAS-file-n*>.

related-SAS-file-1 ... <related-SAS-file-n>
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 373

MEMTYPE=*mtype*
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 either in parentheses after the name of each SAS file or after a forward slash.

Alias: 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 374

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:

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.

Requirement: The BASE= data set must be a member of a SAS library that supports update processing.

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 its preappend 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.)

Example: "Example 6: Concatenating Two SAS Data Sets" on page 472

Syntax

APPEND BASE=<*libref.*>*SAS-data-set*
<APPENDVER=V6>
<DATA=<*libref.*>*SAS-data-set*>
<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 472

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 390

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 392

Example: "Example 6: Concatenating Two SAS Data Sets" on page 472

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: "Example 6: Concatenating Two SAS Data Sets" on page 472 and "Appending to Data Sets That Contain Variables with Different Attributes" on page 391

Example: "Appending to Data Sets with Different Variables" on page 391

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 higher
- 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.

Example: "Example 9: Getting Sort Indicator Information" on page 482

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

Details

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

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 the way you would use them in any other

SAS statement, which is 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 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 make use of available space.

For information about the COMPRESS= and REUSE= data set and system options, see *SAS Data Set Options: Reference* and *SAS System Options: Reference*.

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 DATA= 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 with Generation Groups

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

SAS Statements	Result

`proc datasets;`
`  append base=a`
`  data=b(gennum=2);`
	Appends historical version B#002 to base A

`proc datasets;`
`  append base=a(gennum=2)`
`  data=b(gennum=2);`
	Appends historical version B#002 to historical version A#002

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.

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 select 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 PROC DATASETS on page 394.

ATTRIB Statement

Associates a format, informat, or label with variables in the SAS data set specified in the MODIFY statement.

Restriction: Must appear in a MODIFY RUN group

Example: AUDIT statement on page 456

Syntax

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

Required Arguments

variable list
names the variables that you want to associate with the attributes. You can list the variables in any form that SAS allows.

attribute-list
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 456.

If you are not deleting or changing all attributes, it is easier to use the following statements, LABEL statement on page 429 , FORMAT Statement on page 421, and INFORMAT statement on page 428.

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.

See: "Understanding an Audit Trail" in Chapter 26 of *SAS Language Reference: Concepts*

Syntax

AUDIT *SAS-file* <(*SAS-password*)<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-1* <...*variable-n*>>;

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.

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 INTIATE or USER_VAR statement.

Statements

INITIATE
creates an audit file that has the same name as the SAS data file and a data set type of AUDIT. 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. 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. 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.

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 11: Initiating an Audit File" on page 491

LABEL='*variable-label*'
specifies a label for the variable.

length
specifies the length of the variable. If a length is not specified, the default is 8.

LOG
specifies the audit settings:

ADMIN_IMAGE=YES | NO
specifies whether the administrative events are logged to the audit file (that is, the SUSPEND and RESUME actions).

BEFORE_IMAGE=YES | NO
specifies whether the before-update record images are logged to the audit file.

DATA_IMAGE=YES | NO
specifies whether the added, deleted, and after-update record images are logged to the audit file.

ERROR_IMAGE=YES | NO
specifies whether the after-update record images are logged to the audit file.

Default: All images are logged by default (that is, all four are set to 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;`**

SUSPEND
suspends event logging to the audit file, but does not delete the audit file.

RESUME
resumes event logging to the audit file, if it was suspended.

TERMINATE
terminates event logging and deletes the audit file.

USER_VAR *variable-1 < … variable-n>*
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. The following syntax defines variables:

USER_VAR *variable-name-1* <$> <*length*> <LABEL='*variable-label*' > <… *variable-name-n*<$><*length*><LABEL='*variable-label*' >>
where

variable-name
is a name for the variable.

Restriction: The USER_VAR statement is optional. If specified, the USER_VAR statement must immediately follow the INITIATE statement for the applicable audit file.

$
indicates that the variable is a character variable.

Details

Creating an Audit File

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 SAS 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 TERMINIATE 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.

The following 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 460

Syntax

CHANGE *old-name-1=new-name-1*
<…*old-name-n=new-name-n*>
</ <ALTER=*alter-password*><GENNUM=ALL | *integer*><MEMTYPE=*mtype*>>;

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 460

Optional Arguments

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 373

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 376 and "Understanding Generation Data Sets" in Chapter 26 of *SAS Language Reference: Concepts*

MEMTYPE=*mtype*
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.

Alias: 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 374

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.

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 470

Syntax

CONTENTS <*option-1* <*...option-n*>>;

Summary of Optional Arguments

CENTILES
: prints centiles information for indexed variables.

DATA=*SAS-file-specification*
: specifies the input data set.

DETAILS | NODETAILS
: includes information in the output about the number of observations, number of variables, number of indexes, and data set labels.

DIRECTORY
: prints a list of the SAS files in the SAS library.

FMTLEN
: prints the length of a variable's informat or format.

MEMTYPE=(*mtype-1* <*...mtype-n*>)
: restricts processing to one or more types of SAS files.

NODETAILS
: see the description of DETAILS | NODETAILS.

NODS
: suppresses the printing of individual files.

NOPRINT

suppresses the printing of the output.

ORDER= COLLATE | CASECOLLATE | IGNORECASE | VARNUM
prints a list of variables in a specified order.

OUT=*SAS-data-set*
specifies the name for an output data set.

OUT2=*SAS-data-set*
specifies the name of an output data set to contain information about indexes and integrity constraints.

SHORT
prints abbreviated output.

VARNUM
print a list of the variables by their position in the data set. By default, the CONTENTS statement lists the variables alphabetically.

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 470

DETAILS | NODETAILS
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 381

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 401 to be printed.

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=(*mtype-1 <...mtype-n>*)
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;
```

Alias: MT=, MTYPE=

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.

NODETAILS
See "DETAILS | NODETAILS" on page 401.

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.

See: "VARNUM" on page 403

Note: The ORDER= option does not affect the order of the OUT= and OUT2= data sets.

Example: See "Example 10: Using the ORDER= Option with the CONTENTS Statement" on page 485 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 447 for a description of the variables in the OUT= data set. "Example 8: ODS Output " on page 476 for an example of how to get the CONTENTS output into an ODS data set for processing.

OUT2=*SAS-data-set*
names the output data set to contain information about indexes and integrity constraints.

Tips:
If UPDATECENTILES was not specified in the index definition, then the default value of 5 is used in the RECREATE 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 455 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

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 443.

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.

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.

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 and followed by character variables in the order in which they are encountered. If the data set only contains 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 in order 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;


NEW PROC CONTENTS RESULTS NEEDED HERE!!!!
                The SAS System

    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
```

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.

```
NEW PROC PRINT HERE.
```

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;


 The SAS System

        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.

Tips: See the example in PROC COPY on page 346 to migrate from a 32-bit machine to a 64-bit machine.

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 460

Syntax

COPY OUT=*libref-1*
<CLONE | NOCLONE>
<CONSTRAINT=YES | NO>
<DATECOPY>
<FORCE>
IN=*libref-2*
<INDEX=YES | NO>
<MEMTYPE=(*mtype-1 <...mtype-n>*)>
<MOVE <ALTER=*alter-password*>>;
<OVERRIDE=(*ds_option-1=value-1<...ds_option-n=value-n*>)>

Required Arguments

OUT=*libref-1*
names the SAS library to copy SAS files to.

Alias: OUTLIB= and OUTDD=

Example: "Example 2: Manipulating SAS Files" on page 460

IN=*libref-2*
names the SAS library containing SAS files to copy.

Alias: INLIB= and INDD=

Default: the libref of the procedure input library

Interaction: To copy only selected members, use the SELECT or EXCLUDE statements.

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 the MOVE option deletes the SAS file from the original data library, you need Alter access to move the SAS file.

See: "Using Passwords with the DATASETS Procedure" on page 373

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 16.2 *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.

Option	COPY Statement
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 16.3 *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 16.4 *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 16.5 *CLONE and the Data Representation Attribute*

Option	COPY Statement
CLONE	Results in a copy with the data representation of the input data set. However, specifying OUTREP= value in the OVERRIDE= option list results in a copy that uses the specified data representation
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 example, a file in Windows data representation is native to the Windows operating environment.

For the ENCODING= attribute, the following table summarizes how the COPY statement works.

Table 16.6 *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 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 16.7 *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. For more information, see "Understanding Integrity Constraints" in Chapter 26 of *SAS Language Reference: Concepts*.

DATECOPY
copies the SAS internal date and time when the SAS file was created and the date and time 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 429.

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.

FORCE
allows 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.

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=(*mtype-1 <...mtype-n>*)
restricts processing to one or more member types.

Alias: MT=, MTYPE=

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 412 and "Member Types" on page 375

Example: "Example 2: Manipulating SAS Files" on page 460

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.

Example: "Example 2: Manipulating SAS Files" on page 460

OVERRIDE=(*ds_option-1=value-1<...ds_option-n=value-n>*)
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.

Tip: When copying a data set stored in another host data representation or encoding, the default (or CLONE) behavior of COPY is to preserve the other host data representation or encoding in the new copy of the data set. By specifying OVERRIDE=(OUTREP=*session* ENCODING=*session*) in the COPY statement, the new copy of the data set is created in the host data representation and encoding of the SAS session that is executing the COPY.

NOCLONE
See the description of "CLONE | NOCLONE" on page 406.

Details

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 or an EXCLUDE statement, see "Specifying Member Types When Copying or Moving SAS Files" on page 412 and see Manipulating SAS Files on page 460 for an example. Also, see EXCLUDE statement on page 420 and SELECT statement on page 439.

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. For more information, see “Understanding Integrity Constraints” in *SAS Language Reference: Concepts*.

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;
```

Copying Views

The COPY statement with NOCLONE specified supports the OUTREP= and ENCODING= LIBNAME options for SQL views, DATA step views, and some SAS/ACCESS 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
```

Using the COPY Procedure Instead of the COPY Statement

Generally, the COPY procedure functions the same as the COPY statement in the DATASETS procedure. The following 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.

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. See “Strategies for Moving and Accessing SAS Files” in *Moving and Accessing SAS Files* for more information.

DELETE Statement

Deletes SAS files from a SAS library.

Example: “Example 2: Manipulating SAS Files” on page 460

Syntax

DELETE *SAS-file-1 <...SAS-file-n>*
</ <ALTER=*alter-password*>
<GENNUM=ALL | HIST | REVERT | *integer*>
<MEMTYPE=*mtype*>>;

Required Argument

SAS-file-1 <...SAS-file-n>
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 Lists” in the *SAS Language Reference: Concepts*.

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 373

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. 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). 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: "Understanding Generation Data Sets" in Chapter 26 of *SAS Language Reference: Concepts* and "Restricting Processing for Generation Data Sets" on page 376

MEMTYPE=*mtype*
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.

Alias: MT=, MTYPE=

Default: DATA

See: "Restricting Member Types for Processing" on page 374

Example: "Example 2: Manipulating SAS Files" on page 460

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.

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 460

Syntax

EXCHANGE *name-1=other-name-1* <…*name-n=other-name-n*>
</ <ALTER=*alter-password*><MEMTYPE=*mtype*>>;

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 373

MEMTYPE=*mtype*
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 374

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 only allows 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: Must follow a COPY statement
Cannot appear in the same COPY step with a SELECT statement

Example: "Example 2: Manipulating SAS Files" on page 460

Syntax

EXCLUDE *SAS-file-1*<*...SAS-file-n*> </ MEMTYPE=*mtype*>;

Required Argument

SAS-file-1 <...SAS-file-n>
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:

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=*mtype*
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.

Alias: 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 412 and "Restricting Member Types for Processing" on page 374

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: Must appear in a MODIFY RUN group

Example: "Example 4: Modifying SAS Data Sets" on page 468

Syntax

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

Required Argument

variable-1 <...variable-n>
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 393 and the _ALL_ keyword.

IC CREATE Statement

Creates an integrity constraint.

Restriction: Must be 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/numeric) and length.

See: "Understanding Integrity Constraints" in Chapter 26 of *SAS Language Reference: Concepts*

Syntax

IC CREATE <*constraint-name*=> *constraint* <MESSAGE='*message-string*' <MSGTYPE=USER>>;

Required Argument

constraint
is the type of constraint. The following is a list of valid values:

NOT NULL (*variable*)
specifies that *variable* does not contain a SAS missing value, including special missing values.

UNIQUE (*variables*)
specifies that the values of *variables* must be unique. This constraint is identical to DISTINCT.

DISTINCT (*variables*)
specifies that the values of *variables* must be unique. This constraint is identical to UNIQUE.

CHECK (WHERE-*expression*)
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 (*variables*)
specifies a primary key, that is, a set of variables that do not contain missing values and whose values are unique.

Requirement: 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.

Interaction: A primary key affects the values of an individual data file until it has a foreign key referencing it.

FOREIGN KEY (*variables*) REFERENCES *table-name* <ON DELETE *referential-action*> <ON UPDATE *referential-action*>
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 additionally 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:

Default Name	Constraint Type
NM*xxxx*	Not Null
UN*xxxx*	Unique

Default Name	Constraint Type
CK*xxxx*	Check
PK*xxxx*	Primary key
FK*xxxx*	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*' <MSGTYPE=USER>>

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';
```

<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.

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);
```

IC DELETE Statement

Deletes an integrity constraint.

Restriction: Must be in a MODIFY RUN group

See: "Understanding Integrity Constraints" in Chapter 26 of *SAS Language Reference: Concepts*

Syntax

IC DELETE *constraint-name-1<...constraint-name-n>* | _ALL_;

Required Arguments

constraint-name-1 <...constraint-name-n>

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_DUnique_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: Must be in a MODIFY RUN group

See: "Understanding Integrity Constraints" in Chapter 26 of *SAS Language Reference: Concepts*

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: Must be in a MODIFY RUN group

See: "Understanding SAS Indexes" in Chapter 26 of *SAS Language Reference: Concepts*

Syntax

INDEX CENTILES *index-1 <...index-n>*
</ <REFRESH><UPDATECENTILES= ALWAYS | NEVER | *integer*>>;

Required Argument

index-1 <..index-n>
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.

The following 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.

integer
is the percentage of values for the indexed variable that can be updated before centiles are refreshed.

Alias: UPDCEN

Default: 5 (percent)

INDEX CREATE Statement

Creates simple or composite indexes for the SAS data set specified in the MODIFY statement.

Restriction: Must be in a MODIFY RUN group

See: "Understanding SAS Indexes" in Chapter 26 of *SAS Language Reference: Concepts*

Example: "Example 4: Modifying SAS Data Sets" on page 468

Syntax

INDEX CREATE *index-specification-1 <...index-specification-n>*
</ <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_. For more information, refer to "Rules for Words and Names in the SAS Language" in *SAS Language Reference: Concepts*.

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';
```

Refer to *SAS Language Reference: Concepts*.

BY-group processing ignores indexes that are created with the NOMISS option.

Example: "Example 4: Modifying SAS Data Sets" on page 468

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 468

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. The following 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.

integer
specifies the percentage of values for the indexed variable that can be updated before centiles are refreshed.

Alias: UPDCEN

Default: 5 (percent)

INDEX DELETE Statement

Deletes one or more indexes associated with the SAS data set specified in the MODIFY statement.

Restriction: 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-1<...index-n>* | _ALL_;

Required Arguments

index-1 <...index-n>
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: Must appear in a MODIFY RUN group

Example: "Example 4: Modifying SAS Data Sets" on page 468

Syntax

INFORMAT *variable-1 <informat-1>*
<…variable-n<informat-n>>;

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 informats from a data set, use the ATTRIB statement on page 393 and the _ALL_ keyword.

LABEL Statement

Assigns, changes, and removes variable labels for the SAS data set specified in the MODIFY statement.

Restriction: Must appear in a MODIFY RUN group

Example: "Example 4: Modifying SAS Data Sets" on page 468

Syntax

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

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 393 and the _ALL_ keyword.

MODIFY Statement

Changes the attributes of a SAS file and, through the use of subordinate statements, the attributes of variables in the SAS file.

Restriction: You cannot change the length of a variable using the LENGTH= option on an ATTRIB statement.

Example: "Example 4: Modifying SAS Data Sets" on page 468

Syntax

MODIFY *SAS-file* <(*option-1* <*...option-n*>)>
</ <CORRECTENCODING=*encoding-value*><DTC=*SAS-date-time*>
<GENNUM=*integer*><MEMTYPE=*mtype*>>;

Summary of Optional Arguments

MEMTYPE=*mtype*
restricts processing to a certain type of SAS file.

Modify generation groups

GENMAX=*number-of-generations*
modifies the maximum number of versions for a generation group.

GENNUM=*integer*
modifies a historical version.

Modify passwords

ALTER=*password-modification*
modifies an Alter password.

PW=*password-modification*
modifies a Read, Write, or Alter password.

READ=*password-modification*
modifies a Read password.

WRITE=*password-modification*
modifies a Write password.

Specify data set attributes

CORRECTENCODING=*encoding-value*
changes the character-set encoding.

DTC=*SAS-date-time*
specifies a creation date and time.

LABEL='*data-set-label*' | ' '
assigns or change a data set label.

SORTEDBY=*sort-information*
specifies how the data are currently sorted.

TYPE=*special-type*
assigns or changes a special data set 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 433

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*

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.

DTC= can be used only when the resulting SAS file uses the V8 or V9 engine.

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.

See: "Understanding Generation Data Sets" in *SAS Language Reference: Concepts*

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 393.

See: "Example 4: Modifying SAS Data Sets" on page 468

MEMTYPE=*mtype*
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.

Alias: MTYPE= and 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* /
- /

See: "Manipulating Passwords" on page 433

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 433

Example: "Example 4: Modifying SAS Data Sets" on page 468

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 Lists" in Chapter 25 of *SAS Language Reference: Concepts*.

Example: "Example 4: Modifying SAS Data Sets" on page 468

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 433

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 468.

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;
```

REBUILD Statement

Specifies whether to restore or delete the disabled indexes and integrity constraints.

Default: Rebuild indexes and integrity constraints

Restriction: Data sets created in Version 7 or later

Syntax

REBUILD *SAS-file* </ <ALTER=*password*><GENNUM=*n*><MEMTYPE=*mtype*><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 Lists" in Chapter 25 of *SAS Language Reference: Concepts*.

Optional Arguments

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.

See: "Understanding Generation Data Sets" in Chapter 26 of *SAS Language Reference: Concepts*

MEMTYPE=*mtype*
restricts processing to one member type.

Alias: MT=, MTYPE=

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: Must appear in a MODIFY RUN group

Example: "Example 4: Modifying SAS Data Sets" on page 468

Syntax

RENAME *old-name-1=new-name-1*
<…*old-name-n=new-name-n*>;

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.

See: "Rules for Words and Names in the SAS Language" in Chapter 3 of *SAS Language Reference: Concepts*

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 to a usable condition.

Syntax

REPAIR *SAS-file-1 <...SAS-file-n>*
</ <ALTER=*alter-password*><GENNUM=*integer*><MEMTYPE=*mtype*>>;

Required Argument

SAS-file-1 <...SAS-file-n>
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 Lists" in Chapter 25 of *SAS Language Reference: Concepts*

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 373

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 376 and "Understanding Generation Data Sets" in Chapter 26 of *SAS Language Reference: Concepts*

MEMTYPE=*mtype*
restricts processing to one member type.

Alias: MT=, MTYPE=

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 374

Details

The most common situations that require the REPAIR statement are as follows:

- A system failure occurs while you are updating a SAS data set or catalog.
- The device on which a SAS data set or an associated index resides is damaged. In this case, you can restore the damaged data set or index from a backup device, but the data set and index no longer match.
- The disk that stores the SAS data set or catalog becomes full before the file is completely written to disk. You might need to free some disk space. PROC DATASETS requires free space when repairing SAS data sets with indexes and when repairing SAS catalogs.
- An I/O error occurs while you are writing a SAS data set or catalog entry.

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.

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, the version of SAS that you use must be able to update the catalog. Whether a SAS version can update a catalog (or just read it) is determined by the SAS version that created the catalog:

- A damaged Version 6 catalog can be repaired with Version 6 only.
- A damaged Version 8 catalog can be repaired with either Version 8 or SAS® 9, but not with Version 6.
- A damaged SAS 9 catalog can be repaired with SAS 9 only.

If the REPAIR operation is not successful, try to restore the SAS data set or catalog from your system's backup files.

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 and continue processing, use the NOWARN option in the PROC DATASETS statement.

If you are using Cross-Environment Data Access (CEDA) to process a damaged foreign SAS data set, CEDA cannot repair it. CEDA does not support update processing, which is required in order to repair a damaged data set. To repair the foreign file, you must move it back to its native environment. Note that observations might be lost during the repair process. For more information about CEDA, refer to “Definition of Cross-Environment Data Access (CEDA)” in Chapter 32 of *SAS Language Reference: Concepts*.

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 465

Syntax

SAVE *SAS-file-1 <...SAS-file-n>* </ MEMTYPE=*mtype*>;

Required Argument

SAS-file-1 <...SAS-file-n>
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=*mtype*
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.

Alias: MTYPE= and 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 374

Example: "Example 3: Saving SAS Files from Deletion" on page 465

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. Because 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: Must follow a COPY statement

Cannot appear with an EXCLUDE statement in the same COPY step

Example: "Example 2: Manipulating SAS Files" on page 460

Syntax

SELECT *SAS-file-1 <...SAS-file-n>*
</ <ALTER=*alter-password*><MEMTYPE= *mtype*>>;

Required Argument

SAS-file-1 <...SAS-file-n>
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, the SELECT statement allows only selection of the base 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 373

MEMTYPE=*mtype*
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.

Alias: MTYPE= and 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 412 and "Restricting Member Types for Processing" on page 374

Example: "Example 2: Manipulating SAS Files" on page 460

Details

Selecting Several Like-Named Files

You can use shortcuts for listing several SAS files in 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.

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 GROUP data set, which is shown in the following output.

Only the items in the output that require explanation are discussed.

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).

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 445 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. For example, a file that is in Windows data representation is native to the Windows operating environment.

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.

Data Set Name	HEALTH.GROUP	Observations	148
Member Type	DATA	Variables	11
Engine	V9	Indexes	1
Created	Wed, Sep 12, 2007 01:57:49 PM	Observation Length	96
Last Modified	Wed, Sep 12, 2007 04:01:15 PM	Deleted Observations	0
Protection	READ	Compressed	NO
Data Set Type		Sorted	YES
Label	Test Subjects		
Data Representation	WINDOWS_32		
Encoding	wlatin1 Western (Windows)		

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. The following output is from the Windows operating environment.

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
Filename	c:\\procdatasets\group.sas7bdat
Release Created	9.0201B0
Host Created	XP_PRO

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.

Alphabetic List of Variables and Attributes						
#	Variable	Type	Len	Format	Informat	Label
9	BIRTH	Num	8	DATE7.	DATE7.	
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.		current salary excluding bonus
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.

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 or NODUPREC option when sorting the data set. This field does not appear if you did not use one of these options in a PROC SORT statement (not shown).

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 16.8 *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 16.9 *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 *
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

ODS Table	Description	Generates Table
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
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 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
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

* 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.

NODUPREC
indicates whether the NODUPREC 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. The following are 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. The following are 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.

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 or use the NOTSORTED option in the BY statement.

The following 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 470 and in "Procedure Output" on page 441.

Due to the size of the HEALTH.GRPOUT, the following output is in five sections.

Output 16.1 *An Example of an Output Data Set — Section 1*

Obs	LIBNAME	MEMNAME	MEMLABEL	TYPEMEM	NAME	TYPE	LENGTH	VARNUM
1	HEALTH	GROUP	Test Subjects		BIRTH	1	8	9
2	HEALTH	GROUP	Test Subjects		CITY	2	15	4
3	HEALTH	GROUP	Test Subjects		FNAME	2	15	3
4	HEALTH	GROUP	Test Subjects		HIRED	1	8	10
5	HEALTH	GROUP	Test Subjects		HPHONE	2	12	11
6	HEALTH	GROUP	Test Subjects		IDNUM	2	4	1
7	HEALTH	GROUP	Test Subjects		JOBCODE	2	4	7
8	HEALTH	GROUP	Test Subjects		LNAME	2	15	2
9	HEALTH	GROUP	Test Subjects		SALARY	1	8	8
10	HEALTH	GROUP	Test Subjects		SEX	2	2	6
11	HEALTH	GROUP	Test Subjects		STATE	2	3	5

Output 16.2 *An Example of an Output Data Set — Section 2*

LABEL	FORMAT	FORMATL	FORMATD	INFORMAT	INFORML
	DATE	7	0	DATE	7
		0	0		0
		0	0		0
	DATE	7	0	DATE	7
		0	0		0
		0	0		0
		0	0		0
		0	0		0
current salary excluding bonus	COMMA	8	0		0
		0	0		0
		0	0		0

Output 16.3 *An Example of an Output Data Set — Section 3*

INFORMD	JUST	NPOS	NOBS	ENGINE	CRDATE	MODATE	DELOBS
0	1	8	148	V9	12SEP07:13:57:50	12SEP07:16:01:16	0
0	0	58	148	V9	12SEP07:13:57:50	12SEP07:16:01:16	0
0	0	43	148	V9	12SEP07:13:57:50	12SEP07:16:01:16	0
0	1	16	148	V9	12SEP07:13:57:50	12SEP07:16:01:16	0
0	0	82	148	V9	12SEP07:13:57:50	12SEP07:16:01:16	0
0	0	24	148	V9	12SEP07:13:57:50	12SEP07:16:01:16	0
0	0	78	148	V9	12SEP07:13:57:50	12SEP07:16:01:16	0
0	0	28	148	V9	12SEP07:13:57:50	12SEP07:16:01:16	0
0	1	0	148	V9	12SEP07:13:57:50	12SEP07:16:01:16	0
0	0	76	148	V9	12SEP07:13:57:50	12SEP07:16:01:16	0
0	0	73	148	V9	12SEP07:13:57:50	12SEP07:16:01:16	0

Output 16.4 *An Example of an Output Data Set — Section 4*

IDXUSAGE	MEMTYPE	IDXCOUNT	PROTECT	FLAGS	COMPRESS	REUSE	SORTED	SORTEDBY
COMPOSITE	DATA	1	R--	---	NO	NO	0	.
NONE	DATA	1	R--	---	NO	NO	0	.
NONE	DATA	1	R--	---	NO	NO	0	.
NONE	DATA	1	R--	---	NO	NO	0	.
NONE	DATA	1	R--	---	NO	NO	0	.
NONE	DATA	1	R--	---	NO	NO	0	.
NONE	DATA	1	R--	---	NO	NO	0	.
NONE	DATA	1	R--	---	NO	NO	0	1
COMPOSITE	DATA	1	R--	---	NO	NO	0	.
NONE	DATA	1	R--	---	NO	NO	0	.
NONE	DATA	1	R--	---	NO	NO	0	.

Output 16.5 *An Example of an Output Data Set — Section 5*

CHARSET	COLLATE	NODUPKEY	NODUPREC	ENCRYPT	POINTOBS	GENMAX	GENNUM	GENNEXT
ANSI		NO	NO	NO	YES	0	.	.
ANSI		NO	NO	NO	YES	0	.	.
ANSI		NO	NO	NO	YES	0	.	.
ANSI		NO	NO	NO	YES	0	.	.
ANSI		NO	NO	NO	YES	0	.	.
ANSI		NO	NO	NO	YES	0	.	.
ANSI		NO	NO	NO	YES	0	.	.
ANSI		NO	NO	NO	YES	0	.	.
ANSI		NO	NO	NO	YES	0	.	.
ANSI		NO	NO	NO	YES	0	.	.
ANSI		NO	NO	NO	YES	0	.	.

Note: For information about how to get the CONTENTS output into an ODS data set for processing, see "Example 8: ODS Output " on page 476.

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: MODIFY statement option
ATTRIB

CONTENTS statement

Other features: PROC CONTENTS

Program

```
options ls=79 nodate nocenter;
title;

proc format;
   value clsfmt 1='Freshman' 2='Sophmore' 3='Junior' 4='Senior';
run;

data 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=class;
run;
```

Program Description

The following example deletes all labels and formats within a data set.

Set the following system options.

```
options ls=79 nodate nocenter;
title;
```

Create a user defined FORMAT with a value of CLSFMT.

```
proc format;
   value clsfmt 1='Freshman' 2='Sophmore' 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 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=class;
run;
```

CONTENTS Procedure with Labels and Format

The CONTENTS Procedure

Data Set Name	EXAMPLE.CLASS	Observations	2
Member Type	DATA	Variables	3
Engine	V9	Indexes	0
Created	Sat, Feb 05, 2011 09:01:55 AM	Observation Length	24
Last Modified	Sat, Feb 05, 2011 09:01:55 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	4096
Number of Data Set Pages	1
First Data Page	1
Max Obs per Page	168
Obs in First Data Page	2
Number of Data Set Repairs	0
Filename	c:\procdatasets\class.sas7bdat
Release Created	9.0301B0
Host Created	XP_PRO

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

Within PROC DATASETS, remove all the labels and formats using the MODIFY statement and the ATTRIB option.

```
proc datasets lib=work memtype=data;
   modify class;
     attrib _all_ label=' ';
     attrib _all_ format=;
run;
```

Use the CONTENTS statement within PROC DATASETS to view the contents of the data set without the labels and format.

```
contents data=class;
run;
quit;
```

CONTENTS Statement without Labels and Format

The DATASETS Procedure

Data Set Name	EXAMPLE.CLASS	Observations	2
Member Type	DATA	Variables	3
Engine	V9	Indexes	0
Created	Sat, Feb 05, 2011 09:08:39 AM	Observation Length	24
Last Modified	Sat, Feb 05, 2011 09:08:56 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	4096
Number of Data Set Pages	1
First Data Page	1
Max Obs per Page	168
Obs in First Data Page	2
Number of Data Set Repairs	0
Filename	c:\procdatasets\class.sas7bdat
Release Created	9.0301B0
Host Created	XP_PRO

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=

CHANGE statement

COPY statement options
MEMTYPE
MOVE
OUT=

DELETE statement option
MEMTYPE=

EXCHANGE statement

EXCLUDE statement

SELECT statement option
MEMTYPE=

Details

This example does the following actions:

- 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

Write the programming statements to the SAS log. The SOURCE system option accomplishes this.

```
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;
```

SAS Log

```
117    options pagesize=60 linesize=80 nodate pageno=1 source;
118  LIBNAME dest1 'c:\Documents and Settings\mydir\My
118! Documents\procdatasets\dest1';
NOTE: Libref DEST1 was successfully assigned as follows:
      Engine:        V9
      Physical Name: c:\Documents and Settings\mydir\My
      Documents\procdatasets\dest1
119  LIBNAME dest2 'c:\Documents and Settings\mydir\My
119! Documents\procdatasets\dest2';
NOTE: Libref DEST2 was successfully assigned as follows:
      Engine:        V9
      Physical Name: c:\Documents and Settings\mydir\My
      Documents\procdatasets\dest2
120  LIBNAME health 'c:\Documents and Settings\mydir\My
120! Documents\procdatasets\health';
NOTE: Libref HEALTH was successfully assigned as follows:
      Engine:        V9
      Physical Name: c:\Documents and Settings\mydir\My
      Documents\procdatasets\health


121  proc datasets library=health details;
                                Directory

Libref         HEALTH
Engine         V9
Physical Name  \myfiles\health
Filename       \myfiles\health



                  Member   Obs, Entries
        #  Name   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: SAVE statement option:
MEMTYPE=

This example uses 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

Write the programming statements to the SAS log. SAS option SOURCE writes all programming statements to the log.

```
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;
```

SAS Log

```
161
162   options pagesize=40 linesize=80 nodate pageno=1 source;
163  LIBNAME elder 'c:\Documents and Settings\mydir\My
163! Documents\procdatasets\elder';
NOTE: Libref ELDER was successfully assigned as follows:
      Engine:        V9
      Physical Name: c:\Documents and Settings\mydir\My
      Documents\procdatasets\elder
164  LIBNAME green 'c:\Documents and Settings\mydir\My
164! Documents\procdatasets\green';
NOTE: Libref GREEN was successfully assigned as follows:
      Engine:        V9
      Physical Name: c:\Documents and Settings\mydir\My
      Documents\procdatasets\green
165  proc copy in=green out=elder;

NOTE: Copying GREEN.AGING to ELDER.AGING (memtype=DATA).
NOTE: There were 1 observations read from the data set GREEN.AGING.
NOTE: The data set ELDER.AGING has 1 observations and 2 variables.
NOTE: Copying GREEN.ALCOHOL to ELDER.ALCOHOL (memtype=DATA).
NOTE: There were 1 observations read from the data set GREEN.ALCOHOL.
NOTE: The data set ELDER.ALCOHOL has 1 observations and 2 variables.
NOTE: Copying GREEN.BACKPAIN to ELDER.BACKPAIN (memtype=DATA).
NOTE: There were 1 observations read from the data set GREEN.BACKPAIN.
NOTE: The data set ELDER.BACKPAIN has 1 observations and 2 variables.
NOTE: Copying GREEN.CHRONIC to ELDER.CHRONIC (memtype=DATA).
NOTE: There were 1 observations read from the data set GREEN.CHRONIC.
NOTE: The data set ELDER.CHRONIC has 1 observations and 2 variables.
NOTE: Copying GREEN.CLINICS to ELDER.CLINICS (memtype=CATALOG).
NOTE: Copying GREEN.CLINICS to ELDER.CLINICS (memtype=DATA).
NOTE: There were 1 observations read from the data set GREEN.CLINICS.
NOTE: The data set ELDER.CLINICS has 1 observations and 2 variables.
NOTE: Copying GREEN.DISEASE to ELDER.DISEASE (memtype=DATA).
NOTE: There were 1 observations read from the data set GREEN.DISEASE.
NOTE: The data set ELDER.DISEASE has 1 observations and 2 variables.
NOTE: Copying GREEN.GROWTH to ELDER.GROWTH (memtype=DATA).
NOTE: There were 1 observations read from the data set GREEN.GROWTH.
NOTE: The data set ELDER.GROWTH has 1 observations and 2 variables.
NOTE: Copying GREEN.HOSPITAL to ELDER.HOSPITAL (memtype=CATALOG).
NOTE: PROCEDURE COPY used (Total process time):
      real time           2.42 seconds
      cpu time            0.04 seconds


166  proc datasets lib=elder;
                                 Directory

Libref         ELDER
Engine         V9
Physical Name  \myfiles\elder
Filename       \myfiles\elder


                       Member     File
          #  Name      Type       Size  Last Modified

          1  AGING     DATA       5120  12Sep07:15:52:52
          2  ALCOHOL   DATA       5120  12Sep07:15:52:52
          3  BACKPAIN  DATA       5120  12Sep07:15:52:53
          4  CHRONIC   DATA       5120  12Sep07:15:52:53
          5  CLINICS   CATALOG   17408  12Sep07:15:52:53
          6  CLINICS   DATA       5120  12Sep07:15:52:53
          7  DISEASE   DATA       5120  12Sep07:15:52:54
          8  GROWTH    DATA       5120  12Sep07:15:52:54
          9  HOSPITAL  CATALOG   17408  12Sep07:15:52:54
```

```
167  save chronic aging clinics / memtype=data;
168  run;

NOTE: Saving ELDER.CHRONIC (memtype=DATA).
NOTE: Saving ELDER.AGING (memtype=DATA).
NOTE: Saving ELDER.CLINICS (memtype=DATA).
NOTE: Deleting ELDER.ALCOHOL (memtype=DATA).
NOTE: Deleting ELDER.BACKPAIN (memtype=DATA).
NOTE: Deleting ELDER.CLINICS (memtype=CATALOG).
NOTE: Deleting ELDER.DISEASE (memtype=DATA).
NOTE: Deleting ELDER.GROWTH (memtype=DATA).
NOTE: Deleting ELDER.HOSPITAL (memtype=CATALOG).
```

Example 4: Modifying SAS Data Sets

Features:

PROC DATASETS statement option
- NOLIST

FORMAT statement

INDEX CREATE statement options
- NOMISS
- UNIQUE

INFORMAT statement

LABEL statement

MODIFY statement options
- LABEL=
- READ=
- SORTEDBY=

RENAME 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 470 shows the modifications to the GROUP data set.

This example includes the following actions:

- 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

Write the programming statements to the SAS log. SAS option SOURCE writes the programming statements to the log.

```
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;
```

SAS Log

```
169  options pagesize=40 linesize=80 nodate pageno=1 source;
170  LIBNAME health 'c:\Documents and Settings\mydir\My
170! Documents\procdatasets\health';
NOTE: Libref HEALTH was successfully assigned as follows:
      Engine:        V9
      Physical Name: c:\Documents and Settings\mydir\My
      Documents\procdatasets\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: CONTENTS statement option
DATA=

Other features: SAS data set option
READ=

Details

This example shows 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 468.

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;
```

Program Description

```
options pagesize=40 linesize=80 nodate pageno=1;

LIBNAME health
'SAS-library';
```

Specify HEALTH as the procedure input library, and suppress the directory listing.

```
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;
```

Output 16.6 *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	Tuesday, March 22, 2005 09:00:23 AM	Observation Length	96
Last Modified	Tuesday, March 22, 2005 09:00:23 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	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
Filename	c:\procdatasets\group.sas7bdat
Release Created	9.0201B0
Host Created	XP_PRO

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		

Example 6: Concatenating Two SAS Data Sets

Features: APPEND statement options
BASE=
DATA=
FORCE=

Program

The EXP.RESULTS Data Set 1

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

The EXP.SUR Data Set 2

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

Program Description

This example appends one data set to the end of another data set.

The BASE= data set, EXP.RESULTS

The EXP.RESULTS Data Set 1

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

The data set EXP.SUR contains the variable WT6MOS, but the EXP.RESULTS data set does not.

The EXP.SUR Data Set 2

```
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
```

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 EXP.RESULTS Data Set';
run;
```

Program Description

```
options pagesize=40 linesize=64 nodate pageno=1;

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 EXP.RESULTS Data Set';
run;
```

Output

Output 16.7 *Output*

The EXP.RESULTS Data Set 1

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: AGE statement

This example shows 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

Write the programming statements to the SAS log. SAS option SOURCE writes the programming statements to the log.

```
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;
```

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: ODS Output

Features: CONTENTS Statement

The example shows how to get PROC CONTENTS output into an ODS output data set for processing.

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

Program

```
title1 "PROC CONTENTS ODS Output";

options nodate nonumber nocenter formdlim='-';

data a;
   x=1;
run;


ods output attributes=atr
           variables=var
           enginehost=eng;

ods listing close;

proc contents data=a;
run;

ods listing;

title2 "all Attributes data";

proc print data=atr noobs;
run;

title2 "all Variables data";
```

```
proc print data=var noobs;
run;

title2 "all EngineHost data";

proc print data=eng noobs;
run;

ods output attributes=atr1(keep=member cvalue1 label1
   where=(attribute in ('Data Representation','Encoding'))
   rename=(label1=attribute cvalue1=value))
   attributes=atr2(keep=member cvalue2 label2
   where=(attribute in ('Observations', 'Variables'))
   rename=(label2=attribute cvalue2=value));


ods listing close;

proc contents data=a;
run;

ods listing;

data final;
   set atr1 atr2;
run;

title2 "example of post-processing of ODS output data";

proc print data=final noobs;
run;

ods listing close;
```

Program Description

```
title1 "PROC CONTENTS ODS Output";

options nodate nonumber nocenter formdlim='-';

data a;
   x=1;
run;
```

Use the ODS OUTPUT statement to specify data sets to which CONTENTS data is directed.

```
ods output attributes=atr
           variables=var
           enginehost=eng;
```

Temporarily suppress output to the lst.

```
ods listing close;
```

```
proc contents data=a;
run;
```

Resume output to the lst.

```
ods listing;

title2 "all Attributes data";

proc print data=atr noobs;
run;

title2 "all Variables data";

proc print data=var noobs;
run;

title2 "all EngineHost data";

proc print data=eng noobs;
run;
```

Select specific data from ODS output.

```
ods output attributes=atr1(keep=member cvalue1 label1
   where=(attribute in ('Data Representation','Encoding'))
   rename=(label1=attribute cvalue1=value))
   attributes=atr2(keep=member cvalue2 label2
   where=(attribute in ('Observations', 'Variables'))
   rename=(label2=attribute cvalue2=value));


ods listing close;

proc contents data=a;
run;

ods listing;

data final;
   set atr1 atr2;
run;

title2 "example of post-processing of ODS output data";

proc print data=final noobs;
run;

ods listing close;
```

Results

Output 16.8 *All Attributes Data*

PROC CONTENTS ODS Output
all Attributes data

Member	Label1	cValue1	nValue1	Label2	cValue2	nValue2
ODS.A	Data Set Name	ODS.A	.	Observations	1	1.000000
ODS.A	Member Type	DATA	.	Variables	1	1.000000
ODS.A	Engine	V9	.	Indexes	0	0
ODS.A	Created	Sat, Feb 05, 2011 09:53:52 AM	1612518832	Observation Length	8	8.000000
ODS.A	Last Modified	Sat, Feb 05, 2011 09:53:52 AM	1612518832	Deleted Observations	0	0
ODS.A	Protection		.	Compressed	NO	.
ODS.A	Data Set Type		.	Sorted	NO	.
ODS.A	Label		.			0
ODS.A	Data Representation	WINDOWS_32	.			0
ODS.A	Encoding	wlatin1 Western (Windows)	.			0

Output 16.9 *All Variables Data*

PROC CONTENTS ODS Output
all Variables data

Member	Num	Variable	Type	Len	Pos
ODS.A	1	x	Num	8	0

Output 16.10 *All EngineHost Data*

PROC CONTENTS ODS Output
all EngineHost data

Member	Label1	cValue1	nValue1
ODS.A	Data Set Page Size	4096	4096.000000
ODS.A	Number of Data Set Pages	1	1.000000
ODS.A	First Data Page	1	1.000000
ODS.A	Max Obs per Page	501	501.000000
ODS.A	Obs in First Data Page	1	1.000000
ODS.A	Number of Data Set Repairs	0	0
ODS.A	Filename	c:\procdatasets\a.sas7bdat	.
ODS.A	Release Created	9.0301B0	.
ODS.A	Host Created	XP_PRO	.

Output 16.11 *PROC CONTENTS for All EngineHost Data*

PROC CONTENTS ODS Output
all EngineHost data

The CONTENTS Procedure

Data Set Name	ODS.A	Observations	1
Member Type	DATA	Variables	1
Engine	V9	Indexes	0
Created	Sat, Feb 05, 2011 09:53:52 AM	Observation Length	8
Last Modified	Sat, Feb 05, 2011 09:53:52 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	4096
Number of Data Set Pages	1
First Data Page	1
Max Obs per Page	501
Obs in First Data Page	1
Number of Data Set Repairs	0
Filename	c:\procdatasets\a.sas7bdat
Release Created	9.0301B0
Host Created	XP_PRO

Alphabetic List of Variables and Attributes			
#	Variable	Type	Len
1	x	Num	8

Output 16.12 *Post-processing of ODS Output Data*

PROC CONTENTS ODS Output
example of post-processing of ODS output data

Member	attribute	value
ODS.A	Data Representation	WINDOWS_32
ODS.A	Encoding	wlatin1 Western (Windows)
ODS.A	Observations	1
ODS.A	Variables	1

Example 9: Getting Sort Indicator Information

Features: APPEND statement option
GETSORT
SORTEDBY data set option

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;
```

Program Description

The following example shows that a sort indicator can be inherited using the GETSORT option with the APPEND statement.

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;
```

Output 16.13 *Sort Information Output*

The RESULTS Data Set

The CONTENTS Procedure

Sort Information	
Sortedby	DESCENDING var1
Validated	YES
Character Set	ANSI

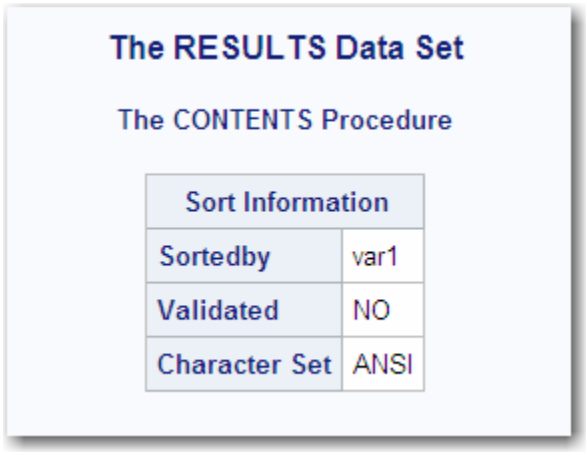
This example shows sort indicators using the SORTEDBY data set option and the SORT procedure.

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;
```

Output 16.14 *Sort Information Output*



The RESULTS Data Set

The CONTENTS Procedure

Sort Information	
Sortedby	var1
Validated	NO
Character Set	ANSI

This example shows the sort indicator information using the SORT procedure.

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 16.15 *Sort Information Output*

The RESULTS Data Set

The CONTENTS Procedure

Sort Information	
Sortedby	var1
Validated	YES
Character Set	ANSI

Example 10: Using the ORDER= Option with the CONTENTS Statement

Features: CONTENTS statement options
- ORDER=
- COLLATE
- CASECOLLATE
- IGNORECASE
- VARNUM

Program

```
options nonotes nodate nonumber nocenter formdlim  ='-';

libname contents 'C:\proccontents';

data contents.test;
   d=2;
   b001 =1;
   b002 =2;
   b003 =3;
   b001z=1;
   B001a=2;
   CaSeSeNsItIvE2=9;
   CASESENSITIVE3=9;
   D=2;
   casesensitive1=9;
   CaSeSeNsItIvE1a=9;
   d001z=1;
   CASESENSITIVE1C=9;
   D001a=2;
   casesensitive1b=9;
   A =1;
   a002 =2;
   a =3;
   a001z=1;
   A001a=2;
```

```
run;

%let mydata=contents.test;

ods output Variables=var1(keep=Num Variable);
ods listing close;

proc contents data=&mydata;
run;

ods listing;
    title "Default options";

proc print data=var1 noobs;
run;

ods output Variables=var2(keep=Num Variable);
ods listing close;

proc contents order=collate data=&mydata;
run;

ods listing;
    title "order=collate option";

proc print data=var2 noobs;
run;

ods output Variables=var3(keep=Num Variable);
ods listing close;

proc contents order=casecollate data=&mydata;
run;

ods listing;
    title "order=casecollate option";

proc print data=var3 noobs;
run;

ods output Variables=var4(keep=Num Variable);
ods listing close;

proc contents order=ignorecase data=&mydata;
run;

ods listing;
    title "order=ignorecase option";

proc print data=var4 noobs;
run;

ods output Position=var5(keep=Num Variable);
ods listing close;

proc contents data=&mydata varnum;
```

```
run;

ods listing;
    title "varnum option";

proc print data=var5 noobs;
run;
```

Program Description

Set up the data set.

```
options nonotes nodate nonumber nocenter formdlim  ='-';

libname contents 'C:\proccontents';

data contents.test;
   d=2;
   b001 =1;
   b002 =2;
   b003 =3;
   b001z=1;
   B001a=2;
   CaSeSeNsItIvE2=9;
   CASESENSITIVE3=9;
   D=2;
   casesensitive1=9;
   CaSeSeNsItIvE1a=9;
   d001z=1;
   CASESENSITIVE1C=9;
   D001a=2;
   casesensitive1b=9;
   A =1;
   a002 =2;
   a =3;
   a001z=1;
   A001a=2;

run;
```

To produce PROC CONTENTS output for a data set of your choice, change data set name to MYDATA.

```
%let mydata=contents.test;

ods output Variables=var1(keep=Num Variable);
ods listing close;

proc contents data=&mydata;
run;

ods listing;
    title "Default options";

proc print data=var1 noobs;
run;
```

```
ods output Variables=var2(keep=Num Variable);
ods listing close;

proc contents order=collate data=&mydata;
run;

ods listing;
    title "order=collate option";

proc print data=var2 noobs;
run;

ods output Variables=var3(keep=Num Variable);
ods listing close;

proc contents order=casecollate data=&mydata;
run;

ods listing;
    title "order=casecollate option";

proc print data=var3 noobs;
run;

ods output Variables=var4(keep=Num Variable);
ods listing close;

proc contents order=ignorecase data=&mydata;
run;

ods listing;
    title "order=ignorecase option";

proc print data=var4 noobs;
run;
```

VARNUM option is used. Note that the name of the ODS output object is different when the VARNUM option is used.

```
ods output Position=var5(keep=Num Variable);
ods listing close;

proc contents data=&mydata varnum;
run;

ods listing;
    title "varnum option";

proc print data=var5 noobs;
run;
```

Using the COLLATE and CASECOLLATE Options

The following output shows the results of the ORDER= default, the COLLATE option, and the CASECOLLATE option.

Default options

Num	Variable
15	A
18	A001a
6	B001a
8	CASESENSITIVE3
12	CASESENSITIVE1C
7	CaSeSeNsItIvE2
10	CaSeSeNsItIvE1a
13	D001a
16	a002
17	a001z
2	b001
3	b002
4	b003
5	b001z
9	casesensitive1
14	casesensitive1b
1	d
11	d001z

order=collate option

Num	Variable
15	A
18	A001a
6	B001a
12	CASESENSITIVE1C
8	CASESENSITIVE3
10	CaSeSeNsItIvE1a
7	CaSeSeNsItIvE2
13	D001a
17	a001z
16	a002
2	b001
5	b001z
3	b002
4	b003
9	casesensitive1
14	casesensitive1b
1	d
11	d001z

order=casecollate option

Num	Variable
15	A
18	A001a
17	a001z
16	a002
2	b001
6	B001a
5	b001z
3	b002
4	b003
9	casesensitive1
10	CaSeSeNsItIvE1a
14	casesensitive1b
12	CASESENSITIVE1C
7	CaSeSeNsItIvE2
8	CASESENSITIVE3
1	d
13	D001a
11	d001z

The following output shows the results of the ORDER= default, IGNORECASE option, and VARNUM option.

Output 16.16 *Results of Using the IGNORECASE and VARNUM Options*

Default options

Num	Variable
15	A
18	A001a
6	B001a
8	CASESENSITIVE3
12	CASESENSITIVE1C
7	CaSeSeNsItIvE2
10	CaSeSeNsItIvE1a
13	D001a
16	a002
17	a001z
2	b001
3	b002
4	b003
5	b001z
9	casesensitive1
14	casesensitive1b
1	d
11	d001z

order=ignorecase option

Num	Variable
15	A
16	a002
18	A001a
17	a001z
2	b001
3	b002
4	b003
6	B001a
5	b001z
9	casesensitive1
7	CaSeSeNsItIvE2
8	CASESENSITIVE3
10	CaSeSeNsItIvE1a
14	casesensitive1b
12	CASESENSITIVE1C
1	d
13	D001a
11	d001z

varnum option

Num	Variable
1	d
2	b001
3	b002
4	b003
5	b001z
6	B001a
7	CaSeSeNsItIvE2
8	CASESENSITIVE3
9	casesensitive1
10	CaSeSeNsItIvE1a
11	d001z
12	CASESENSITIVE1C
13	D001a
14	casesensitive1b
15	A
16	a002
17	a001z
18	A001a

Results

The following output shows the results of the ORDER= default, the COLLATE option, and the CASECOLLATE option:

The following table shows the results of the ORDER= default, IGNORECASE option, and VARNUM option.

Table 16.10 *Results of Using the IGNORECASE and VARNUM Options*

default	IGNORECASE	VARNUM

Num	Variable	Num	Variable	Num	Variable
15	A	15	A	1	d
18	A001a	16	a002	2	b001
6	B001a	18	A001a	3	b002
8	CASESENSITIVE3	17	a001z	4	b003
12	CASESENSITIVE1C	2	b001	5	b001z
7	CaSeSeNsItIvE2	3	b002	6	B001a
10	CaSeSeNsItIvE1a	4	b003	7	CaSeSeNsItIvE2
13	D001a	6	B001a	8	CASESENSITIVE3
16	a002	5	b001z	9	casesensitive1
17	a001z	9	casesensitive1	10	CaSeSeNsItIvE1a
2	b001	7	CaSeSeNsItIvE2	11	d001z
3	b002	8	CASESENSITIVE3	12	CASESENSITIVE1C
4	b003	10	CaSeSeNsItIvE1a	13	D001a
5	b001z	14	casesensitive1b	14	casesensitive1b
9	casesensitive1	12	CASESENSITIVE1C	15	A
14	casesensitive1b	1	d	16	a002
1	d	13	D001a	17	a001z
11	d001z	11	d001z	18	A001a

Example 11: Initiating an Audit File

Features: AUDIT Statements and Options
- AUDIT
- INITIATE
- USER_VAR
- LOG
- SUSPEND
- RESUME
- TERMINATE

Program

```
libname mylib "c:\mylib";

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;   /* resume audit file */
audit inventory;
resume;
quit;

/* additional step(s) which update the inventory dataset could go here*/

proc datasets lib=mylib;  /* terminate audit file */
audit inventory;
terminate;
quit;
```

Initiates an audit file using USER_VAR.

```
libname mylib "c:\mylib";

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;
```

```
1    options nocenter;
2
3    libname mylib "c:\mylib";
NOTE: Libref MYLIB was successfully assigned as follows:
      Engine:        V9
      Physical Name: c:\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
```

The SAS System

Directory	
Libref	MYLIB
Engine	V9
Physical Name	c:\mylib
Filename	c:\mylib

#	Name	Member Type	File Size	Last Modified
1	INVENTORY	DATA	5120	01Mar11:12:45:51

The SAS System

Directory	
Libref	MYLIB
Engine	V9
Physical Name	c:\mylib
Filename	c:\mylib

#	Name	Member Type	File Size	Last Modified
1	INVENTORY	DATA	5120	01Mar11:12:50:09
	INVENTORY	AUDIT	5120	01Mar11:12:50:09

The SAS System

vendor	item	description	units	reason	_ATDATETIME_	_ATOBSNO_	_ATRETURNCODE_	_ATUSERID_	_ATOPCODE_	_ATMESSAGE_
Bordens	B132	Milk	100	increase on hand	01MAR2011:12:50:05	4	.	suholm	DA	
Tropicana	B002	OrangeJuice	45		01MAR2011:12:50:10	2	.	suholm	DR	
Tropicana	B002	OrangeJuice	10	recounted inventory	01MAR2011:12:50:10	2	.	suholm	DW	

The SAS System

Directory	
Libref	MYLIB
Engine	V9
Physical Name	c:\mylib
Filename	c:\mylib

#	Name	Member Type	File Size	Last Modified
1	INVENTORY	DATA	5120	01Mar11:12:50:09
	INVENTORY	AUDIT	5120	01Mar11:12:50:09

The SAS System

Directory	
Libref	MYLIB
Engine	V9
Physical Name	c:\mylib
Filename	c:\mylib

#	Name	Member Type	File Size	Last Modified
1	INVENTORY	DATA	5120	01Mar11:12:50:09
	INVENTORY	AUDIT	5120	01Mar11:12:50:09

Chapter 17

DISPLAY Procedure

Overview: DISPLAY Procedure

The DISPLAY procedure executes SAS/AF applications. These applications consist of a variety of entries that are stored in a SAS catalog and that have been built with the BUILD procedure in SAS/AF software. For complete documentation on building SAS/AF applications, see *Guide to SAS/AF Applications Development*.

You can use the DISPLAY procedure to execute an application that runs in NODMS batch mode. Be aware that any SAS programming statements that you submit with the DISPLAY procedure through the SUBMIT block in SCL are not submitted for processing until PROC DISPLAY has executed.

If you use the SAS windowing environment, you can use the AF command to execute an application. SUBMIT blocks execute immediately when you use the AF command. You can use the AFA command to execute multiple applications concurrently.

Syntax: DISPLAY Procedure

PROC DISPLAY CATALOG=*libref.catalog.entry.type* <BATCH>;

PROC DISPLAY Statement

Executes a SAS/AF application.

Example: "Example: Executing a SAS/AF Application" on page 500

Syntax

PROC DISPLAY CATALOG=*libref.catalog.entry.type* <BATCH>;

Required Argument

CATALOG=*libref.catalog.entry.type*
specifies a four-level name for the catalog entry.

libref
specifies the SAS library where the catalog is stored.

catalog
specifies the name of the catalog.

entry
specifies the name of the entry.

type
specifies the entry's type, which is one of the following. For details, see the description of catalog entry types in the BUILD procedure in online Help.

- CBT
- FRAME
- HELP
- MENU
- PROGRAM
- SCL

Optional Argument

BATCH
runs PROGRAM and SCL entries in batch mode. If a PROGRAM entry contains a display, then it will not run, and you will receive the following error message:
`ERROR: Cannot allocate window.`

Restriction: PROC DISPLAY cannot pass arguments to a PROGRAM, a FRAME, or an SCL entry.

Example: Executing a SAS/AF Application

Features: PROC DISPLAY statement
CATALOG = argument

Details

Suppose that your company has developed a SAS/AF application that compiles statistics from an invoice database. Further, suppose that this application is stored in the SASUSER library, as a FRAME entry in a catalog named INVOICES.WIDGETS. You can execute this application using the following SAS code.

```
proc display catalog=sasuser.invoices.widgets.frame;
run;
```

Chapter 18

EXPORT Procedure

Overview: Export Procedure

PROC EXPORT reads data from a SAS data set and writes it to an external data source. External data sources include delimited files and JMP files. If you have a license for SAS/ACCESS Interface to PC Files, you can also export to such files as Microsoft Access Databases, Microsoft Excel Workbooks, DBF files, and Lotus spreadsheets. In delimited files, a delimiter such as a blank, comma, or tab separates columns of data values.

The EXPORT procedure uses one of these methods to export data:

- generated DATA step code
- generated SAS/ACCESS code
- translation engines

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.

You can also use the Export Wizard to guide you through the steps to export a SAS data set. The Export Wizard can generate EXPORT procedure statements, which you can save to a file for subsequent use. To open the Export Wizard, from the SAS windowing environment, select **File** ⇨ **Export Data**. For more information about the Export Wizard, see the Base SAS online Help and documentation.

Syntax: EXPORT Procedure

Restriction: The EXPORT procedure is available for the following operating environments:

- Windows
- UNIX

PROC EXPORT DATA=<*libref.*>*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;
statements for exporting to JMP files
DBENCODING=*12-byte SAS encoding-value* ;
FMTLIB=<*libref.*>*format-catalog*;
META=*libref.member-data-set*;

Statement	Task	Example
"PROC EXPORT Statement"	Export SAS data sets to an external data file	Ex. 1, Ex. 2
"DBENCODING Statement"	Indicate the encoding used to save data in JMP files	
"DELIMITER Statement"	Specify the delimiter to separate columns of data in the delimited output file	Ex. 1
"FMTLIB Statement"	Write SAS format values defined in the format catalog to the JMP file for the value labels	
"META Statement"	Write SAS metadata information to the JMP file	

PROC EXPORT Statement

Exports SAS data sets to an external data file.

Syntax

PROC EXPORT DATA=<*libref.*>*SAS data set*<(*SAS data set options*)>
OUTFILE="*filename*" | OUTTABLE="*tablename*"
<DBMS=*identifier*> <REPLACE> <LABEL>;

Required Arguments

DATA=<*libref.*>*SAS data set*

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.

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.

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.

Examples:

"Example 1: Exporting to a Delimited External Data Source" on page 506

"Example 2: Exporting a Subset of Observations to a CSV File" on page 510

OUTFILE="*filename*"

specifies the complete path and filename or a fileref for the output PC file, spreadsheet, or delimited external file. If you specify a fileref, or if the complete path and filename do not include special characters (such as the backslash in a path), lowercase characters, or spaces, you can omit the quotation marks. A fileref is a SAS name that is associated with the physical location of a file. To assign a fileref, use the FILENAME statement.

Alias: FILE

Restriction: 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.

See: *SAS/ACCESS Interface to PC Files: Reference* for more information about PC file formats.

Examples:

"Example 1: Exporting to a Delimited External Data Source" on page 506

"Example 2: Exporting a Subset of Observations to a CSV File" on page 510

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.

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 data to export. To export to a DBMS table, you must specify the DBMS option by using a valid database identifier. For DBMS=DLM, the default delimiter character is a space. However, you can use DELIMITER='char'.

The following values are valid for the DBMS identifier.

Table 18.1 *DBMS Identifiers Supported in Base SAS*

Identifier	Output Data Source	Extension
CSV	Delimited file (comma-separated values)	.csv
DLM	Delimited file (default delimiter is a blank)	
JMP	JMP files	.jmp
TAB	Delimited file (tab-delimited values)	.txt

Restriction: The availability of an output external data source depends on these conditions:

- the operating environment, and in some cases the platform, as specified in the previous table.
- 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.

See: *SAS/ACCESS Interface to PC Files: Reference* for a list of additional DBMS identifiers when using SAS/ACCESS Interface to PC Files.

Example: "Example 1: Exporting to a Delimited External Data Source" on page 506

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 510

(*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 510

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-byte SAS encoding-value* ;

Required Argument

12-byte 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: The DELIMTER statement is valid only when DBMS=DLM.

Example: "Example 1: Exporting to a Delimited External Data Source" on page 506

Syntax

DELIMITER=*char* | '*nn*'x;

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 character or as a hexadecimal value. For example, if you want columns of data to be separated by an ampersand, specify DELIMITER='&'.

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.

Interaction: The META statement is valid only when DBMS=JMP.

Syntax

META=*libref.member-data-set*;

Required Argument

libref.member-data-set
specifies the data set containing the metadata information to be written to the JMP file.

Examples: EXPORT Procedure

Example 1: Exporting to a Delimited External Data Source

Features: PROC EXPORT statement options
DATA=
DBMS=
OUTFILE=
REPLACE
DELIMITER=

Details

This example exports the SASHELP.CLASS data set to a delimited external file, and the PROC PRINT of this example is shown below.

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

Note that the filename does not contain an extension. DBMS=DLM specifies that the output file is a delimited file. The DELIMITER option specifies that an & (ampersand) will delimit data fields in the output file.

Program

```
proc export data=sashelp.class
   outfile="c:\myfiles\class"
   dbms=dlm;
   delimiter='&';
 run;
```

The SAS Log

The SAS log displays this information about the successful export, including the generated SAS DATA step.

```
1    proc export data=sashelp.class outfile="c:\myfiles\class" dbms=dlm;
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

NOTE: 20 records were written to the file 'c:\myfiles\class'.
      The minimum record length was 17.
      The maximum record length was 26.
NOTE: There were 19 observations read from the data set SASHELP.CLASS.
NOTE: DATA statement used (Total process time):
      real time           0.91 seconds
      cpu time            0.04 seconds

19 records created in c:\myfiles\class from SASHELP.CLASS.

NOTE: "c:\myfiles\class" file was successfully created.
NOTE: PROCEDURE EXPORT used (Total process time):
      real time           9.07 seconds
      cpu time            0.15 seconds
```

Output

The EXPORT procedure produces this external file:

Output 18.1 *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. 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.

Program

```
proc export data=sashelp.class (where=(sex='F'))
     outfile="c:\myfiles\Femalelist.csv"
     dbms=csv
     replace;
run;
```

Chapter 19

FCMP Procedure

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.

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 COMPILE
- PROC COMPUTAB
- PROC GA
- PROC GENMOD
- PROC MCMC
- PROC MODEL
- PROC NLIN
- PROC NLMIXED
- PROC NLP

- PROC PHREG
- PROC REPORT COMPUTE blocks
- Risk Dimensions procedures
- PROC SIMILARITY
- PROC SQL (functions with array arguments are not supported)

For more information about using PROC FCMP with ODS, see the *SAS Output Delivery System: User's Guide*.

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 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 that has no return value. 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 =
sasuser.MySubs.MathFncs;

   function day_date(indate, type $);
      if type = "DAYS" then wkday = weekday(indate);
      if type = "YEARS" then wkday = weekday(indate*365);
      return(wkday);
   endsub;

   subroutine inverse(in, inv);
      outargs inv;
      if in = 0 then inv = .;
      else inv = 1/in;
   endsub;

run;
```

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 *sasuser.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.

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 sasuser.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 will 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 *sasuser.funcs.trial*. The program calls the function, passing the variable values for *start* and *today*, and returns the result in the variable SD.

```
options pageno=1 nodate;

proc fcmp outlib=sasuser.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=sasuser.funcs;
data _null_;
   start = '15Feb2006'd;
   today = '27Mar2006'd;
   sd = study_day(start, today);
   put sd=;
run;
```

Log 19.1 *Output from the STUDY_DAY User-Defined Function*

```
sd=41
```

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-n);
   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 on an OUTARGS statement. The syntax of a CALL routine declaration is as follows:

```
subroutine
name(argument-1, ..., argument-n);
   outargs out-argument-1, ..., out-argument-N;
   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 will modify 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 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. See “PROC FCMP and DATA Step Differences” on page 528 for more information.

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. PROC FORMAT describes the process. For more information, see PROC FORMAT, "Using a Function to Format Values," in the *Base SAS Procedures Guide*.

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. See "PROC FCMP and DATA Step Differences" on page 528 for a list of statements and the differences.

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 | *variables* | *constants* | (*initial-values)*>;
ATTRIB *variables* <FORMAT=*format-name* LABEL='*label*' LENGTH=*length*>;
DELETEFUNC *function-name*;
DELETESUBR *subroutine-name*;
FUNCTION *function-name(argument-1, ..., argument-n)* <VARARGS> <$> <*length*> <KIND | GROUP='*string*'>;
LABEL *variable='label'*;
LISTFUNC *function-name*;
LISTSUBR *subroutine-name*;
STRUCT *structure-name variable*;
SUBROUTINE *subroutine-name (argument-1, ..., argument-n)* <VARARGS> <LABEL='*label*'> <KIND | GROUP='*string*'>;
OUTARGS *out-argument-1, ..., out-argument-n*;

Statement	Task	Example
"PROC FCMP Statement"	Create, test, and store SAS functions for use by other SAS procedures	Ex. 1, Ex. 2, Ex. 8, Ex. 7
"ABORT Statement"	Terminate the execution of the current DATA step, SAS job, or SAS session	
"ARRAY Statement"	Associate a name with a list of variables and constants	Ex. 8
"ATTRIB Statement"	Specify format, label, and length information for a variable	
"DELETEFUNC Statement"	Delete a function from the function library that is specified in the OUTLIB option	
"DELETESUBR Statement"	Delete a subroutine from the function library that is specified in the OUTLIB option	
"FUNCTION Statement"	Return changed variable values	Ex. 1, Ex. 2, Ex. 7
"LABEL Statement"	Specify a label for variables	
"LISTFUNC Statement"	Write the source code of a function in the SAS listing	
"LISTSUBR Statement"	Write the source code for a subroutine in the SAS listing	

Statement	Task	Example
"STRUCT Statement"	Declare (create) structure types	
"SUBROUTINE Statement"	Declare (create) independent computational blocks of code	Ex. 8
"OUTARGS Statement"	(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 544
"Example 2: Creating a CALL Routine and a Function" on page 545
"Example 7: Using GTL with User-Defined Functions" on page 549
"Example 8: Standardizing Each Row of a Data Set" on page 552

Syntax

PROC FCMP *options*;

Optional Arguments

ENCRYPT
HIDE
specifies to encode the source code in a data set.

FLOW
specifies printing a message for each statement in a program as it is executed. This option produces extensive output.

LIBRARY | INLIB=*library.dataset*
LIBRARY | INLIB=(*library-1.dataset library-2.dataset ... library-n.dataset)*
LIBRARY | INLIB=*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).

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 INLIB= 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=sasuser.exsubs;
proc fcmp library=(sasuser.exsubs work.examples);
proc fcmp library=lib1-lib10;
```

LIST
specifies that both the LISTSOURCE and the 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 the way that the expression is actually calculated, because intermediate results for common subexpressions can be re-used. However, the expressions are printed in expanded form by the LISTPROG option. To see how the expression is actually evaluated, refer to the listing from the LISTCODE option.

LISTSOURCE
specifies that source code statements for the program be printed.

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=sasuser.fcmpsubs.pkt1;

proc fcmp outlib=sasuser.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.

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.

Tip: Specifying TRACE is equivalent to specifying FLOW, PRINT, and PRINTALL.

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 552

Syntax

ARRAY *array-name*[*dimensions*] </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

/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.

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.

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. The following 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 curly braces { }.
- The ARRAY statement in PROC FCMP does not support lower-bound specifications.
- You can use a maximum of six dimensions for an array.

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 have the CALL routine modify the values of the array, you must specify the name for the array argument in an OUTARGS statement in the CALL routine.

Example

The following are 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;`

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

The following are 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.;
  ```

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 544
"Example 2: Creating a CALL Routine and a Function" on page 545
"Example 3: Using Numeric Data in the FUNCTION Statement" on page 547
"Example 4: Using Character Data in the FUNCTION Statement" on page 548
"Example 5: Using Variable Arguments with an Array" on page 548
"Example 7: Using GTL with User-Defined Functions" on page 549

Syntax

FUNCTION *function-name*(*argument-1*, ..., *argument-n*) <VARARGS> <$> <*length*> <KIND | GROUP='*string*'>;

... *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. 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.

See: "Example 5: Using Variable Arguments with an Array" on page 548

$
specifies that the function returns a character value. If $ is not specified, the function returns a numeric value.

length
specifies the length of a character value.

Default: 8

KIND='*string*'
GROUP='*string*'
specifies a collection of items that have specific attributes.

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

The following are examples of the LABEL statement:

- `label date='Maturity Date';`
- `label bignum='Very very large numeric value';`

LISTFUNC Statement

Causes the source code for a function to be written to the SAS listing.

Syntax

LISTFUNC *function-name*;

Required Argument

function-name
specifies the name of the function for which source code is written to the SAS listing.

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 a CALL Routine and a Function" on page 545
"Example 8: Standardizing Each Row of a Data Set" on page 552
"Example 6: Using the SUBROUTINE Statement with a CALL Statement" on page 549

Syntax

OUTARGS *out-argument-1*, ..., *out-argument-n*;

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 527 for an example of how to use the OUTARGS statement in a subroutine.

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

The following 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 552
"Example 2: Creating a CALL Routine and a Function" on page 545

Syntax

SUBROUTINE *subroutine-name* (*argument-1*, ..., *argument-n*) <VARARGS> <KIND | GROUP='*string*'>;

OUTARGS *out-argument-1*, ..., *out-argument-n*;

... *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. 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.

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.

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 curly 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 522.

Data Set Input and Output

PROC FCMP does not support the DATA and the OUTPUT statements for creating and writing to an output data set. 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 are 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 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 superscripts. 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_;**.

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. The following is an example:

```
SELECT;
   WHEN(expression-1)
statement-1;
   statement-2;
   WHEN (expression-2)
statement-3;
   statement-4;
END;
```

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.

The FCmp Function Editor

The FCmp Function Editor is an application for traversing packages of functions and is built into the SAS Explorer. You can access the FCmp Function Editor from the Solutions menu of an interactive SAS session. For more information, see "Introduction to the FCmp Function Editor" on page 597.

Computing Implicit Values of a Function

PROC FCMP uses a SOLVE function for computing implicit values of a function. For more information, see "SOLVE Function" on page 590.

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.

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[*] $);
```

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-dimN-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 the %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=sasuser.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=sasuser.funcs;
data _null_;
   x=99;
   call subA();
```

```
   put 'In DATA step: ' x=;
run;
```

SAS writes the following output to the log:

Log 19.2 *Output from Local Variables in Different Routines Having 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 **P(*n*, *k*) = *n*! /(*n* - *k*)!** permutations 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=sasuser.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=sasuser.funcs;
data _null_;
   call allpermk(5, 3);
run;
```

Log 19.3 *Log Output from Recursion Example*

```
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 Transversal

Overview of Directory Transversal

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 Transversal 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=sasuser.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; yet 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;
         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=sasuser.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;
```

Log 19.4 *Output 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.

Syntax of the CMPLIB= System Option

The syntax for the CMPLIB= option has the following form:

OPTIONS CMPLIB = *library*

OPTIONS CMPLIB = *(library–1, ..., library-n)*

OPTIONS CMPLIB = *list-1, ..., list-n*

where

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.

Example 1: Setting the CMPLIB= System Option

The following example shows how to set the CMPLIB= system option.

- `OPTIONS cmplib = sasuser.funcs;`
- `OPTIONS cmplib = (sasuser.funcs work.functions mycat.funcs);`
- `OPTIONS cmplib =( sasuser.func1 - sasuser.func10);`

Example 2: Compiling and Using Functions

In the following example, PROC FCMP compiles the SIMPLE function and stores it in the **`sasuser.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.

```
proc fcmp outlib = sasuser.models.yval;
   function simple(a,b,x);
      y=a+b*x;
      return(y);
   endsub;
run;

options cmplib = sasuser.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;
```

Display 19.1 *Compiling and Using Functions: Output 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.

Display 19.2 *Compiling and Using Functions: Output 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

Display 19.3 *Compiling and Using Functions: Output 3*

The SAS System

The MODEL Procedure

Nonlinear OLS Summary of Residual Errors

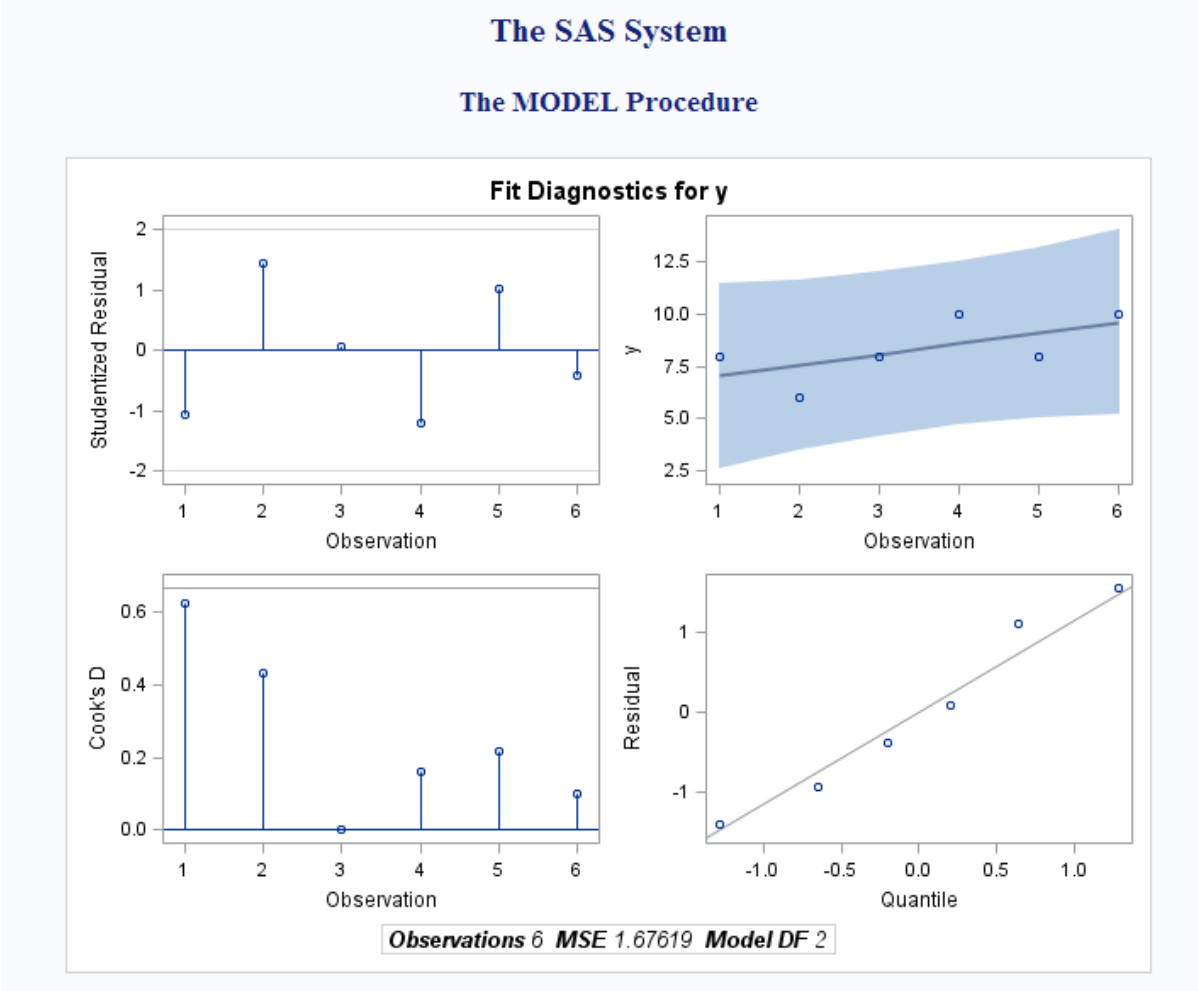
Equation	DF Model	DF Error	SSE	MSE	Root MSE	R-Square	Adj R-Sq
y	2	4	6.7048	1.6762	1.2947	0.4084	0.2605

Nonlinear OLS Parameter Estimates

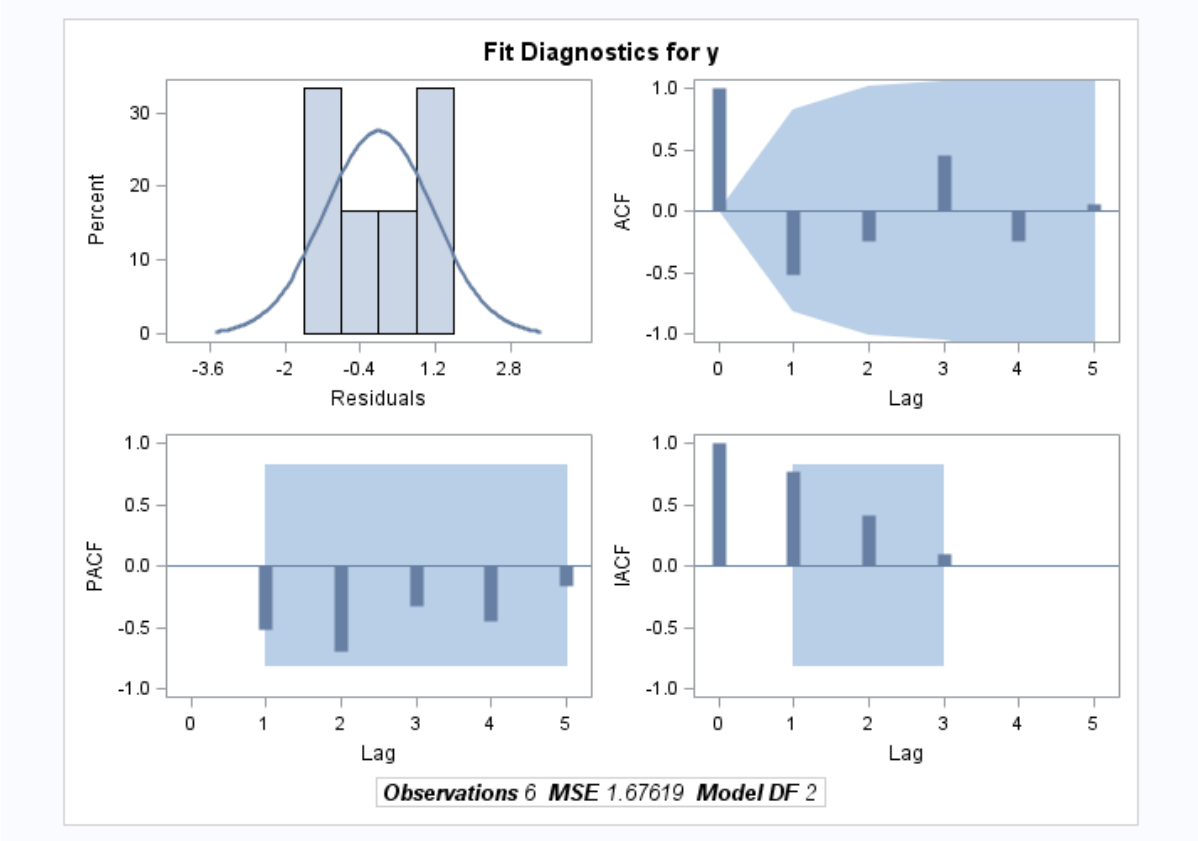
Parameter	Estimate	Approx Std Err	t Value	Approx Pr > |t|
a	6.533333	1.2053	5.42	0.0056
b	0.514286	0.3095	1.66	0.1719

Number of Observations		Statistics for System	
Used	6	Objective	1.1175
Missing	0	Objective*N	6.7048

Display 19.4 *Compiling and Using Functions: Output 4*



Display 19.5 *Compiling and Using Functions: Output 5*



For information about PROC MODEL, see the *SAS/ETS User's Guide*.

PROC FCMP and DATA Step Component Objects

SAS provides two predefined component objects for use with PROC FCMP and the DATA step: the hash object and the hash iterator object. These objects enable you to quickly and efficiently store, search, and retrieve data based on lookup keys.

The component objects are data elements that consist of attributes, methods, and operators. Attributes are the properties that specify the information that is associated with an object. Methods define the operations that an object can perform. For component objects, operators provide special functionality. You use the DATA step object dot notation to access the component object's attributes and methods.

The following methods are available:

- CHECK
- DEFINEDATA
- DEFINEDONE
- DEFINEKEY
- DELETE
- FIND

- FIRST
- LAST
- NEXT
- NUM_ITEMS
- PREV

For more information about hash and hash iterator component objects, see *SAS Component Objects: Reference*.

Examples: FCMP Procedure

Example 1: Creating a Function and Calling the Function from a DATA Step

Features: PROC FCMP statement option
OUTLIB=
DATA step

This example shows how to compute a study day during a drug trial by creating a function in FCMP and using that function in a DATA step.

Program

```
proc fcmp outlib=sasuser.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=sasuser.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 Sasuser.Funcs.

```
proc fcmp outlib=sasuser.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=sasuser.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 sasuser.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;
```

Output: Log

Log 19.5 *Output from Creating a Function and Calling the Function from a DATA Step*

```
sd=41
```

Example 2: Creating a CALL Routine and a Function

Features: PROC FCMP statement option
OUTLIB=
OUTARGS statement

This example shows how to use PROC FCMP to create and store CALL routines and functions.

Program

```
proc fcmp outlib =
sasuser.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);
   endsub;

run;
```

Program Description

Specify the entry where the function package information is saved. The package is a three-level name.

```
proc fcmp outlib =
sasuser.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 will be 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;

      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;
```

The RETURN statement returns the value of the GARKHPRC function.

```
      return (garkhprc);
   endsub;
```

Execute the FCMP procedure. The RUN statement executes the FCMP procedure.

```
run;
```

Output: Log

Log 19.6 *Output from Creating a CALL Routine and a Function*

```
NOTE: Function garkhprc saved to sasuser.exsubs.pkt1.
NOTE: Function calc_years saved to sasuser.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 n=0 then inv=.;
      else inv=1/in;
      return(inv);
   endsub;
run;
```

Example 4: Using Character Data in 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');
   endsub;
run;

data _null_;
   spanish=test('yes');
   put spanish=;
run;
```

Output: Log

Log 19.7 *Output from Using Character Data in the Function Statement*

```
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);
```

Program

```
options cmplib=sasuser.funcs;

proc fcmp outlib=sasuser.funcs.temp;
function summation (b[*]) varargs;
    total = 0;
    do i = 1 to dim(b);
        total = total + b[i];
    end;
```

```
return(total);
endsub;
sum=summation(1,2,3,4,5);
   put sum=;
run;
```

Example 6: Using the SUBROUTINE Statement with a CALL Statement

Details

The following 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=sasuser.funcs.temp;
subroutine inverse(in, inv) group="generic";
   outargs inv;
   if in=0 then inv=.;
   else inv=1/in;
endsub;

options cmplib=sasuser.funcs;
data _null_;
   x = 5;
   call inverse(x, y);
   put x= y=;
run;
```

Output: Log

Log 19.8 *Output from Using the SUBROUTINE Statement in PROC FCMP*

```
x=5 y=0.2
```

Example 7: Using GTL with User-Defined Functions

Features: PROC FCMP functions
- OSCILLATE
- OSCILLATEBOUND

Other procedures
- PROC TEMPLATE
- PROC SGRENDER

The following example shows how to use functions in a GTL EVAL function.

Details

The following example 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=sasuser.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);
  endsub;

 function oscillateBound(x,amplitude);
       if amplitude le 0 then amp=1; else amp=amplitude;
       y=constant("e")**(-amp*x);
       return (y);
  endsub;
 run;


 options cmplib=sasuser.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;
        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=sasuser.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);
 endsub;
```

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);
 endsub;
run;
```

Create a data set called RANGE that will be used by PROC SGRENDER.

```
options cmplib=sasuser.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.

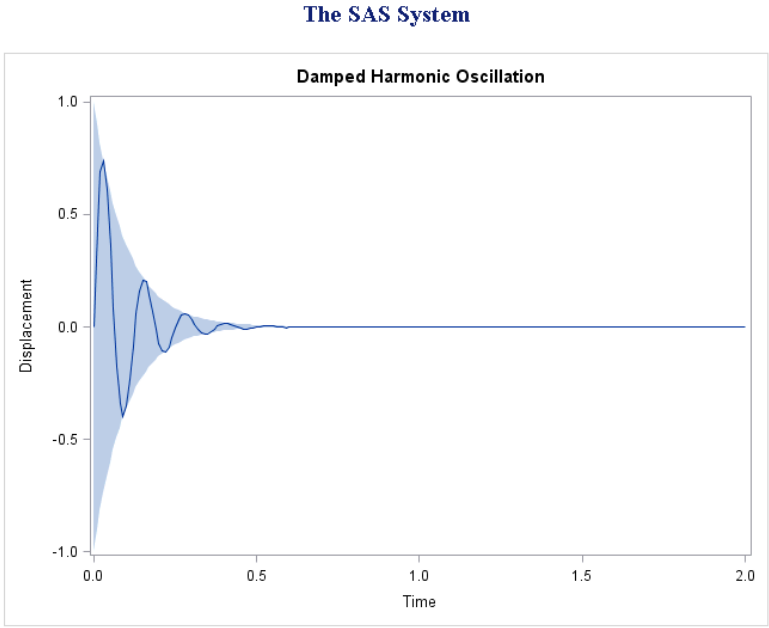
```
proc template ;
   define statgraph damping;
   dynamic X AMP FREQ;
   begingraph;
       entrytitle "Damped Harmonic Oscillation";
       layout overlay / yaxisopts=(label="Displacement");
          if (exists(X) and exists(AMP) and exists(FREQ))
                bandplot x=X  limitlower=eval(-oscillateBound(X,AMP))
                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: HTML

Output 19.1 *Output from Using GTL with User-Defined Functions*



Example 8: Standardizing Each Row of a Data Set

Features: PROC FCMP functions
RUN_MACRO
RUN_SASFILE
READ_ARRAY
WRITE_ARRAY

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] = ranuni(12345) * (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 = sasuser.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 cmplib = (sasuser.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] = ranuni(12345) * (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 = sasuser.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;
```

Execute the function for each row in the DATA step.

```
options cmplib = (sasuser.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: HTML

Output 19.2 *Output 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

Chapter 20

FCMP Special Functions and Call Routines

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

Reading Arrays and Writing Arrays to a Data Set

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.

CALL Routines for 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.

C Helper Functions and CALL Routines

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.

Functions for 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.

Special Purpose Functions

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).

Functions and CALL Routines by Category

Category	Language Elements	Description
Array	CALL DYNAMIC_ARRAY Routine (p. 563)	Enables an array that is declared within a function to change size in an efficient manner.
	READ_ARRAY Function (p. 583)	Reads data from a SAS data set into a PROC FCMP array variable.
	WRITE_ARRAY (p. 594)	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. 571)	Sets a pointer element of a structure to null.
	CALL STRUCTINDEX Routine (p. 571)	Enables you to access each structure element in an array of structures.
	ISNULL Function (p. 579)	Determines whether a pointer element of a structure is null.
Calling SAS Code from within Functions	RUN_MACRO Function (p. 585)	Executes a predefined SAS macro.
	RUN_SASFILE Function (p. 588)	Executes SAS code in a fileref that you specify.
Compute Implicit Values	SOLVE Function (p. 590)	Computes implicit values of a function.
Matrix Operations	CALL ADDMATRIX Routine (p. 560)	Performs an elementwise addition of two matrices or a matrix and a scalar.
	CALL CHOL Routine (p. 561)	Calculates the Cholesky decomposition for a given symmetric matrix.
	CALL DET Routine (p. 562)	Calculates the determinant of a specified matrix that should be square.
	CALL ELEMMULT Routine (p. 564)	Performs an elementwise multiplication of two matrices.
	CALL EXPMATRIX Routine (p. 565)	Returns a matrix etA given the input matrix A and a multiplier t.
	CALL FILLMATRIX Routine (p. 566)	Replaces all of the element values of the input matrix with the specified value.
	CALL IDENTITY Routine (p. 567)	Converts the input matrix to an identity matrix.

Category	Language Elements	Description
	CALL INV Routine (p. 568)	Calculates a matrix that is the inverse of the provided input matrix that should be a square, non-singular matrix.
	CALL MULT Routine (p. 569)	Calculates the multiplicative product of two input matrices.
	CALL POWER Routine (p. 570)	Raises a square matrix to a given scalar value.
	CALL SUBTRACTMATRIX Routine (p. 573)	Performs an element-wide subtraction of two matrices or a matrix and a scalar.
	CALL TRANSPOSE Routine (p. 574)	Returns the transpose of a matrix.
	CALL ZEROMATRIX Routine (p. 575)	Replaces all of the element values of the numeric input matrix with 0.
Special Purpose Functions	INVCDF Function (p. 576)	Computes the quantile from any distribution for which you have defined a cumulative distribution function (CDF).
	LIMMOMENT Function (p. 580)	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* x *n* (that is, X[*m*, *n*]) or a scalar.

Y

specifies an input matrix with dimensions *m* x *n* (that is, Y[*m*, *n*]) or a scalar.

Z

specifies an output matrix with dimensions *m* x *n* (that is, Z[*m*, *n*]), such that

Z = *X* + *Y*

Example

The following example uses the ADDMATRIX CALL routine:

```
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 20.1 *Output 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 will be a lower triangle matrix.

Syntax

CALL CHOL(*X*, *Y* <, *validate*>);

Required Arguments

X

specifies a symmetric positive-definite input matrix with dimensions *m* x *m* (that is, X[*m*, *m*]).

Y

specifies an output matrix with dimensions *m* x *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 will be 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 will be checked 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:

```
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;
```

SAS produces the following output:

Output 20.2 *Output 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* x *n* (that is, X[*m*, *m*]).

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). See the *SAS/IML User's Guide* for more information.

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 20.3 *Output 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-dim1–size*, ...,, *new-dimN-size*);

Required Arguments

array-name
specifies the name of a temporary array.

new-dim-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 datalen;
      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/datalen;

   return(avedev);
endsub;
run;
```

CALL ELEMMULT Routine

Performs an elementwise multiplication of two matrices.

Category: Matrix Operations

Requirement: All input and output matrices must have the same dimensions.

Syntax

CALL ELEMMULT(*X*, *Y*, *Z*);

Required Arguments

X
specifies an input matrix with dimensions *m* x *n* (that is, X[*m*, *n*]).

Y
specifies an input matrix with dimensions *m* x *n* (that is, Y[*m*, *n*]).

Z
specifies an output matrix with dimensions *m* x *n* (that is, Z[*m*, *n*]).

Example

The following example uses the ELEMMULT 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 20.4 *Output from the ELEMMULT 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* x *m* (that is, X[*m*, *m*]).

t

specifies a double scalar value.

Y

specifies an output matrix with dimensions *m* x *m* (that is, Y[*m*, *m*]), such that

$Y = \varepsilon^{tX}$

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. Refer to the EXPMATRIX documentation in the *SAS/IML User's Guide* for more information.

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 20.5 *Output 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 will fill 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 20.6 *Output 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 will be set to 1, and the rest of the values will be set to 0.

Syntax

CALL IDENTITY(*X*);

Required Argument

X
specifies an input matrix with dimensions *m* x *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 20.7 *Output 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* x *m* (that is, X[*m*, *m*]).

Y

specifies an output matrix with dimensions *m* x *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 20.8 *Output 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* x *n* (that is, X[*m*, *n*]).

Y

specifies an input matrix with dimensions *n* x *p* (that is, Y[*n*, *p*]).

Z

specifies an output matrix with dimensions *m* x *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 20.9 *Output 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* x *m* (that is, X[*m*, *m*]).

a
specifies an integer scalar value (power).

Y
specifies an output matrix with dimensions *m* x *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. See the *SAS/IML User's Guide* for more information.

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 20.10 *Output 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 that the same LINKLIST structure that is described in "Example 1: Generating a Linked List" on page 579 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(*struct_array*, *index*, *struct_element*);

Required Arguments

struct_array
specifies an array.

index
is a 1–based index as used in most SAS arrays.

struct_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 20.11 *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* x *n* (that is, X[*m*, *n*]) or a scalar.

Y

specifies an input matrix with dimensions *m* x *n* (that is, Y[*m*, *n*]) or a scalar.

Z

specifies an output matrix with dimensions *m* x *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 20.12 *Output 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* x *n* (that is, X[*m*, *n*]).

Y

specifies an output matrix with dimensions *n* x *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 20.13 *Output 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 20.14 *Output 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; <parameters>)*, then the INVCDF function returns a quantile *q* that satisfies *F(q;<parameters>) = p*. In other words, $q=F^{-1}(p)$.

Syntax

quantile=**INVCDF**('*CDF-function-name*', *options-array*, *cumulative-probability*, *parameter-1, parameter-2, ..., parameter-n*);

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, …, parameter-n);

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). The default for *initial-value* is 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. The default for *desired-accuracy* is 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,∞). Any other value indicates a support over the entire real line, that is (-∞,∞). The default for *domain-type* is 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. This must be in the range [0,1).

parameters
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:

```
cumulative-probability = CDF-function-name(quantile,<parameters>)
```

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. The following is an example of an options array:

```
array opts[4] initial epsilon support (1.5 1.0e-6 0);
```

where

```
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 can only 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 the 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;
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 20.15 *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, parameter-n*);

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, …, parameter-n);

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 employed 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*. The default for *desired-accuracy* is 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. The default for *initial-step-size* is 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. The default value for *maximum-iterations* is 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. This value must be in the range [1,10].

limit
specifies the upper limit that is used to compute the desired limited moment. This value must be greater than 0.

parameters
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;θ)* and cumulative distribution function *F(x;θ)*, 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. The following is an example of an options array:

```
array opts[4] epsilon initial maxiter (1.0e-5 1 6);
```

where

`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 the 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;
```

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* <, '*col_name_1*', ..., '*col_name_n*'>);

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 will be 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

col_name
specifies optional names for the specific columns of the data set that will be read.

If specified, *col_name* must be a literal string enclosed in quotation marks. *col_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 will be 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.

```
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 20.16 *Output 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_n*>);

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;
```

Output 20.17 *Output from Executing a Predefined Macro with PROC FCMP*

```
p=4.6
```

Example 2: 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, 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 20.18 *Output for Calling a DATA Step within a DATA Step*

```
                    The SAS System
                                                1

                             Wage       Wage
        Obs      Name      Category     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

                             Wage       Wage
         Obs     Name      Category     Rate

          1      Jason     Hourly        200
          2      Paul      Salaried     4000
```

```
                    The SAS System
                                                3

                             Wage       Wage
         Obs     Name      Category     Rate

          1      Kevin     Salaried     5500
          2      Amber     Hourly        150
          3      Tina      Salaried    13000
```

```
                    The SAS System
                                                 4

                             Wage       Wage
         Obs     Name      Category     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-n*>);

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 will be 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.

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-n*);

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. When 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,
..., argument-n)
```

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.

The following is an example of an options array:

```
array opts[5] initial abconv relconv maxiter (.5 .001 1.0e-6 100);
```

where

- **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 20.19 *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;

   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 20.20 *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

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 will be indexed as [row, column].

Syntax

rc = **WRITE_ARRAY**(*data_set_name*, *array_variable* <, '*col_name_1*', ..., '*col_name_n*'>);

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 will be 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 will be written to *data_set_name*.

Optional Argument

col_name
specifies optional names for the columns of the data set that will be created.

If specified, *col_name* must be a literal string enclosed in quotation marks. *col_name* cannot be a PROC FCMP variable. If column names are not specified, the column name will be 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 20.21 *Output 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 20.22 *Output 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
```

Chapter 21

FCmp Function Editor

Introduction to the FCmp Function Editor

SAS language functions and CALL routines that are created with PROC FCMP are stored in SAS data sets that are contained in package declarations. Each package declaration contains one or more functions or CALL routines. The FCmp Function Editor displays all of the functions and CALL routines that are included in a package.

With the FCmp Function Editor, you can view functions in a package declaration as well as create new functions. You can add these new functions to an existing package, or create a new package declaration.

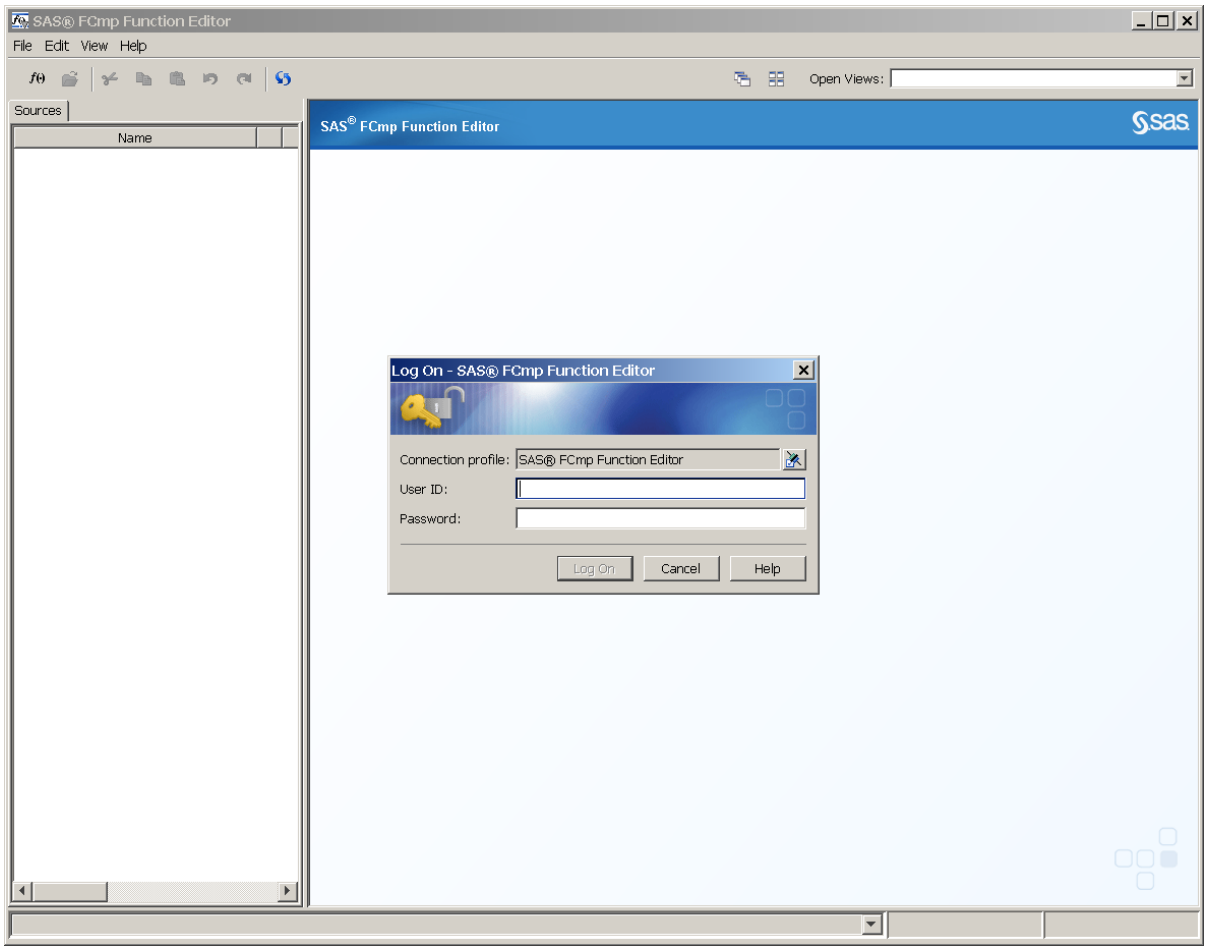
Open the FCmp Function Editor

If you are working in the Windows operating environment and SAS is installed locally on your computer, the sign-on dialog box is bypassed because Windows supports single sign-on functionality.

If you are not working in the Windows operating environment, or if you do not have SAS installed locally, then you will be prompted for your authorization credentials, which are your user ID and password.

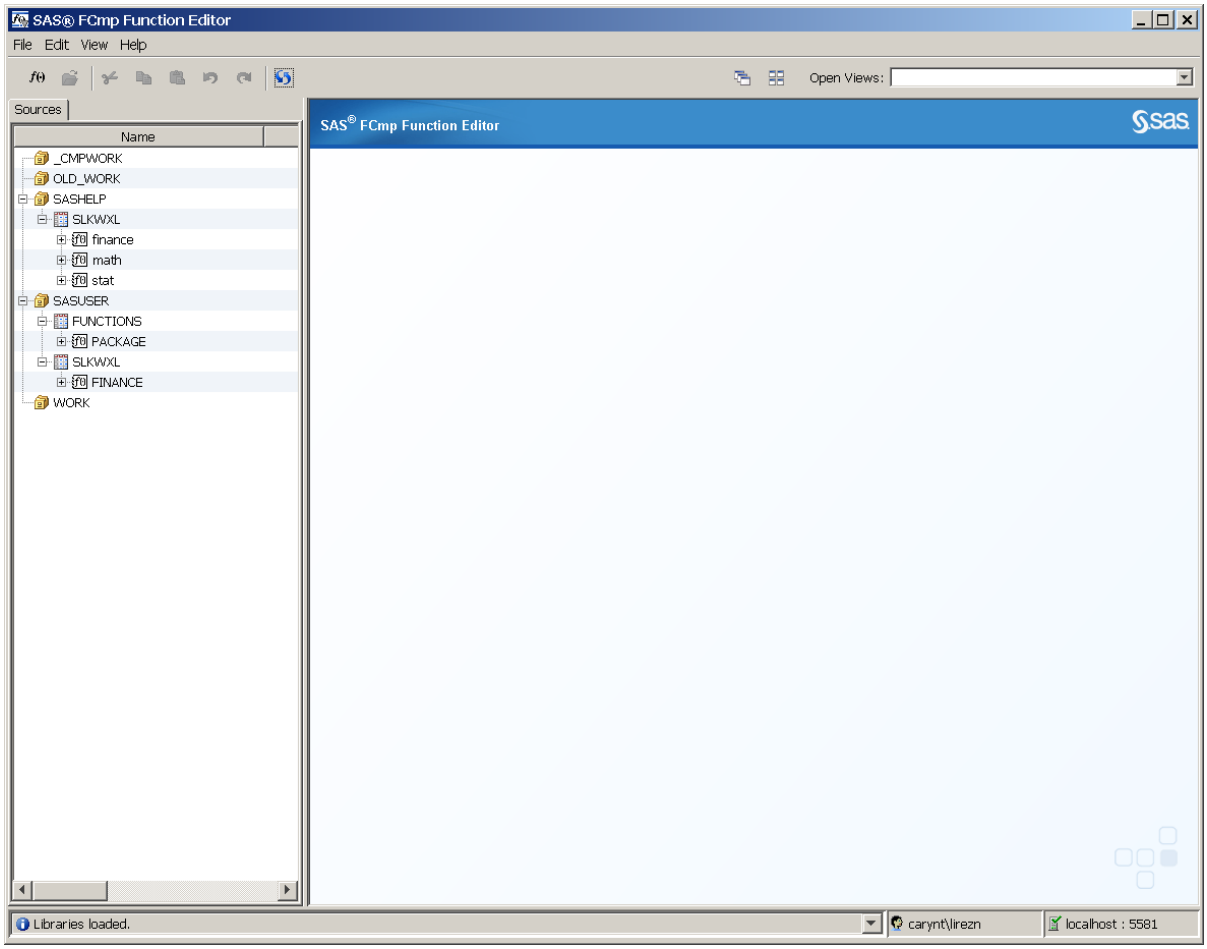
To open the FCmp Function Editor, select **Solutions** ⇨ **Analysis** ⇨ **FCmp Function Editor** from the menu in your SAS session. The following dialog box appears:

Display 21.1 *Initial Dialog Box for the FCmp Function Editor*



After you enter your user ID and password and click **Log On**, SAS establishes a connection to a port. A window that displays your libraries appears:

Display 21.2 *The FCmp Function Editor with Libraries Displayed*



In the window above, you can see that the left pane lists the functions that are in the SASHELP and SASUSER libraries. The WORK library is empty. You cannot access the WORK library directly from a spawning SAS session. The FCmp Function Editor remaps the WORK library from the spawning SAS session to the location of OLD_WORK so that you can access the contents of WORK from OLD_WORK.

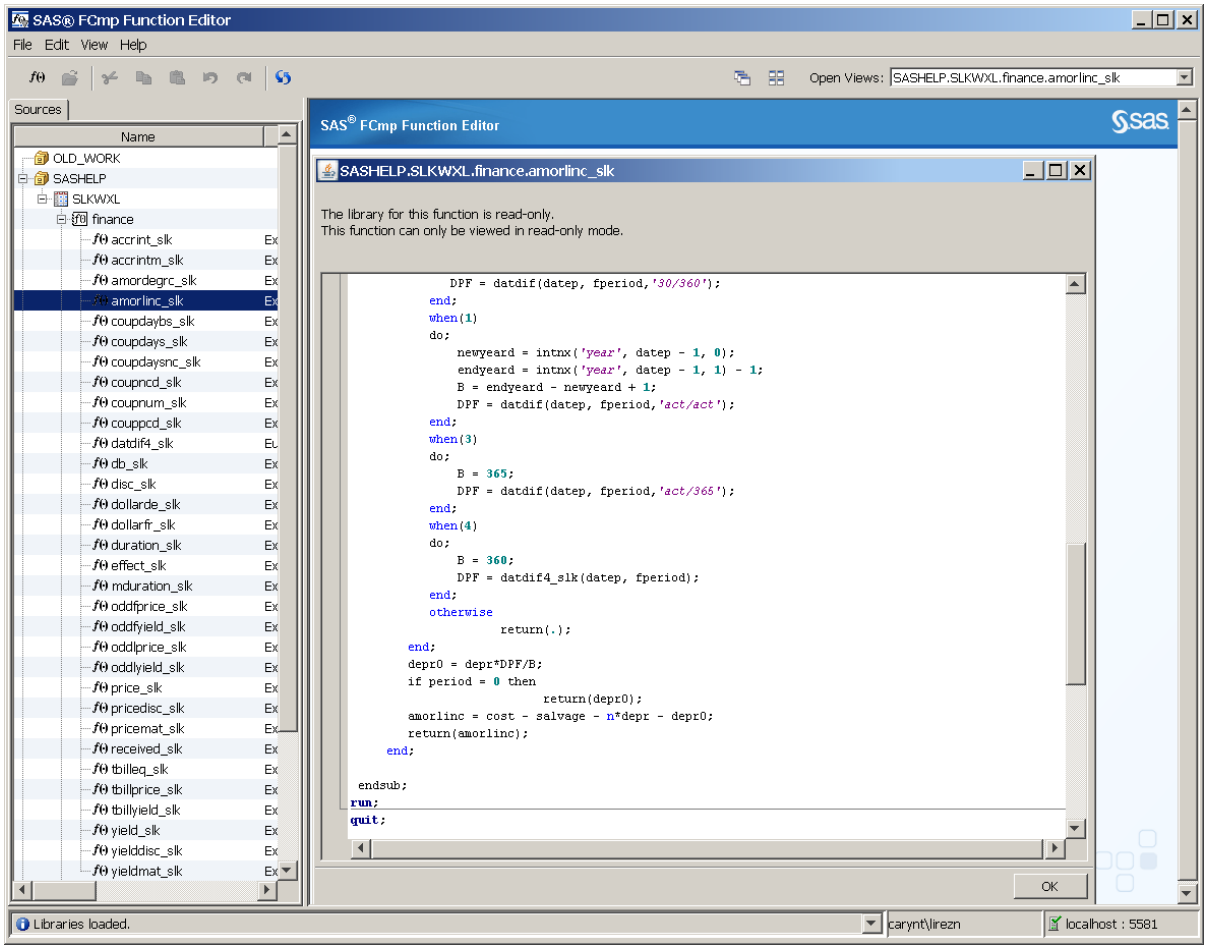
Working with Existing Functions

Open a Function

To open a function, select a library from the left pane, expand the library, and drill down until a list of functions appears. Double-click the name of the function that you want to open.

If you open a function from a read-only library, a window similar to the following appears:

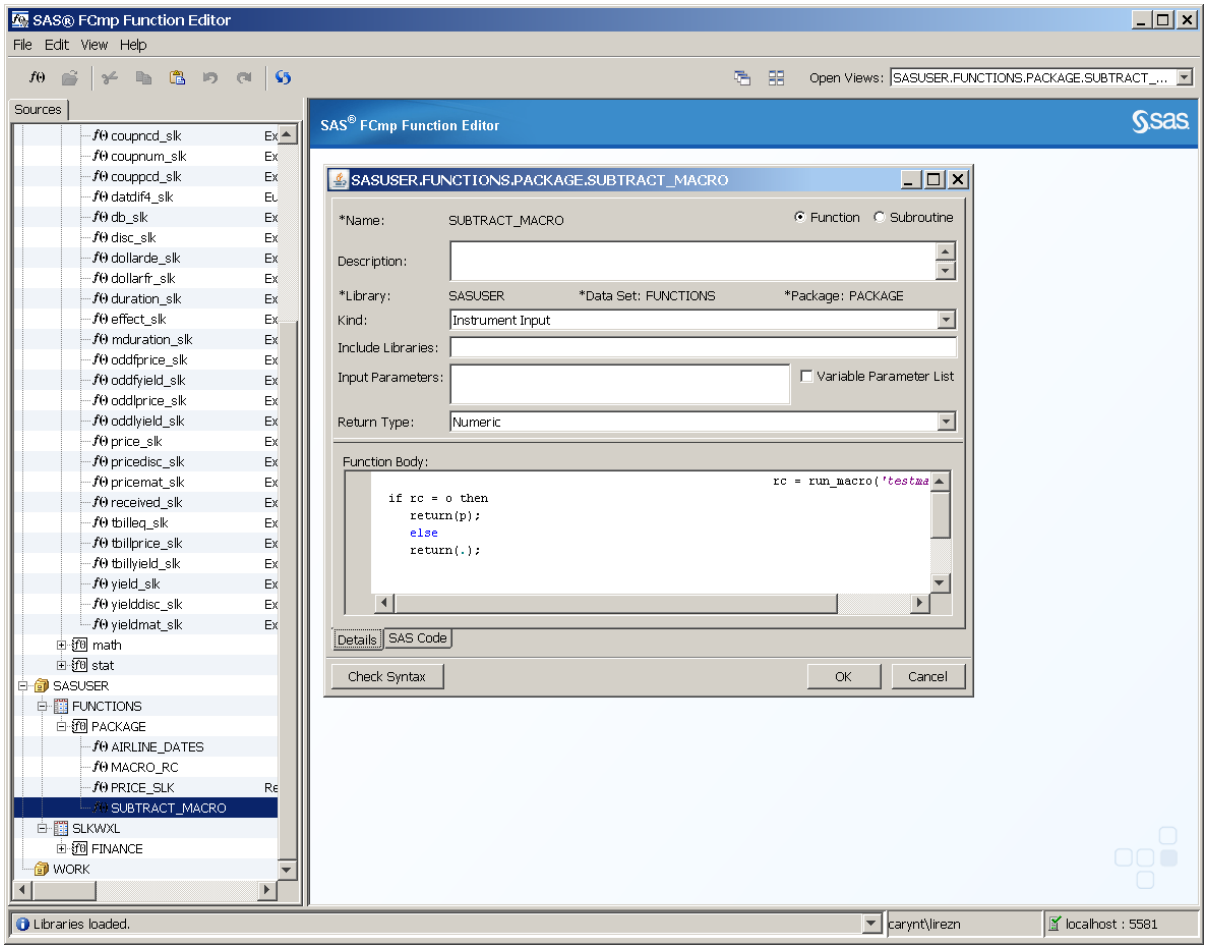
Display 21.3 *A Function in a Library That Has Read-Only Access*



In the window above, the AMORLINC_SLK function is selected from the read-only SASHELP library. Use the scroll bar to scroll to the top of the function.

If you open a function from a library to which you have write access, a window similar to the following appears:

Display 21.4 *A Function in a Library That Has Write Access*



In the window above, SUBTRACT_MACRO is selected from the write-enabled SASUSER library.

You can see that there is a difference in the windows that appear depending on whether the library has read-only access or write access. If the library has write access, you can enter information in the top section of the window that you are viewing. These fields are the same fields that you use when you create a new function. For a description of the fields, see "Creating a New Function" on page 604.

Opening Multiple Functions

Opening multiple functions results in multiple windows being opened. For example, if you open a second function, a second window appears that shows the code for that function.

The upper right corner of the FCmp Function Editor contains a field called **Open Views**. Click the arrow to list the functions that are open. When you select a function, the window for that function is brought to the foreground.

Two icons that you can use to alter the display of your functions are located to the left of the **Open Views** field:

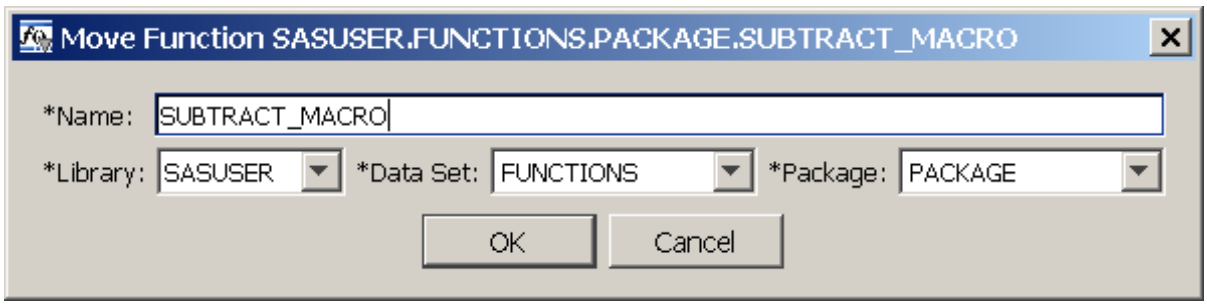
cascades the display of the functions that are open.

arranges the functions to display side by side.

Move a Function

You can move a function to a different library, data set, or package. To move a function, select a function in the left pane. Right-click the function, and select **Move** from the menu. The following dialog box appears:

Display 21.5 *The Move Function Dialog Box*



In the Move Function dialog box, you can perform the following tasks:

- enter a new name for the function
- select a library into which the function is to be moved
- enter a new data set name
- enter a package name

The descriptions of the fields in the Move Function dialog box are listed below:

Name
specifies the new name for the function.

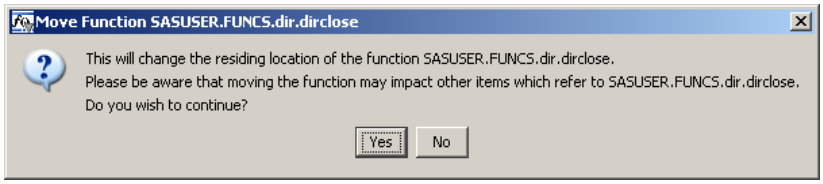
Library
specifies the library that will contain the function that you move. Use the menu in the **Library** field to select a library.

Data Set
specifies the data set that will contain the function that you move. Enter the name of the data set, or click the down arrow in the **Data Set** field to select a data set. If you do not choose a data set, then the value in this field defaults to Functions.

Package
specifies the name of the package that will contain the new function that you move. Enter the name of the package, or click the down arrow in the **Package** field to select a package. If you do not choose a package, then the value in this field defaults to PACKAGE.

When you click **OK**, the following dialog box appears, cautioning you about the move:

Display 21.6 *The Move Function Confirmation Dialog Box*



CAUTION:

Other functions and macros that reference the function that you want to move will not be updated with the new function location. This situation can cause referencing objects such as macros to be out of synchronization.

Click **Yes** or **No**.

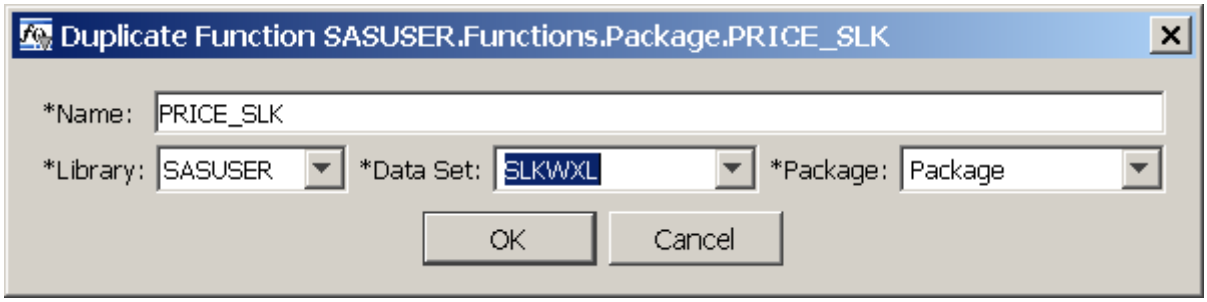
Close a Function

When you right-click the function name in the left pane and select **Close**, the window that displays that function closes. You can also close the function by clicking **OK** in the bottom right corner of the window that displays the function.

Duplicate a Function

You can duplicate (copy) a function that you are viewing to an existing or new package or library to which you have write access. To duplicate a function, select the function in the left pane. Right-click the function and select **Duplicate** from the menu. The following dialog box appears:

Display 21.7 *The Duplicate Function Dialog Box*



The fields in this dialog box automatically display the function name, library, data set, and package of the function that you want to duplicate. You can change these fields when you duplicate the function.

For a description of these fields, see "Move a Function" on page 602.

Rename a Function

Use the Rename dialog box to rename a function within a given package. You must have write access to the library that contains the function. When you rename a function, the new function resides in the same library as the original function.

To rename a function, select the function in the left pane. Right-click the function and select **Rename** from the menu. Enter the new name of the function and click **OK**.

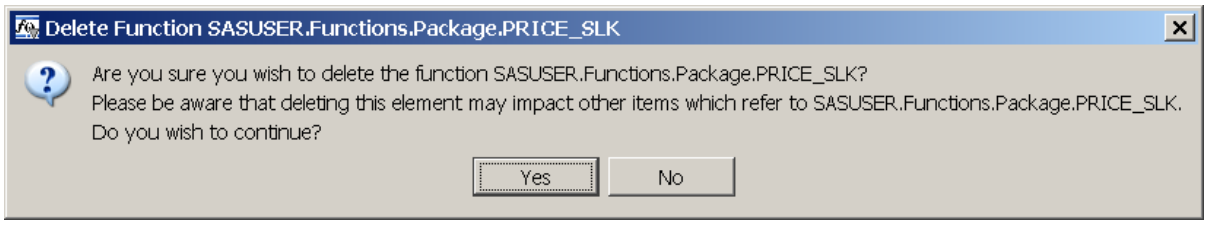
CAUTION:

Rename enables you to rename a function within a given package. Just as with moving a function, the renaming of a function does not modify dependent macros and other entities.

Delete a Function

You can delete a function from a library to which you have write access. To delete a function, select the function that you want to delete. Right-click the function and select **Delete** from the menu. The following dialog box appears, cautioning you about the impact that **Delete** has on other items:

Display 21.8 *Delete Function Confirmation Dialog Box*



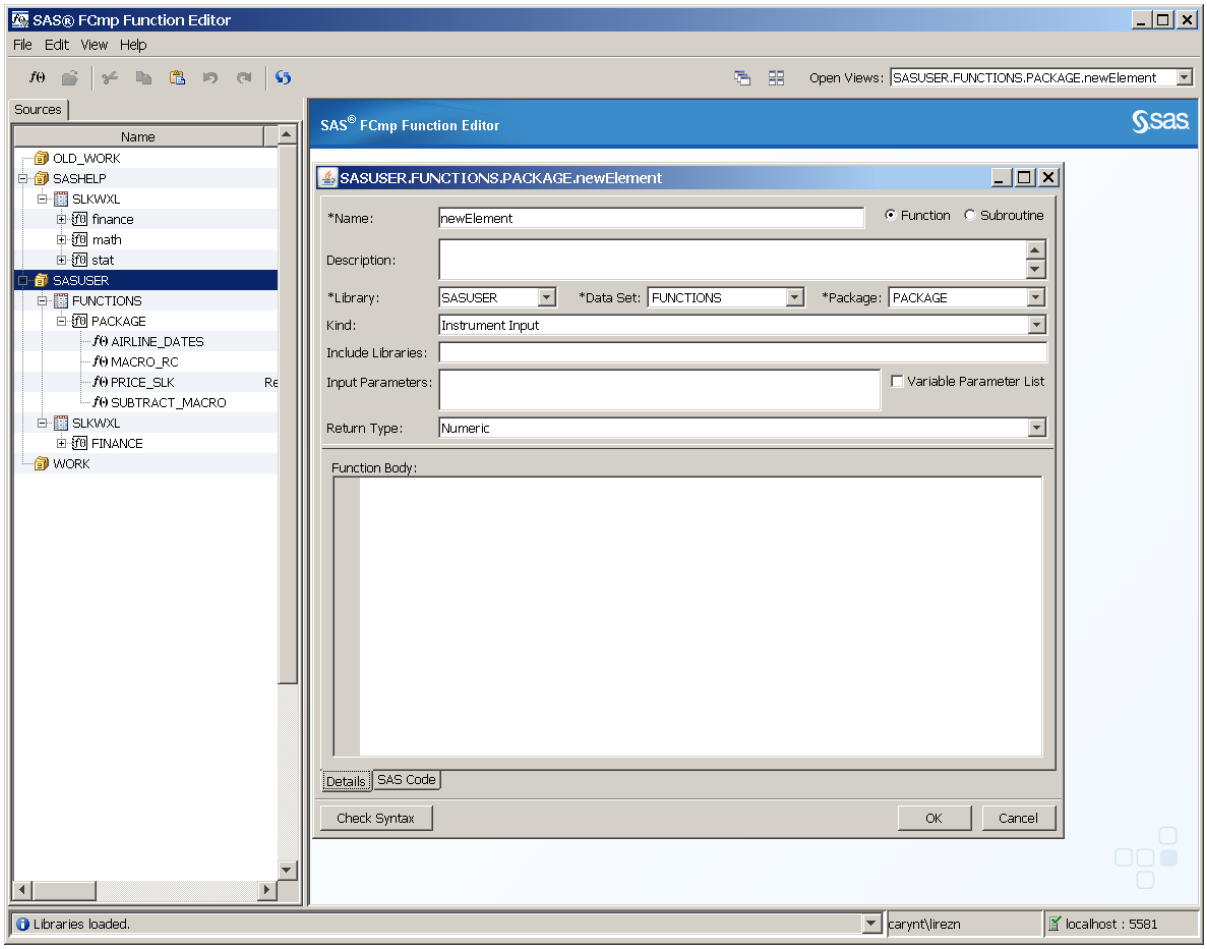
Click **Yes** or **No**.

Creating a New Function

You can create a new function whenever you have a library, data set, or package selected. To create a new function in a library, position your cursor on the library into which the new function will be added. Right-click the library and select **New Function**.

You can also select **File** ⇨ **New Function** from the menu or click  in the upper left corner below the menu bar. The following window appears:

Display 21.9 *The newElement Window*



The upper right corner of the window contains two buttons: **Function** and **Subroutine**. Click one of the buttons depending on whether you want to create a new function or a new subroutine.

The newElement window contains the following fields:

Name
specifies the name of the new function.

Description
describes the new function.

Library
specifies the library that will contain the new function. Enter the name of the library, or click the down arrow in the **Library** field to select a library.

Data Set
specifies the data set that will contain the new function. Enter the name of the data set, or click the down arrow in the **Data Set** field to select a data set. If you do not specify a value, the value in this field defaults to Functions.

Package
specifies the name of the package that will contain the new function. Enter the name of the package, or click the down arrow in the **Package** field to select a package. The **Package** field is a required field. If you do not specify a value, the value in this field defaults to Package.

Kind
enables you to group functions within a given package. Four predefined kind groupings are available and are typically used with SAS Risk Management:

- Project
- Risk Factor Transformation
- Instrument Pricing
- Instrument Input

You can use one of these four groupings, or enter your own kind value in the **Kind** field. The function tree in the left pane groups the functions in a package into their kind grouping, if you specified a value for **Kind**.

Include Libraries
specifies libraries that contain SAS code that you want to include in your function.

Input Parameters
specifies the arguments that you use as input to the function.

Variable Parameter List
specifies whether the function supports a variable number of arguments.

Return Type
specifies whether the function returns a character or numeric value.

Function Body
is the area in the window in which you code your function.

Three buttons are located at the bottom left of the newElement window:

Details
provides you with an area in which to write descriptive information (name of the new function, list of include libraries, input parameters, and so on) about your function. You code your new function in the **Function Body** section. The **Details** tab is selected by default.

SAS Code
enables you to view the function that you have written. The **SAS Code** selection provides read-only capabilities.

Check Syntax
enables you to check the syntax for the code that you have written. If the syntax is correct, a dialog box appears stating that the syntax is correct. If the syntax contains an error, a dialog box appears that describes the error. An error message also appears in the lower left bar of the window. Syntax errors are written to the log, which you can access from the **View** ⇨ **Show Log** menu.

When you enter information in the descriptive portion of the **Details** tab, as well as in the **Function Body** section, the information is converted to SAS code that you can see when you select the **SAS Code** button.

Displaying New Libraries in the FCmp Function Editor

You can create a new library in a SAS session while still being logged on to the FCmp Function Editor. However, clicking the refresh button in the FCmp Function Editor will

not display the new library. You must log off from the FCmp Function Editor and then log back in to see the new library.

Viewing the Log Window, Function Browser, and Data Explorer

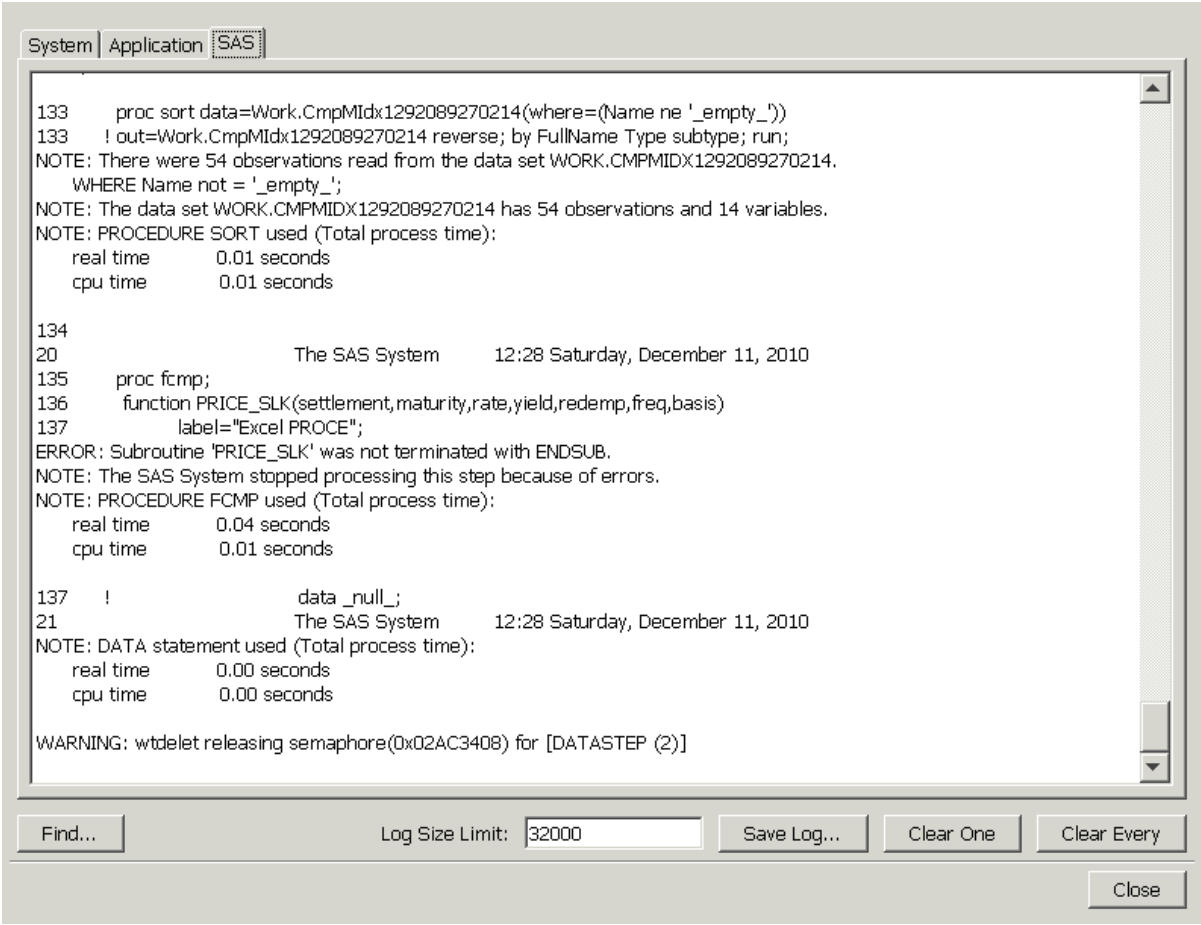
Log Window

To display the Log window, select **View** ⇨ **Show Log** from the menu. When you display the Log window, you can view system, application, and program results by selecting the tabs that are located in the upper left corner of the window.

Click the **SAS** tab to view the contents of the SAS log. The content of the log represents output from the SAS server. In addition, commands that are sent to SAS are also present to add context to the log output.

The following display shows the Log window with a SAS log displayed:

Display 21.10 *The Log Window*



The **System** tab in the Log window shows detailed information in the form of system messages. The window will be blank if no messages are logged.

The System window contains two vertical tabs that are located in the upper right section of the window. These tabs provide information about messages that might be of interest:

System.out
displays system output if messages are routed to this location.

System.err
displays error messages if the messages are routed to this location.

The Log window contains three buttons that are located at the bottom right of the window:

Save Log
saves the log output to a file that you choose.

Clear One
clears the results in the active window.

Clear Every
clears the results in all three of the windows.

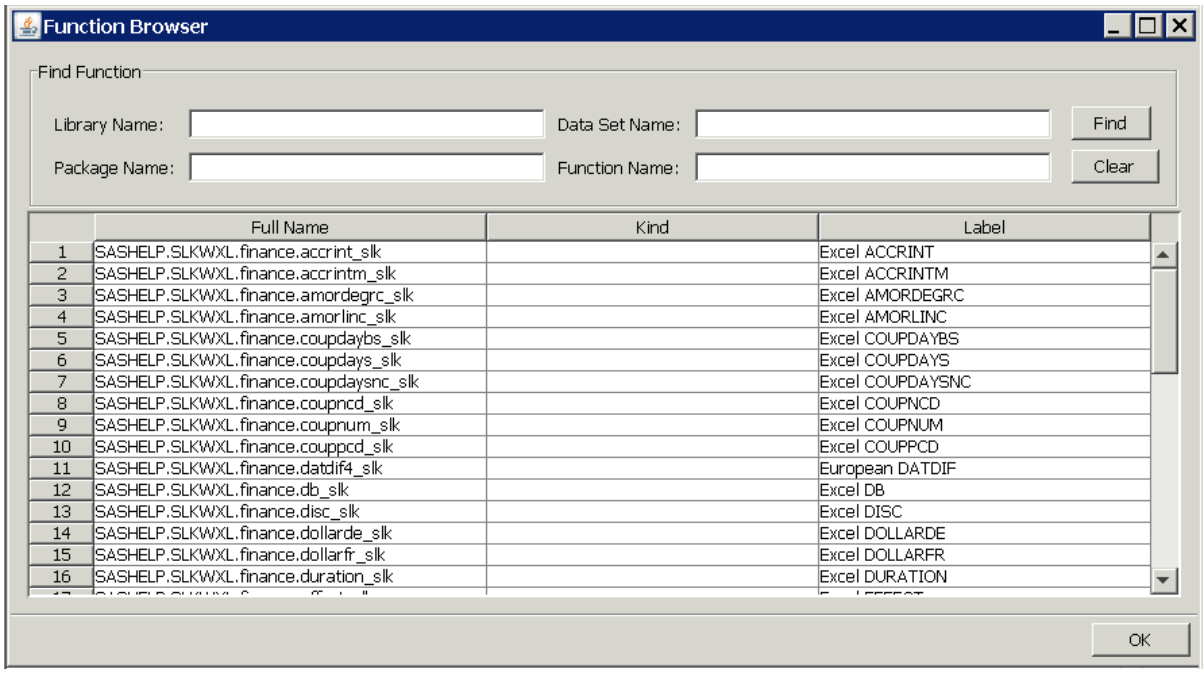
The **Find** button is located at the bottom left of the window. This button opens a dialog box that enables you to search your output. For example, searching for ERROR when the **SAS** tab is selected enables you to quickly find errors in the SAS log.

Function Browser

The Function Browser displays all of the functions that are listed in the left pane of the window. You can filter this list of functions to display a subset of the functions.

To display the Function Browser, select **View** ⇨ **Show Function Browser** from the menu. A window similar to the following appears:

Display 21.11 *The Function Browser Window*

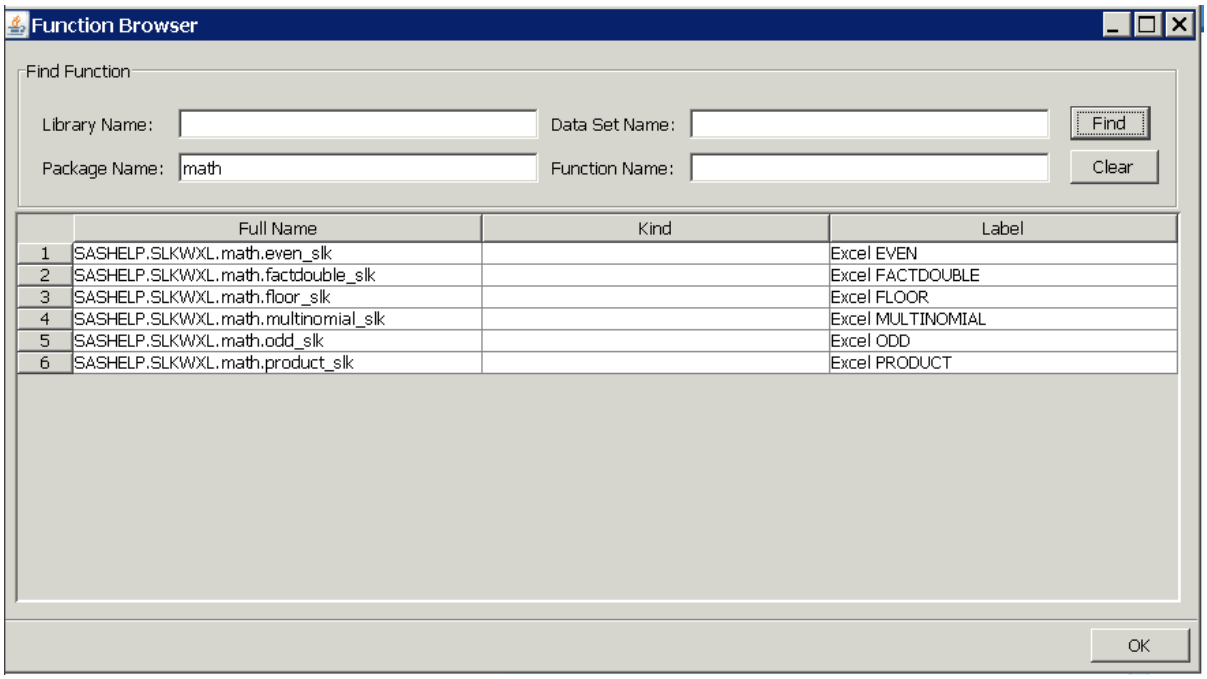


The partial output displayed above shows the functions in the application tree. You can filter the output and create a subset of the functions by entering your criteria in the

Function Browser fields that are located above the list of functions. These fields are **Library Name**, **Data Set Name**, **Package Name**, and **Function Name**.

In the following display, the **Package Name** field is used as the filter. When you press the **OK** button that is located in the bottom right corner of the window, or if you press the **Find** button that is located in the upper right corner, the following window appears:

Display 21.12 *Filtered Output from the Function Browser*



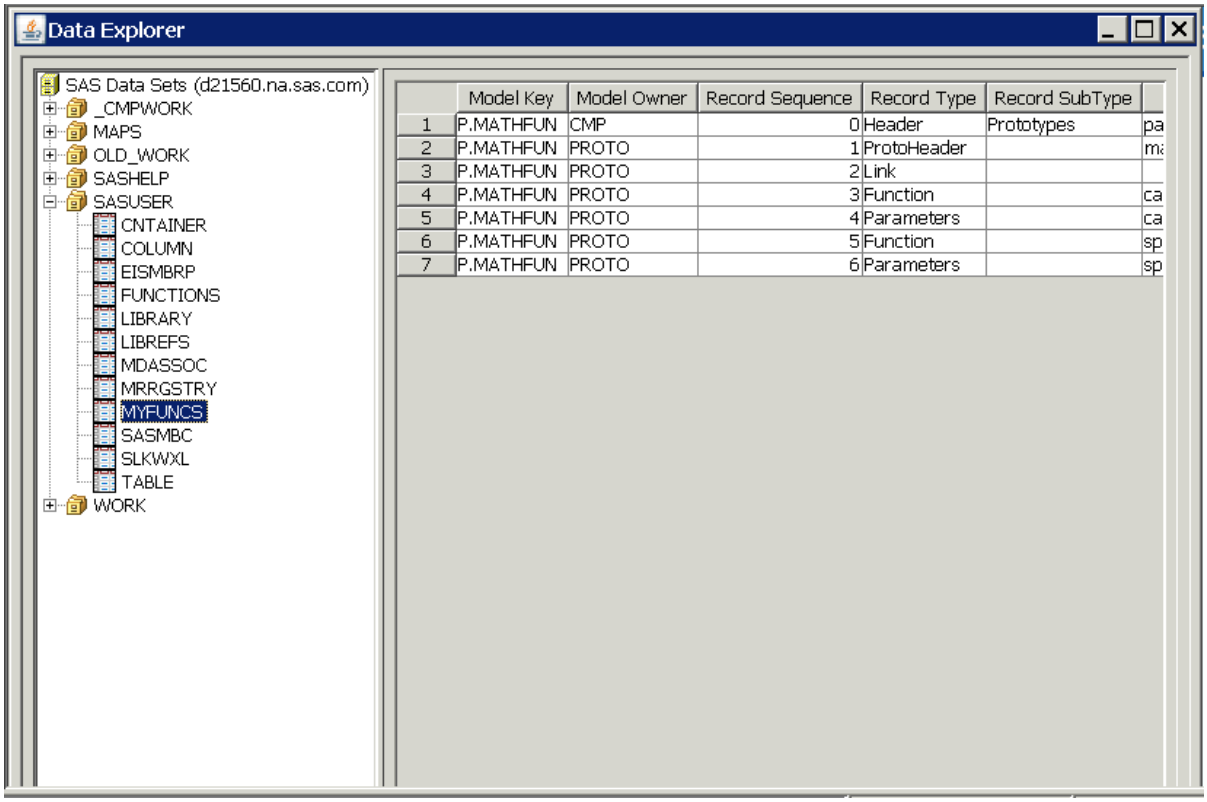
The math functions that are listed are a subset of all of the functions.

You can enter information in the fields that you choose, depending on your filter criteria. For example, if you enter a value, such as SASHELP, in the **Library Name** field, then all of the functions that are in the SASHELP library appear.

Data Explorer

The Data Explorer enables you to view the data in a data set that you select. To display the Data Explorer, select **View** ⇨ **Show Data Explorer** from the menu. A window similar to the following appears:

Display 21.13 *The Data Explorer Window*



The Data Explorer window displays data set information based on the data set you select from the left pane.

By clicking the column headings, you can move the columns to reposition them in the display. When you click **OK** in the lower right section of the window, the changes that you made are saved.

Using Functions in Your DATA Step Program

For an example of how PROC FCMP and DATA step syntax work together, see "Directory Transversal" on page 534.

Chapter 22

FONTREG Procedure

Overview: FONTREG Procedure

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:

- "Font and Font Rendering" in *SAS/GRAPH: Reference*
- GDEVICE Procedure in *SAS/GRAPH: Reference*
- the FONTSLOC and SYSPRINTFONT system options 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 (for example, in the TEMPLATE procedure or in the STYLE= option in the REPORT procedure), use the tag to distinguish between fonts that have the same name:

```
proc report data=grocery nowd
            style(header)=[font_face='<ttf> Palatino Linotype'];
run;
```

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 22.1 *Supported Font Types*

Rank	Type	Tag	File extension(s)
1	TrueType	**<ttf>**	**.ttf**
2	Type1	**<at1>**	**.pfa** **.pfb**

Note: OpenType font is an extension of TrueType font and is supported by SAS.

Note: 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.

Note: PDF and PostScript do not support double-byte Type1 fonts.

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, GIF733, GIFANIM
- SAS/GRAPH JPEG
- SAS/GRAPH PCL
- SAS/GRAPH PNG
- SAS/GRAPH SASEMF
- SAS/GRAPH SASWMF

- SAS/GRAPH TIFFP, TIFFB
- Universal EMF
- Universal PNG
- Universal Printing GIF
- Universal Printing PCL
- Universal Printing PDF
- Universal PS
- Universal SVG

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, as well as all of the other font files in that Window's font directory, by submitting the following code:

```
proc fontreg mode=add;
fontpath '!SYSTEMROOT\fonts';
run;
```

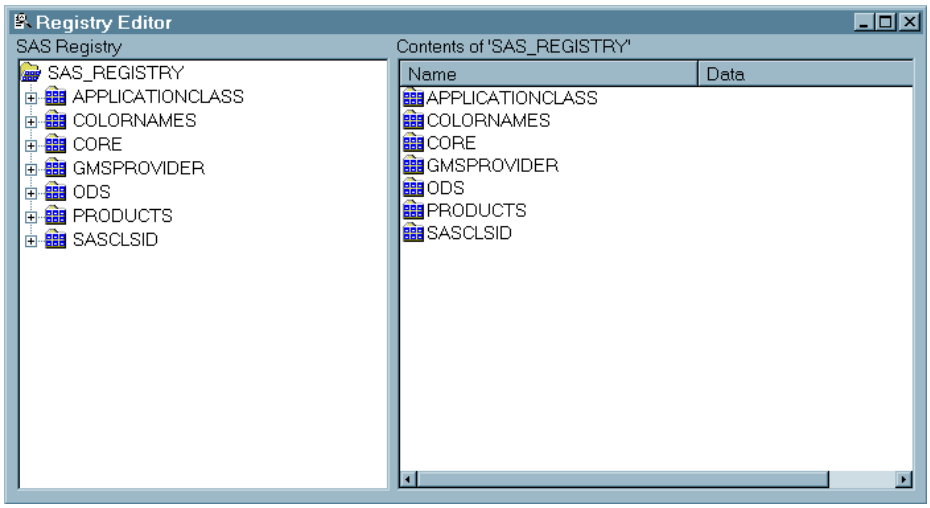
Removing Fonts from the SAS Registry

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.

Display 22.1 *SAS Registry Editor*



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 618.

For more information about PROC REGISTRY, see Chapter 45, "REGISTRY Procedure," on page 1203.

Font Aliases and Locales

The FONTFILE, FONTPATH, and TRUETYPE 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.

Note: For z/OS sites that do not use the hierarchical file system (HFS), only the FONTFILE statement is supported. For more information, see "FONTREG Procedure" in *SAS Companion for z/OS*

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.

See: FONTREG Procedure in *SAS Companion for z/OS*

PROC FONTREG <*option(s)*>;
FONTFILE '*file*' <…'*file*'> || '*file-1*, *pfm-file-1*, *afm-file-1*' <...'*file-n*'>;
FONTPATH '*directory*' <…'*directory*'>;
REMOVE '*family-name*' | '*alias*' | *family-type* | _ALL_;
TRUETYPE '*directory*' <…'*directory*'>;
TYPE1 '*directory*' <…'*directory*'>;

Statement	Task
"PROC FONTREG Statement"	Specify how to handle new and existing fonts
"FONTFILE Statement"	Identify which font files to process
"FONTPATH Statement"	Search directories to identify valid font files to process. (In the Windows operating environment only, locate the fonts folder if you don't know where the folder is located.)
"REMOVE Statement"	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 Statement"	Search directories to identify TrueType font files
"TYPE1 Statement"	Search directories to identify valid Type 1 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 <*option(s)*>;

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 623

MSGLEVEL=VERBOSE | NORMAL | TERSE | NONE
specifies the level of detail to include in the SAS log:

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 621

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 621

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 or the FONTPATH statements, 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 an AFM file, then SAS does not search for the PFM or the AFM files.

Specifying a PFM or an 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 an 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 specific to Windows that 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 an AFM File

You do not need to specify a PFM or an 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 an 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 621

Syntax

FONTPATH '*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.

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 (FONTFILE, FONTPATH, TRUETYPE, and TYPE1) are processed in the order in which they are received, but 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 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 623

Syntax

TRUETYPE '*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.

TYPE1 Statement

Specifies one or more directories to be searched for valid Type1 font files.

Syntax

TYPE1 '*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.

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;
```

Output: Log

Log 22.1 *Log Output from Adding a Single Font File*

```
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;
```

Output: Log

Log 22.2 *Log Output 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;
```

Output: Log

Log 22.3 *Log Output from 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
```

Chapter 23

FORMAT Procedure

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 print the parts of a catalog that contain informats or formats, store descriptions of informats or formats in a SAS data set, and use a SAS data set to create informats or formats.

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.

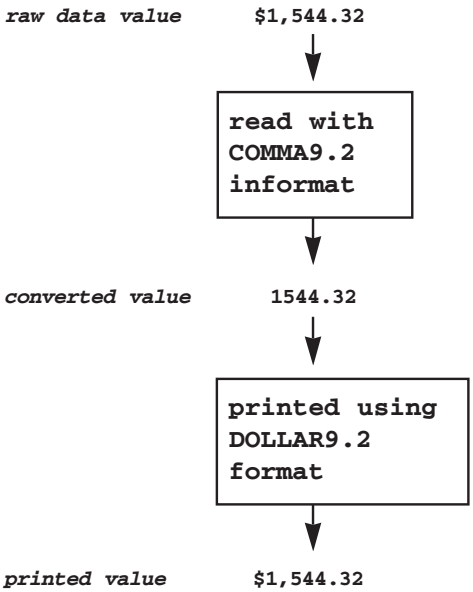
With formats, you can do the following:

- 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 COMMA*w.d* informat and the DOLLAR*w.d* format are provided by SAS.

Display 23.1 *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 627.

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 23.1 *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.

Step	Informats	Formats
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: creating the format or informat with the FORMAT procedure, and then assigning the format or informat with the FORMAT, INFORMAT, or ATTRIB statement.

It is often useful to assign informats in the FSEDIT procedure in SAS/FSP software and in the BUILD procedure in SAS/AF software.

For complete documentation on the ATTRIB, INFORMAT, and FORMAT statements, see *SAS Statements: Reference*. For complete documentation on the INPUT and PUT functions, see *SAS Functions and CALL Routines: Reference*. See "Formatted Values" on page 26 for more information and examples of using formats in Base SAS procedures.

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

[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 Language Reference: Concepts*.

- 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.

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 631.

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 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 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-n*>);

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 9: Writing Ranges for Character Strings" on page 690.

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 print 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 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.

Syntax: FORMAT Procedure

Restriction: You cannot use a SELECT statement and an EXCLUDE statement within the same PROC FORMAT step.

Tip: You can also use appropriate global statements with this procedure. See "Global Statements" on page 20 for a list.

See: "FORMAT Procedure: z/OS" in *SAS Companion for z/OS*

PROC FORMAT <*option(s)*>;
EXCLUDE *entry(s)*;
INVALUE <$>*name* <(*informat-option(s)*)>
<*value-range-set(s)*>;
PICTURE *name* <(*format-option(s)*)>
<*value-range-set-1*<(*picture-1-option(s)*)>>
<…*value-range-set-n*<(*picture-n-option(s)*)>>;
SELECT *entry(s)*;
VALUE <$>*name* <(*format-option(s)*)>
<*value-range-set(s)*>;

Statement	Task	Example
"PROC FORMAT Statement"	Define formats and informats for variables	Ex. 2, Ex. 6, Ex. 7, Ex. 8
"EXCLUDE Statement"	Exclude catalog entries from processing by the FMTLIB and CNTLOUT= options	
"INVALUE Statement"	Create an informat for reading and converting raw data values	Ex. 5
"PICTURE Statement"	Create a template for printing numbers	Ex. 2, Ex. 10, Ex. 11
"SELECT Statement"	Select catalog entries for processing by the FMTLIB and CNTLOUT= options	Ex. 7
"VALUE Statement"	Create a format that specifies character strings to use to print variable values	Ex. 3, Ex. 4, Ex. 9

PROC FORMAT Statement

Creates user-specified formats and informats for variables.

Tip: You can use data set options with the CNTLIN= and CNTLOUT= data set options. See "Data Set Options" on page 19 for a list.

Examples: "Example 2: Creating a Picture Format" on page 673
"Example 6: Creating a Format from a Data Set" on page 682
"Example 7: Printing the Description of Informats and Formats" on page 686
"Example 8: Retrieving a Permanent Format" on page 688

Syntax

PROC FORMAT <*option(s)*>;

Summary of Optional Arguments

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.

MAXLABLEN=*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

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.

Note: 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 668

Example: "Example 6: Creating a Format from a Data Set" on page 682

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.

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 666

FMTLIB

To get information about specific informats or formats, subset the catalog using the SELECT or EXCLUDE statement.

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 the SELECT or EXCLUDE statement and omit the FMTLIB and CNTLOUT= options, then the procedure invokes the FMTLIB option and you receive FMTLIB option output.

Example: "Example 7: Printing the Description of Informats and Formats" on page 686

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 628

Example: "Example 2: Creating a Picture Format" on page 673

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= Values and Default Settings for ENCODING, PAPERSIZE, DFLANG, and DATESTYLE Options" in Chapter 18 of *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;
```

MAXLABLEN=*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.

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 5: Converting Raw Character Data to Numeric Values" on page 680

Syntax

INVALUE <$>*name* <(*informat-option(s)*)>
<*value-range-set(s)*>;

Summary of Optional Arguments

Control the attributes of the format

MAX=*length*
specifies a maximum length for the format.

MIN=*length*
specifies a minimum length for the format.

NOTSORTED
stores values or ranges in the order in which you define them.

UPCASE
upper cases all input strings before they are compared to ranges.

Control the attributes of the informat

DEFAULT=*length*
specifies the default length of the format.

JUST
left-justifies 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.

Restriction: A user-defined informat name cannot be the same as an informat name that is supplied by SAS.

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 format. 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.

The default length of an informat depends on whether the informat is character or numeric. The default length of character informats is the length of the longest informatted value. The default of a numeric informat is 12 if you have numeric data to the left of the equal sign. If you have a quoted string to the left of the equal sign, then the default length is the length of the longest string.

Tip: As a best practice, if you specify an existing informat in a value-range set, always specify the DEFAULT= option.

JUST
left-justifies all input strings before they are compared to ranges.

MAX=*length*
specifies a maximum length for the informat or format. When you associate the format with a variable, you cannot specify a width greater than the MAX= value.

Default: 40

Range: 1–40

MIN=*length*
specifies a minimum length for the informat or format.

Default: 1

Range: 1–40

NOTSORTED
stores values or ranges for informats or formats 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 or 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 informat or 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 the 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.

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 will become. The *value-range-set(s)* can be one or more of the following:

value-or-range-1<…, *value-or-range-n*>=*informatted-value* | [existing-informat]

The informat converts the raw data to the values of *informatted-value* on the right side of the equal sign.

value-or-range
See “Specifying Values or Ranges” on page 661.

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: 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.

Restriction: National language format directives can be used in the PICTURE statement only under DBCS and UTF-8 environments.

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 657.

If you are formatting DBCS characters, 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.

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 2: Creating a Picture Format" on page 673

"Example 10: Filling a Picture Format" on page 693

"Example 11: Creating a Format in a non-English Language" on page 694

Syntax

PICTURE *name* <(*format-option(s)*)>

<*value-range-set-1*<(*picture-1-option(s)*)>>

<…*value-range-set-n*<(*picture-n-option(s)*)>>;

Summary of Optional Arguments

Control the attributes of each picture in the format

DATATYPE=DATE | TIME | DATETIME

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.

DIG3SEP='*character*'
specifies the three-digit separator character for a number.

FILL='*character*'
specifies a character that completes the formatted value.

LANGUAGE=
specifies the language that is used for weekdays and months year that you can substitute in a date, time, or datetime picture.

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

DEFAULT=*length*
specifies the default length of the picture.

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 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.

Restriction: A user-defined format cannot be the name of a format supplied by SAS.

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 Chapter 3, "Rules for Words and Names in the SAS Language," in *SAS Language Reference: Concepts*.

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
enables the use of directives in the picture as a template to format date, time, or datetime values. Specify either DATE, TIME, or DATETIME based on the directive that you use in the picture format. See the definition and list of directives on page 646 in the description of *picture*.

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.

The default length of a picture is the length of the longest picture value.

Tip: If you are formatting DBCS characters, 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*'
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)

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 10: Filling a Picture Format" on page 693

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**.

If a variable value matches one value or range without the fuzz factor, and also matches another value or range with the fuzz factor, then the format assigns the variable value to the value or range that it matched without the fuzz factor.

Default: 1E−12 for numeric formats.

Tip: Specify FUZZ=0 to save storage space when you use the VALUE statement to create numeric formats.

LANGUAGE=

specifies the language that is used for weekdays and months year 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.

Valid values for LANGUAGE= are the same languages that you can specify for the DFLANG= system 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	

Default: English

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= Values and Default Settings for ENCODING, PAPERSIZE, DFLANG, and DATESTYLE Options" in Chapter 18 of *SAS National Language Support (NLS): Reference Guide*.

See: "DFLANG= System Option: UNIX, Windows, and z/OS" 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: 1–40

MIN=*length*

specifies a minimum length for the format.

Default: 1

Range: 1–40

MULTILABEL

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 variables, SAS stores the variables in sorted order. This order can produce unexpected results when variables 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.

Example: "Example 2: Creating a Picture Format" on page 673

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);
```

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 the 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

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 2: Creating a Picture Format" on page 673

"Example 10: Filling a Picture Format" on page 693

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-n>='picture'

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 9's as nonzero digit selectors.

message characters
are nonnumeric characters that print 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 print as

```
                              02
  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 641 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 permitted directives are as follows:

%a
abbreviated weekday name (for example, Wed).

Interaction: This directive can be used for all formats, including national language formats.

%A
full weekday name (for example, Wednesday).

Interaction: This directive can be used for all formats, including national language formats.

%b
abbreviated month name (for example, Jan).

Interaction: This directive can be used for all formats, including national language formats.

%B
full month name (for example, January).

Interaction: This directive can be used for all formats, including national language formats.

%C
long month name with blank padding (January through December) (for example, December).

Interaction: This directive can be used only for national language formats.

%d

day of the month.

Interaction: This directive can be used for all formats, including national language formats.

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").

Interaction: This directive can be used only for national language formats.

%F

full weekday name with blank padding.

Interaction: This directive can be used only for national language formats.

%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.

Interaction: This directive can be used only for national language formats.

%H

hour (24-hour clock).

Interaction: This directive can be used for all formats, including national language formats.

Note: To add a leading zero before a single-digit number, insert a 0 before the directive (for example, %0H).

%I

hour (12-hour clock).

Interaction: This directive can be used for all formats, including national language formats.

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.

Interaction: This directive can be used for all formats, including national language formats.

Note: To add a leading zero before a single-digit number, insert a 0 before the directive (for example, %0j).

%m

month (1–12).

Interaction: This directive can be used for all formats, including national language formats.

Note: To add a leading zero before a single-digit number, insert a 0 before the directive (for example, %0m).

%M
minute (0–59).

Interaction: This directive can be used for all formats, including national language formats.

Note: To add a leading zero before a single-digit number, insert a 0 before the directive (for example, %0M).

%n
number of days in a duration as a decimal number (maximum of 10 digits) (for example, 25).

Interaction: This directive is not valid for national languages.

%o
month (1-12) with blank padding (for example, " 2").

Interaction: This directive can be used only for national language formats.

%p
equivalent to either a.m. or p.m.

Interaction: This directive can be used for all formats, including national language formats.

%q
abbreviated quarter of the year string such as 1, 2, 3, or 4.

Interaction: This directive can be used for all formats, including national language formats.

%Q
quarter of the year string, such as Quarter1, Quarter2, Quarter3, or Quarter4.

Interaction: This directive can be used only for national language formats.

%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.

Interaction: This directive is not valid for national languages.

%S
seconds (0–59), allowing for possible leap seconds.

Interaction: This directive can be used for all formats, including national language formats.

Note: 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).

Interaction: This directive can be used only for national language formats.

%U
week number of the year as a decimal number (0–53). Sunday is considered the first day of the week.

Interaction: This directive can be used for all formats, including national language formats.

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.

Interaction: This directive can be used only for national language formats.

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).

Interaction: This directive can be used for all formats, including national language formats.

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.

Interaction: This directive can be used only for national language formats.

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).

Interaction: This directive can be used for all formats, including national language formats.

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).

Interaction: This directive can be used for all formats, including national language formats.

%z

UTC time-zone offset.

Interaction: This directive can be used only for national language formats.

%Z

time-zone name.

Interaction: This directive can be used only for national language formats.

%%

the % character.

Tip: Add code to your program to direct how you want missing values to be displayed.

value-or-range
See "Specifying Values or Ranges" on page 661.

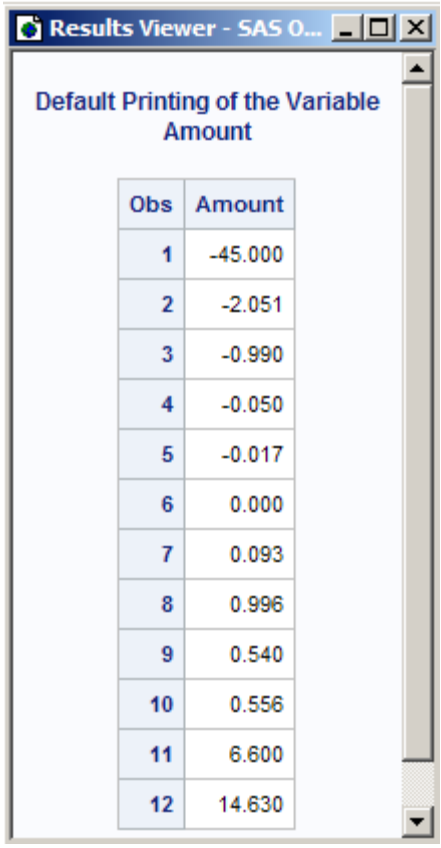
Interaction: If you specify LANGUAGE= and PICTURE= in the format definition, the format supports only English and the European languages. To use a user-defined format in languages other than those that are supported by the LANGUAGE= option, use the PICTURE= statement. Do not specify the LANGUAGE= option. 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 for formatting numbers with leading zeros. In the SAMPLE data set, the default printing of the variable Amount has leading zeros on numbers between 1 and −1:

```
data sample;
   format Amount nozeros.;
   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;
   title 'Default Printing of the Variable Amount';
run;
```



Obs	Amount
1	-45.000
2	-2.051
3	-0.990
4	-0.050
5	-0.017
6	0.000
7	0.093
8	0.996
9	0.540
10	0.556
11	6.600
12	14.630

The following PROC FORMAT step uses the ROUND format option and creates the NOZEROS. format, which eliminates leading zeros in the formatted values:

```
libname library 'SAS-library';

proc format;
   picture nozeros (round)
     low -  -1      =  '000.00' (prefix='-')
     -1 < - < -.99  =  '9.99'   (prefix='-' mult=100)
     -0.99 <-<  0   =  '99'     (prefix='-.' mult=100)
                0   =  '9.99'
      0 <  -< .99   =  '99'     (prefix='.' mult=100)
      0.99 - < 1    =  '9.99'   (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 23.3 on page 653.

Table 23.2 *Building a Picture Format*

Step	Rule	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.

Step	Rule	In This Example
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.)	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	Add a fuzz factor of 10e-8 to the number. If the ROUND option is not in effect, truncate the number after the decimal. If the ROUND option is in effect, then 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. Note: The fuzzing factor is not related to the FUZZ= option. The FUZZ= option is used in matching ranges to values.	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.
5	Turn the number into a character string. If the 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 Z*w.* format. Z*w.* is explained in the section on SAS formats in *SAS Formats and Informats: Reference*.	The numbers 205, 5, and 660 become the character strings `0205`, `005`, and `0660`, respectively. Because each picture is longer than the numbers, the format adds a leading zero to each value. The format does not add leading zeros to the number 56 because the corresponding picture has only two digit selectors.

Step	Rule	In This Example
6	Apply the character string to the picture. The format only maps 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 it and all subsequent leading zeros become part of the formatted value. If all of the leading zeros map to zero digit selectors, then none of the leading zeros become part of the formatted value. The format replaces the leading zeros in the character string with blanks. **	The leading zero is dropped from each of the character strings `0205` and `0660` because the leading zero maps to a zero digit selector in the picture.
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 `-.05` and `.55`.

* A decimal in a PREFIX= option is not part of the picture.

** You can use the FILL= option to specify a character other than a blank to become part of the formatted value.

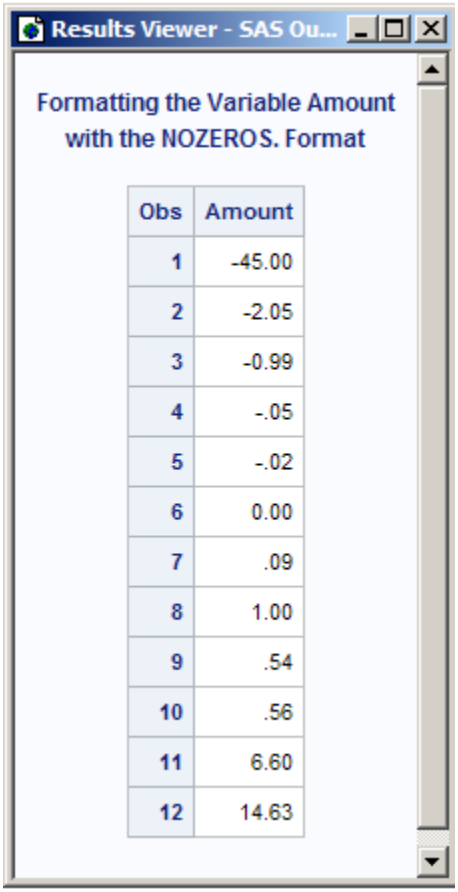
Table 23.3 *Steps to Format Various Values*

Step	Action	-2.051	-.05	.556	.996	6.6
1.	Range	low - -1	-0.99 < - <0	0 < - < .99	0.99 - <1	1 - high
2.	Picture	000.00	99	99	9.99	09.99
3.	Absolute value	2.051	.05	.556	.996	6.6

Step	Action	-2.051	-.05	.556	.996	6.6
4.	MULT=	2.051x10 ^{2}=205.1	.05x100=5	. 556x100=55.60 0	.996x100=99.6	6.6x10^{2}=660
5.	Round	205	5	56	100	660
6.	Character string	0205	05	56	100	0660
7.	Template	2.05	05	56	1.00	6.60
8.	Prefix	prefix='-'	prefix='-.'	prefix='.'	none	none
	Formatted result	-2.05	-.05	.56	1.00	6.60

The following PROC PRINT step associates the NOZEROS. format with the AMOUNT variable in SAMPLE. The output shows the result of rounding.

```
proc print data=sample noobs;
   format amount nozeros.;
   title 'Formatting the Variable Amount';
   title2 'with the NOZEROS. Format';
run;
```



Obs	Amount
1	-45.00
2	-2.05
3	-0.99
4	-.05
5	-.02
6	0.00
7	.09
8	1.00
9	.54
10	.56
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 7: Printing the Description of Informats and Formats" on page 686

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 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 or the CNTLOUT= option, the catalog is opened for read access.
- If you use the SELECT statement without the FMTLIB 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 3: Creating a Format for Character Values" on page 675
"Example 4: Writing a Format for Dates Using a Standard SAS Format" on page 677
"Example 9: Writing Ranges for Character Strings" on page 690

Syntax

VALUE <$>*name* <(*format-option(s)*)>
<*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.

value-range-set(s)
specifies the assignment of a value or a range of values to a formatted value.

Control the attributes of the format

NOTSORTED
stores values or ranges in the order in which you define them.

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.

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.

Tips:
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 663

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.

The default length of a format is the length of the longest formatted value.

Tip: As a best practice, if you specify an existing format 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 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**.

If a variable value matches one value or range without the fuzz factor, and also matches another value or range with the fuzz factor, then the format assigns the variable value to the value or range that it matched without the fuzz factor.

Default: 1E−12 for numeric formats and 0 for character formats.

Tip: Specify FUZZ=0 to save storage space when you use the VALUE statement to create 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: 1–40

MIN=*length*
specifies a minimum length for the format.

Default: 1

Range: 1–40

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 mat 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 the 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-n*>= | [*existing-format*]

The variable values on the left side of the equal sign prints as the character string on the right side of the equal sign.

value-or-range
For details about how to specify *value-or-range*, see "Specifying Values or Ranges" on page 661.

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 657.

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.

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*, with commas separating 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. If you omit the quotation marks around *value*, then PROC FORMAT assumes the quotation marks to be there.

You can use the keyword OTHER= as a single value. OTHER matches all values that do not match any other value or range. 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. For more information, see "Using a Function to Format Values" on page 663. For examples, see "Example 3: Creating a Format for Character Values"

on page 675 and "Example 12: Creating a Function to Use as a Format" on page 697.

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.

If you omit the quotation marks, then the procedure assumes quotation marks around each string. For example, if you specify the range **abc-zzz**, then the procedure interprets it as **'abc'-'zzz'**.

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
```

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 < after the value. If you are excluding the last value in a range, then put the < before the value. For example, the following range does not include 0:

```
0<-100
```

Likewise, the following range does not include 100:

```
0-<100
```

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 < noninclusion notation, 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 noninclusive notation 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, YYQ*w*., YYQ*xw*., YYQR*w*., and YYQR*xw*. 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.

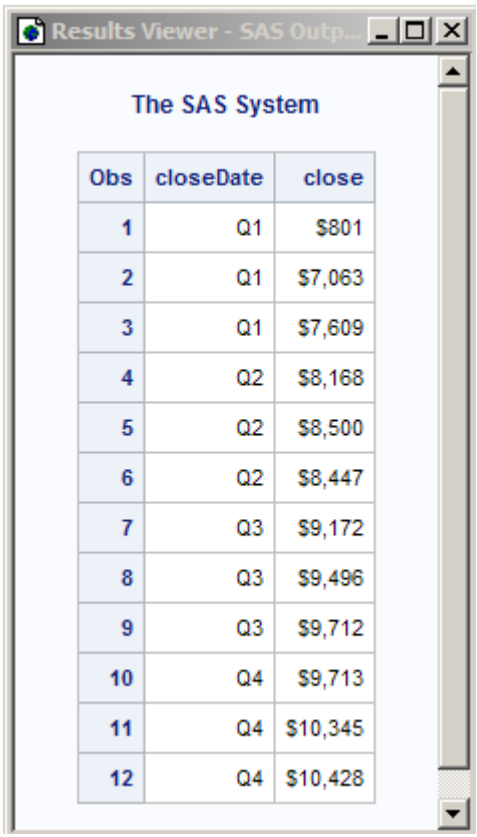
Here is an example:

```
/* 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 djia2009;
      input closeDate date7. close;
```

```
      datalines;
01jan09 800.86
02feb09 7062.93
02mar09 7608.92
01apr09 8168.12
01may09 8500.33
01jun09 8447.00
01jul09 9171.61
03aug09 9496.28
01sep09 9712.28
01oct09 9712.73
02nov09 10344.84
02dec09 10428.05
run;

proc print data=djia2009;
format closedate qfmt. close dollar9.;
run;
```

Here is the output:

Display 23.2 *Dow Jones Closings by Quarter*



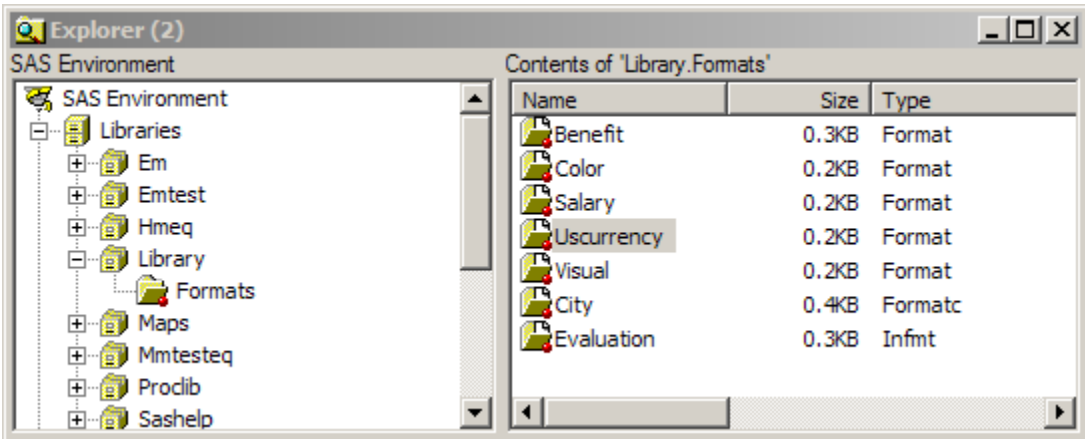
Obs	closeDate	close
1	Q1	$801
2	Q1	$7,063
3	Q1	$7,609
4	Q2	$8,168
5	Q2	$8,500
6	Q2	$8,447
7	Q3	$9,172
8	Q3	$9,496
9	Q3	$9,712
10	Q4	$9,713
11	Q4	$10,345
12	Q4	$10,428

You can use the function format as the value to OTHER=.

Viewing a Format Definition Using SAS Explorer

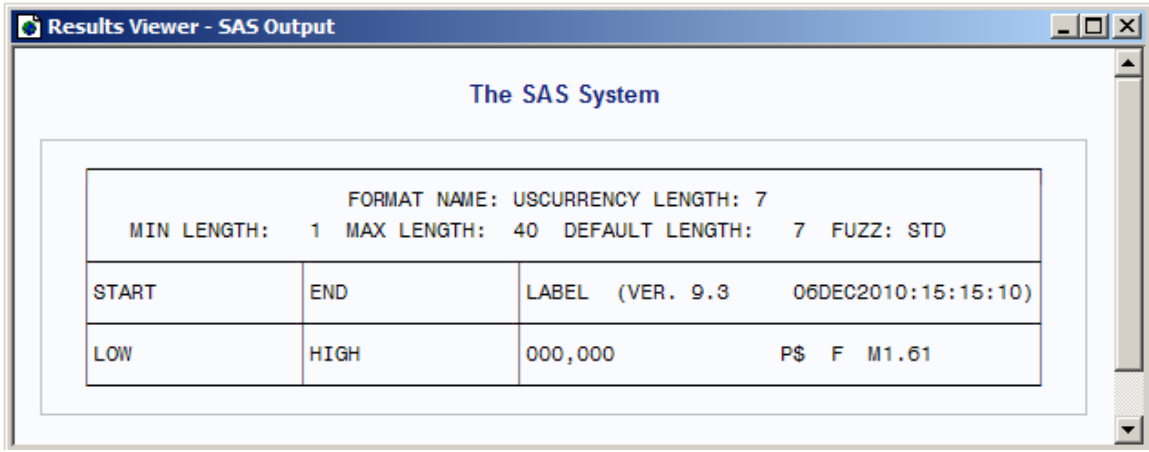
After you create a format, you can view the definition for the format using SAS Explorer.

1. Select **View** ⇨ **Explorer**.
2. Open the format folder. To view the default format folder, expand **Libraries** ⇨ **Library** and select **Formats**.



3. To view the format description, do one of the following actions on the format name in the contents pane:
   - double-click the format name
   - right-click the format name and select **View**

   The format description appears in the Results Viewer window.



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, and then using 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 with 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:

DEFAULT
specifies a numeric variable that indicates the default length for format or informat.

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.

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.

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 23.1 *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 655 and "EXCLUDE Statement" on page 634.

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 666. 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 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.

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. You can, however, 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.

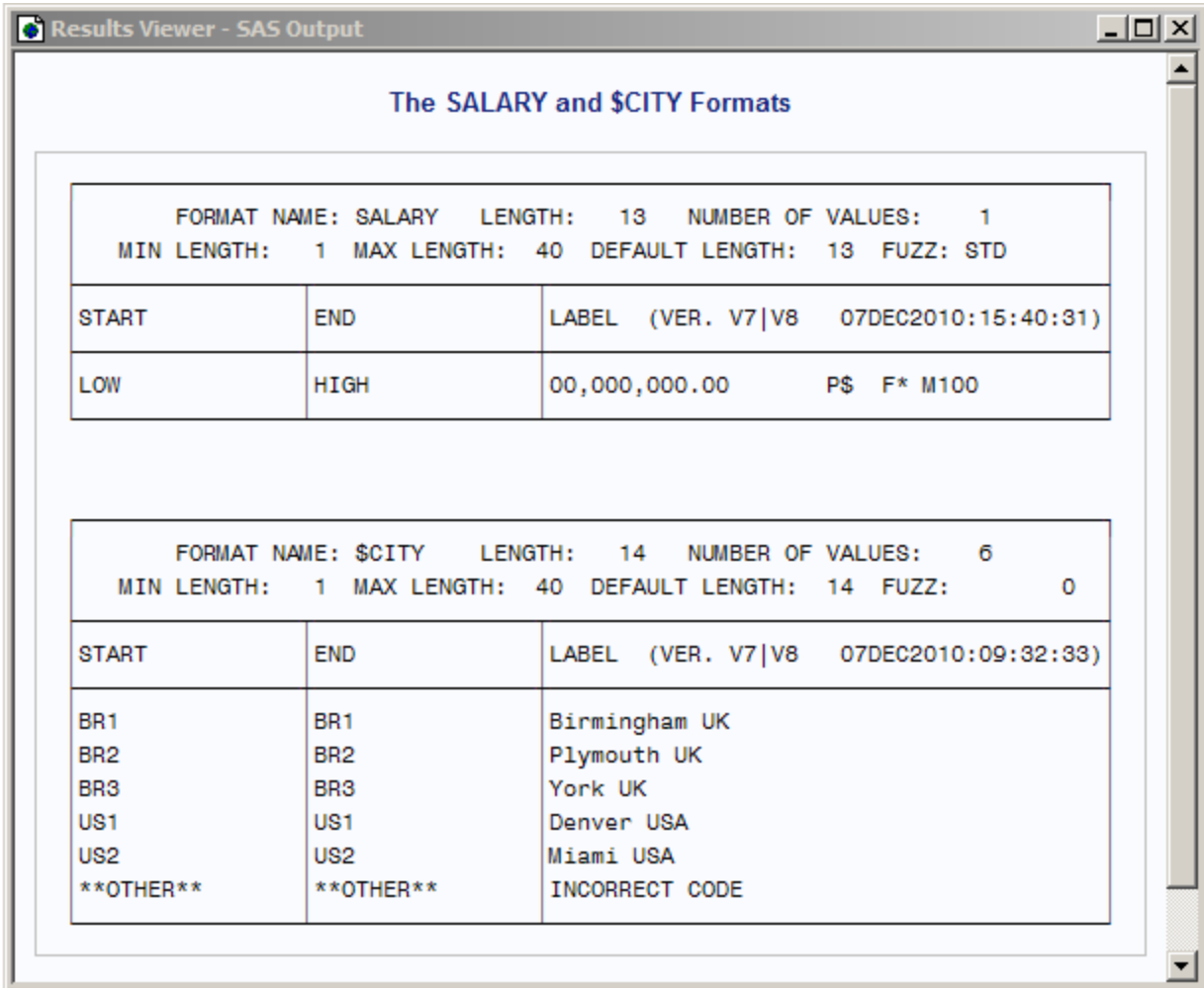
For an example featuring an input control data set, see "Example 6: Creating a Format from a Data Set" on page 682.

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 655 and "EXCLUDE Statement" on page 634. For an example, see "Example 7: Printing the Description of Informats and Formats" on page 686.

The FMTLIB output shown in the following output contains a description of the $CITY. format, which is created in "Example 3: Creating a Format for Character Values" on page 675, and the EVALUATION. informat, which is created in "Example 5: Converting Raw Character Data to Numeric Values" on page 680.

Output 23.2 *Output from PROC FORMAT with the FMTLIB Option*



Results Viewer - SAS Output

The SALARY and $CITY Formats

FORMAT NAME: SALARY LENGTH: 13 NUMBER OF VALUES: 1
MIN LENGTH: 1 MAX LENGTH: 40 DEFAULT LENGTH: 13 FUZZ: STD

START	END	LABEL (VER. V7|V8 07DEC2010:15:40:31)
LOW	HIGH	00,000,000.00 P$ F* M100

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 07DEC2010:09:32:33)
BR1	BR1	Birmingham UK
BR2	BR2	Plymouth UK
BR3	BR3	York UK
US1	US1	Denver USA
US2	US2	Miami USA
OTHER	**OTHER**	INCORRECT CODE

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.

LABEL
LABEL appears only for formats and contains either the formatted value or picture. The SAS version number and the date on which the format or informat was created are in parentheses after INVALUE or LABEL.

Note: If SAS displays version numbers V7|V8, then the format is compatible with those versions. If it is not compatible with those earlier releases, the release that created the format is shown. Version V9 supports long format and informat names (over 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 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 666 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 date7.;
   format hiredate date7.;
   datalines;
Capalleti, Jimmy#  2355# 21163# BR1# 30JAN09
Chen, Len#         5889# 20976# BR1# 18JUN06
Davis, Brad#       3878# 19571# BR2# 20MAR84
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# 15APR66
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 date7.;
   format hiredate date7.;
```

```
   datalines;
Capalleti, Jimmy#  2355# 21163# BR1# 30JAN09
Chen, Len#         5889# 20976# BR1# 18JUN06
Davis, Brad#       3878# 19571# BR2# 20MAR84
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# 15APR66
Sorrell, Joseph#   4421# 38760# US1# 19JUN11
Zook, Carla#       7385# 22988# BR3# 18DEC10
;
```

Example 2: 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

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.

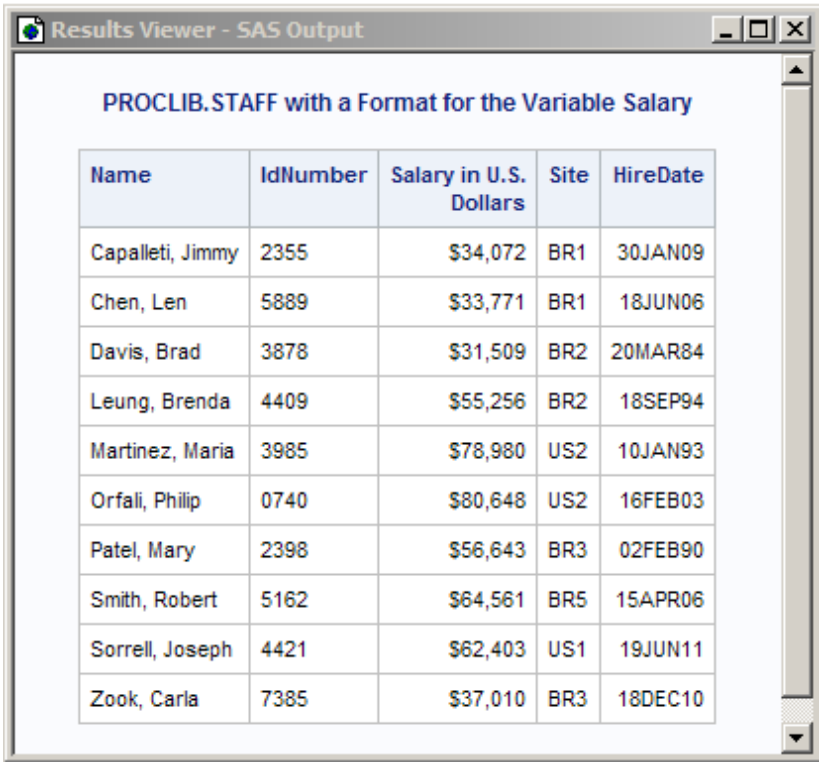
```
label salary='Salary in U.S. Dollars';
format salary uscurrency.;
```

Specify the title.

```
   title 'PROCLIB.STAFF with a Format for the Variable Salary';
run;
```

Output

Output 23.3 *PROCLIB.STAFF with a Format for the Variable Salary*



Results Viewer - SAS Output

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	20MAR84
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 3: Creating a Format for Character Values

Features: VALUE statement
OTHER keyword

Data set: PROCLIB.STAFF

Format: USCurrency

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.

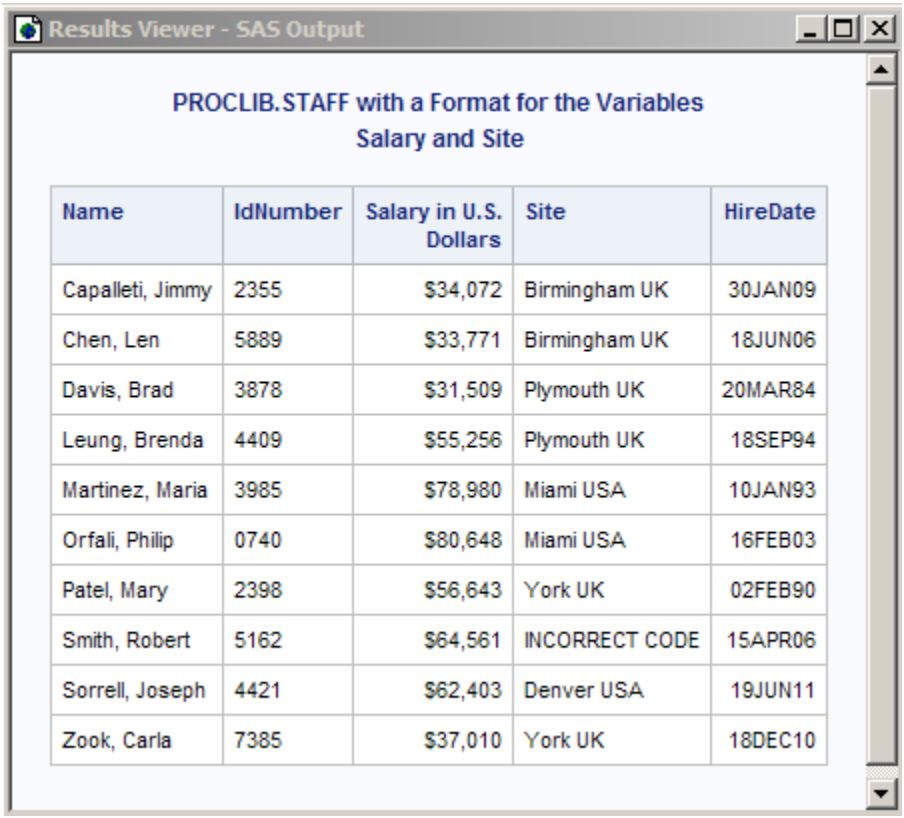
```
    format salary uscurrency. site $city.;
```

Specify the titles.

```
    title 'PROCLIB.STAFF with a Format for the Variables';
    title2 'Salary and Site';
run;
```

Output

Output 23.4 *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	20MAR84
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 4: Writing a Format for Dates Using a Standard SAS Format

Features: VALUE statement:
HIGH keyword

Data set: PROCLIB.STAFF

Format: USCurrency.

Format: $City.

Details

This example uses an existing format that is supplied by SAS as a formatted value.

Tasks include the following:

- creating a numeric format
- nesting formats
- writing a format using a standard SAS format
- formatting dates

Program

```
libname proclib 'SAS-library-1';
libname library 'SAS-library-2';

proc format library=library;

   value benefit
      low-'31DEC1999'd=[worddate20.]

      '01JAN2000'd-high='  ** Not Eligible **';
   run;

proc print data=proclib.staff noobs label;

   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 31DEC1999. 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, 1999. 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 a catalog named 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 31DEC1999 and those who were hired after that date. The keyword LOW and the SAS date constant '31DEC1999'D create the first range, which includes all date values that occur on or before December 31, 1999. For values that fall into this range, SAS applies the WORDDATE*w*. format. For more information about SAS date constants, see Chapter 7, “Dates, Times, and Intervals,” in *SAS Language Reference: Concepts*.For more information about the WORDDATE formats, see “WORDDATEw. Format” in *SAS Formats and Informats: Reference*.

```
value benefit
   low-'31DEC1999'd=[worddate20.]

   '01JAN2000'd-high='  ** Not Eligible **';
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.

```
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, Site, and Hiredate. The FORMAT statement associates the USCurrency. format (created in "Example 2: Creating a Picture Format" on page 673) with SALARY, the $CITY. format (created in "Example 3: Creating a Format for Character Values" on page 675) 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 23.5 *PROCLIB.STAFF with a Format for the Variables Salary, Site, and HireDate*

Results Viewer - SAS Output

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	** Not Eligible **
Davis, Brad	3878	$31,509	Plymouth UK	March 20, 1984
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	** Not Eligible **
Patel, Mary	2398	$56,643	York UK	February 2, 1990
Smith, Robert	5162	$64,561	INCORRECT CODE	** Not Eligible **
Sorrell, Joseph	4421	$62,403	Denver USA	** Not Eligible **
Zook, Carla	7385	$37,010	York UK	** Not Eligible **

Example 5: 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 'O'=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.

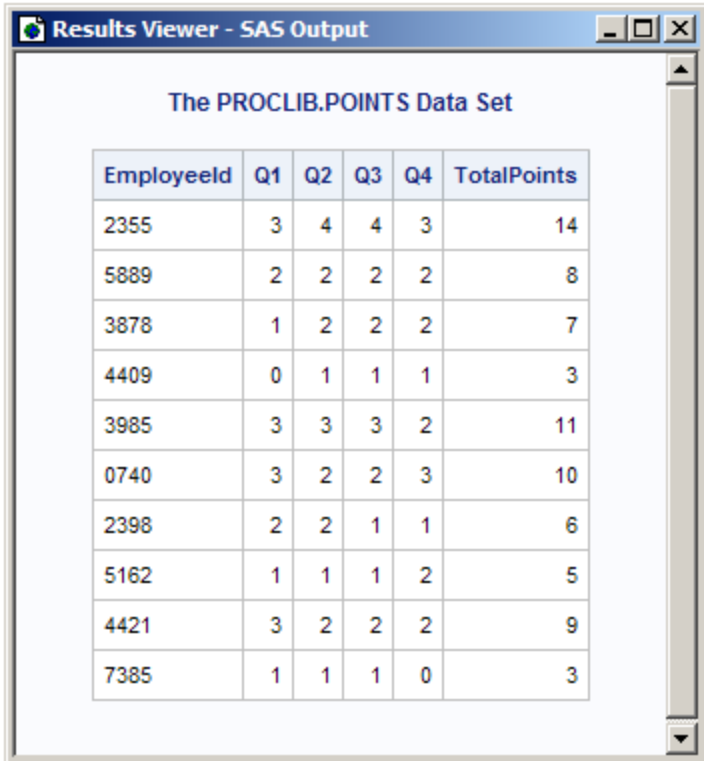
```
proc print data=proclib.points noobs;
```

Specify the title.

```
   title 'The PROCLIB.POINTS Data Set';
run;
```

Output

Output 23.6 *The PROCLIB.POINT Data Set*



Results Viewer - SAS Output

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 6: Creating a Format from a Data Set

Features: PROC FORMAT statement option:
CNTLIN=
Input control data set

Data set: WORK.POINTS, created from data lines in the sample code.

Details

This example shows how to create a format from a SAS data set.

Tasks include the following:

- creating a format from an input control data set
- creating an input control data set from an existing SAS data set

Program

```
data scale;
   input begin $ 1-2 end $ 5-8 amount $ 10-12;
   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 $ 1-2 end $ 5-8 amount $ 10-12;
   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 23.7 *The CTRL Data Set*

Results Viewer - SAS Output

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

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 PTSFRMT.

```
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=library;
   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
;
```

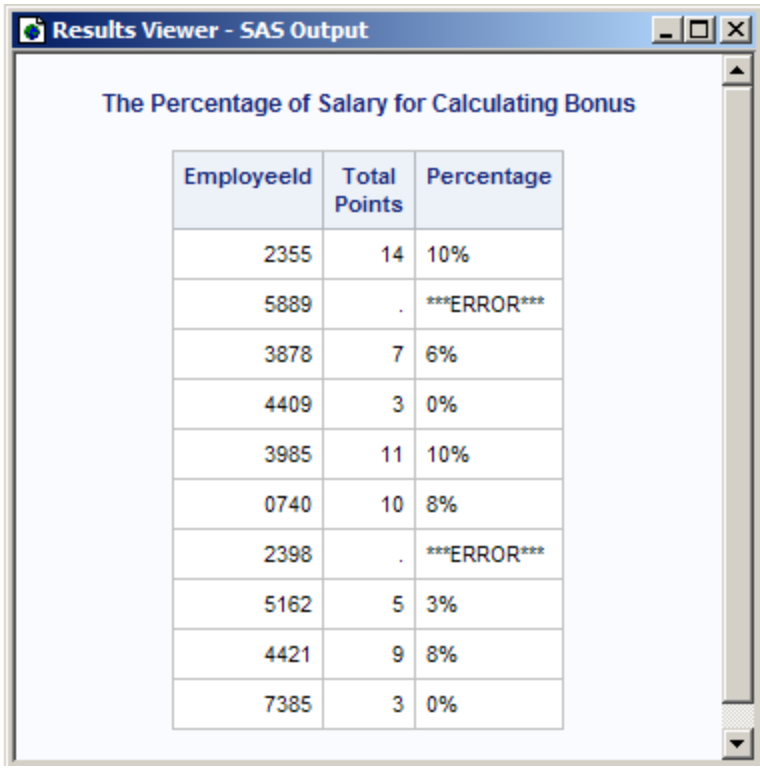
Generate a report for WORK.POINTS and associate the PTSFRMT. format with the TotalPoints variable. The DEFINE statement performs the association. The column that contains the formatted values of TotalPoints is using the alias Pctage. Using an alias enables you to print a variable twice, once with a format and once with the default format. For more information about the REPORT procedure, see Chapter 46, “REPORT Procedure,” on page 1220.

```
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 23.8 *The Percentage of Salary for Calculating Bonus*



Results Viewer - SAS Output

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 7: Printing the Description of Informats and Formats

Features: PROC FORMAT statement option:
FMTLIB
SELECT statement

Format:

Informat: EVALUATION.Evaluation

Details

This example illustrates how to print a description of an informat and a format. The description shows the values that are input and output.

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.

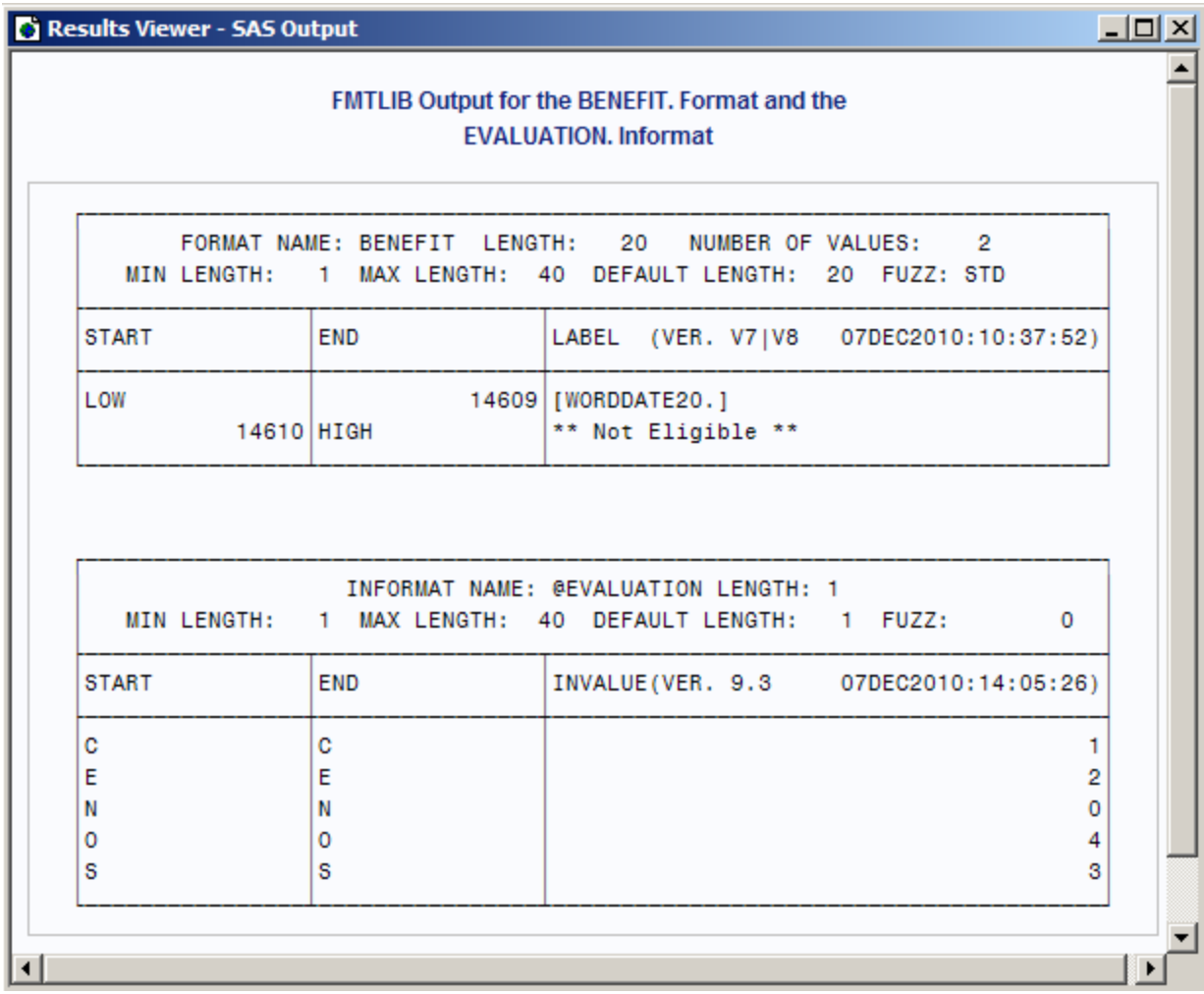
```
   select @evaluation benefit;
```

Specify the titles.

```
   title 'FMTLIB Output for the BENEFIT. Format and the';
   title2 'EVALUATION. Informat';
run;
```

Output

Output 23.9 *FMTLIB Output for the BENEFIT Format and the EVALUATION Informat*



Example 8: Retrieving a Permanent Format

Features: PROC FORMAT statement options:
LIBRARY=

Other features: 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
               low   -     -1  =  '00.00' (prefix='-'          )
                -1  <-<     0  =     '99' (prefix='-.' mult=100)
                 0   -<     1  =     '99' (prefix='.'  mult=100)
                 1   -   high  =  '00.00';
run;

options  fmtsearch=(proclib);

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 650.

```
   picture nozeros
               low   -     -1  =  '00.00' (prefix='-'          )
                -1  <-<     0  =     '99' (prefix='-.' mult=100)
                 0   -<     1  =     '99' (prefix='.'  mult=100)
                 1   -   high  =  '00.00';
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);
```

Print the SAMPLE data set. The FORMAT statement associates the NOZEROS. format with the Amount variable.

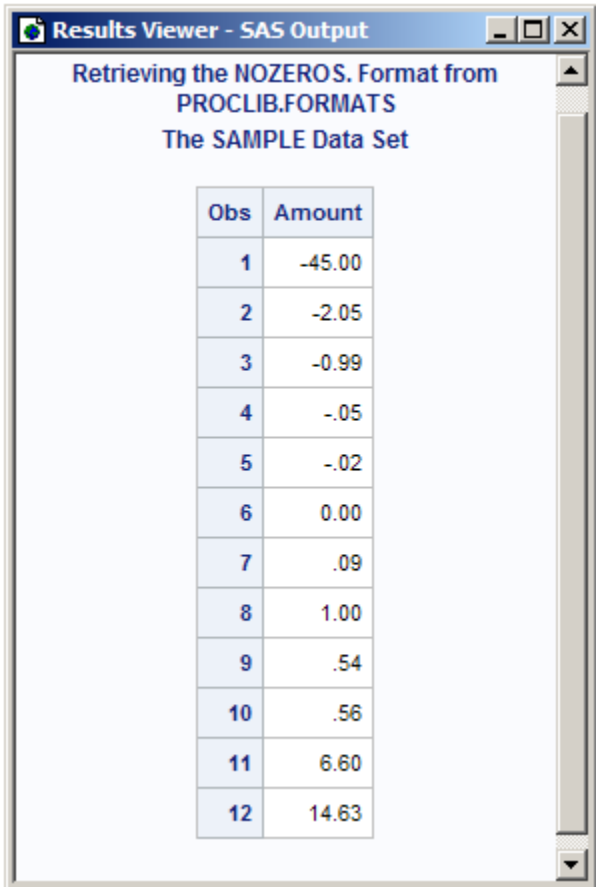
```
proc print data=sample;
   format amount nozeros.;
```

Specify the titles.

```
   title1 'Retrieving the NOZEROS. Format from PROCLIB.FORMATS';
   title2 'The SAMPLE Data Set';
run;
```

Output

Output 23.10 *Retrieving the NOZEROS. Format from PROCLIB.FORMATS*



Obs	Amount
1	-45.00
2	-2.05
3	-0.99
4	-.05
5	-.02
6	0.00
7	.09
8	1.00
9	.54
10	.56
11	6.60
12	14.63

Example 9: 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 1: Create the Example Data Set" on page 672.

```
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.

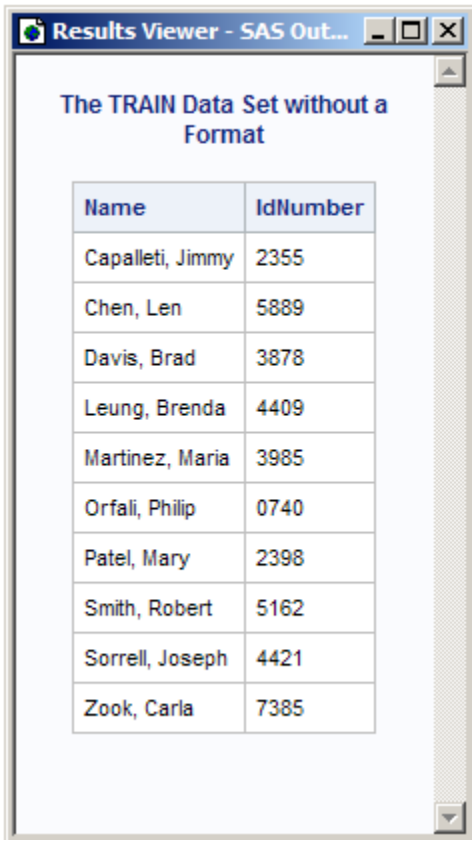
```
proc print data=train noobs;
```

Specify the title.

```
   title 'The TRAIN Data Set without a Format';
run;
```

Output

Output 23.11 *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 $SKILL. 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.

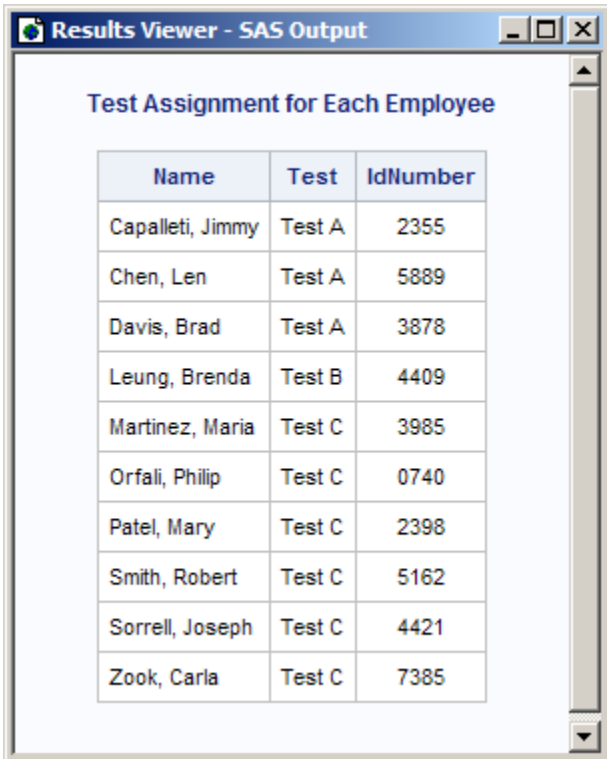
```
   value $skilltest  'a'-<'e','A'-<'E'='Test A'
                     'e'-<'m','E'-<'M'='Test B'
                     'm'-'z~','M'-'Z~'='Test C';
run;
```

Generate a report of the TRAIN data set. The FORMAT= option in the DEFINE statement associates $SkillTest. 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. See for more information about PROC REPORT.

```
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 23.12 *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 10: Filling a Picture Format

Features: PICTURE statement options:
FILL=
PREFIX=

Details

This example does the following:

- 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 monthlysalary 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 leftmost 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.

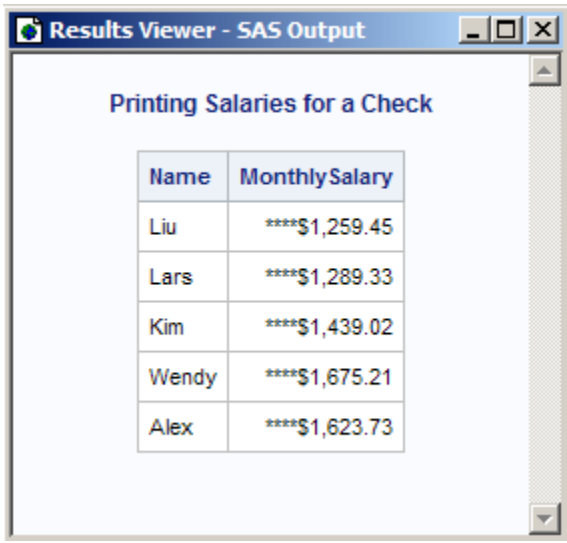
```
proc print data=pay noobs;
   format monthlysalary salary.;
```

Specify the title.

```
   title 'Printing Salaries for a Check';
run;
```

Output

Output 23.13 *Printing Salaries for a Check*



Example 11: Creating a Format in a non-English Language

Features: PICTURE statement options:
DATATYPE=
LANGUAGE=

Other features: LOCALE= system option

Details

This example does the following:

- 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 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 langtsfr prints the datetime in French.For the remaining directives, see the PICTURE statement on page 639.

```
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 LANGSTSFR 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

```
148  proc format;
149     picture mdy(default=8) other='%0d%0m%Y' (datatype=date);
NOTE: Format MDY has been output.
150     picture langtsda (default=50) other='%A, %d %B, %Y' (datatype=date);
NOTE: Format LANGTSDA has been output.
151     picture langtsdt (default=50) other='%A, %d,%B, %Y %H %M %S'
(datatype=datetime);
NOTE: Format LANGTSDT has been output.
152     picture langtsfr (default=50) other='%A, %d %B, %Y %H %M %S'
(datatype=datetime
152! language=french);
NOTE: Format LANGTSFR has been output.
153     picture alltest (default=100)
154             other='%a %A %b %B %d %H %I %j %m %M %p %S %w %U %y %%'
(datatype=datetime);
NOTE: Format ALLTEST has been output.
155  run;

NOTE: PROCEDURE FORMAT used (Total process time):
      real time           0.01 seconds
      cpu time            0.00 seconds


156
157  option locale = de_DE ;
158  data _null_ ;
159     a= 18903;
160     b = 1633239000;
161     put a= mdy.;
162     put a= langtsda.;
163     put b= langtsdt.;
164     put b= langtsfr.;
165     put b= alltest.;
166  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 12: 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 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

```
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 13: Creating a Format for Trafficlighting

Features: PROC FORMAT statement:
VALUE

Other features: PROC PRINT statement:
VAR statement STYLE option

Details

This example expands on "Example 4: Writing a Format for Dates Using a Standard SAS Format" on page 677 to use trafficlighting. If an employee is eligible for the benefit, the table cell background for the hire date is green. If the employee is not eligible for the benefit, the table cell background hire date is red.

To create the trafficlighting, the program is enhanced by adding these statements:

- In the FORMAT procedure, a VALUE= statement creates a format to specify which values receive the red or green table cell background.
- In the PRINT procedure, a VAR statement for the hiredate variable specifies the background format.

Program

```
proc format library=library;
   value color
      low-'31DEC1999'd='green'
         '01JAN2000'd-high='red';
run;

proc print data=proclib.staff noobs;
   var name idnumber;
   var hiredate / style={background=color.};

   title 'PROCLIB.STAFF Using the COLOR. Format'
run;
```

Program Description

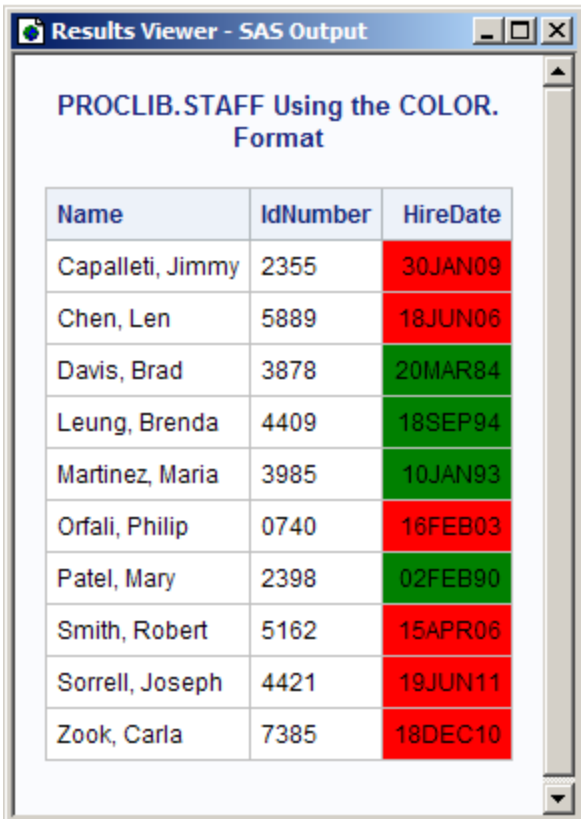
Create the COLOR. format. Hire dates before January 1, 2000 will display l background color of green. Dates beginning with January 1, 2000 and later will display a background of red.

```
proc format library=library;
   value color
      low-'31DEC1999'd='green'
         '01JAN2000'd-high='red';
run;
```

Use the COLOR. format for the hiredate variable. Use two VAR statements to specify the variables to print. The first VAR statement identifies the variables that are not formatted with the COLOR. format. In the second VAR statement, you specify the COLOR. format using the STYLE option BACKGROUND attribute.

```
proc print data=proclib.staff noobs;
   var name idnumber;
   var hiredate / style={background=color.};

   title 'PROCLIB.STAFF Using the COLOR. Format'
run;
```

Output 23.14 *Trafficlighting Using the COLOR. Format*



Name	IdNumber	HireDate
Capalleti, Jimmy	2355	30JAN09
Chen, Len	5889	18JUN06
Davis, Brad	3878	20MAR84
Leung, Brenda	4409	18SEP94
Martinez, Maria	3985	10JAN93
Orfali, Philip	0740	16FEB03
Patel, Mary	2398	02FEB90
Smith, Robert	5162	15APR06
Sorrell, Joseph	4421	19JUN11
Zook, Carla	7385	18DEC10

Example 14: Using a Format to Create a Drill-down Table

Features: VALUE statement

Other features: 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 Web site. 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 Web site.

```
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.

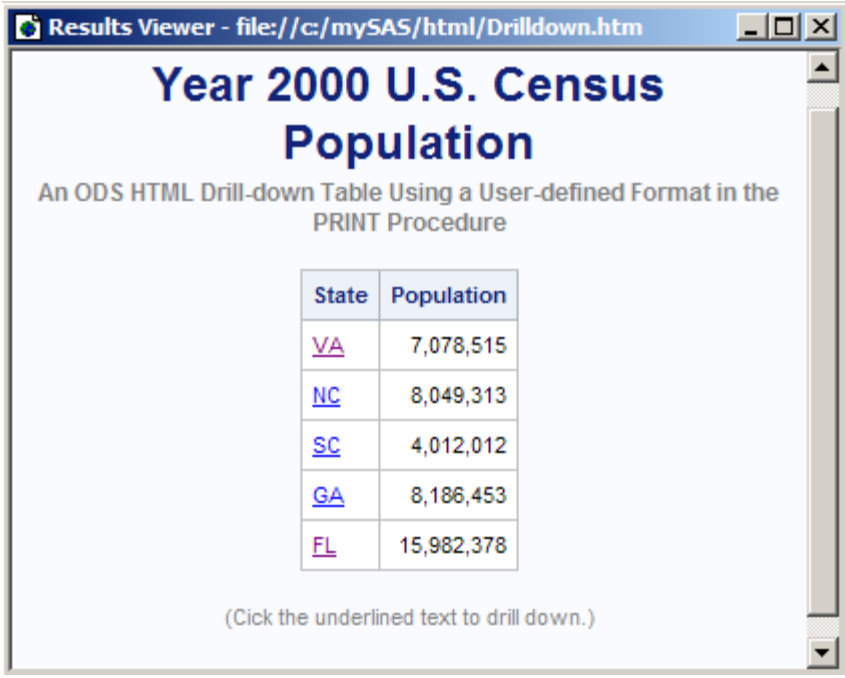
```
ods html file="c:\mySAS\html\Drilldown.htm"
 (title="An ODS HTML Drill-down Table Using a User-defined Format in the PRINT
 Procedure");

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 Web site. 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 23.15 *Using a Format to Create Drill-down Text in an HTML Table*



Chapter 24

FSLIST Procedure

Overview: FSLIST Procedure

The FSLIST procedure enables you to browse, within a SAS session, external files that are not SAS data sets. Because the files are displayed in an interactive window, the procedure provides a highly convenient mechanism for examining file contents. In addition, you can copy text from the FSLIST window into any window that uses the SAS Text Editor.

Syntax: FSLIST Procedure

PROC FSLIST FILEREF=*file-specification*|UNIT=*nn* <*options*>;
PROC FSLIST DDNAME=*file-specification*|UNIT=*nn* <*options*>;
PROC FSLIST DD=*file-specification*|UNIT=*nn* <*options*>;

Statement	Task
"PROC FSLIST Statement"	Initiate the FSLIST procedure and specify the external file to browse

PROC FSLIST Statement

Enables you to browse external files that are not SAS data sets within a SAS session.

Syntax

PROC FSLIST FILEREF=*file-specification* | UNIT=*nn* <*options*>;
PROC FSLIST DDNAME=*file-specification* | UNIT=*nn* <*options*>;
PROC FSLIST DD=*file-specification* | UNIT=*nn* <*options*>;

Summary of Optional Arguments

CAPS | NOCAPS
: controls how search strings for the FIND command are treated.

CC | FORTCC | NOCC
: indicates whether carriage-control characters are used to format the display.

HSCROLL=*n* | HALF | PAGE
: indicates the default horizontal scroll amount for the LEFT and RIGHT commands.

NOBORDER
: suppresses the sides and bottom of the FSLIST window's border.

NUM | NONUM
: controls the display of line sequence numbers.

OVP | NOOVP
: indicates whether the carriage-control code for overprinting is in effect.

Required Arguments

FILEREF | DDNAME | DD=*file-specification*
specifies the external file to browse. *file-specification* can be one of the following:

'external-file'
: is the complete operating environment file specification (called the fully qualified pathname under some operating environments) for the external file. You must enclose the name in quotation marks.

fileref
: is a fileref that has been previously assigned to the external file. You can use the FILENAME statement to associate a fileref with an actual filename. For information about the FILENAME statement, see the section on statements in *SAS Statements: Reference*.

UNIT=*nn*
defines the FORTRAN-style logical unit number of the external file to browse. This option is useful when the file to browse has a fileref of the form FT*nn*F001, where *nn* is the logical unit number that is specified in the UNIT= argument. For example, you can specify the following: `proc fslist unit=20;` instead of `proc fslist fileref=ft20f001;`

Optional Arguments

CAPS | NOCAPS
controls how search strings for the FIND command are treated:

CAPS
converts search strings into uppercase unless they are enclosed in quotation marks. For example, with this option in effect, the command **`find nc`**locates occurrences of **`NC`**, but not **`nc`**. To locate lowercase characters, enclose the search string in quotation marks: **`find 'nc'`**

NOCAPS
does not perform a translation; the FIND command locates only those text strings that exactly match the search string.

The default is NOCAPS. You can use the CAPS command in the FSLIST window to change the behavior of the procedure while you are browsing a file.

CC | FORTCC | NOCC
indicates whether carriage-control characters are used to format the display. You can specify one of the following values for this option:

CC
uses the native carriage-control characters of the operating environment.

FORTCC
uses FORTRAN-style carriage control. The first column of each line in the external file is not displayed; the character in this column is interpreted as a carriage-control code. The FSLIST procedure recognizes the following carriage-control characters:

+
skip zero lines and print (overprint)

blank
skip one line and print (single space)

0
skip two lines and print (double space)

-
skip three lines and print (triple space)

1
go to new page and print.

NOCC
treats carriage-control characters as regular text.

If the FSLIST procedure can determine from the file's attributes that the file contains carriage-control information, then that carriage-control information is used to format the displayed text (the CC option is the default). Otherwise, the entire contents of the file are treated as text (the NOCC option the default).

Note: Under some operating environments, FORTRAN-style carriage control is the native carriage control. For these environments, the FORTCC and CC options produce the same behavior.

If the FSLIST procedure can determine from the file's attributes that the file contains carriage-control information, then that carriage-control information is used to format the displayed text. In this case the CC option is the default. Otherwise, the entire contents of the file are treated as text. In this case the NOCC option is the default.

HSCROLL=*n* | HALF | PAGE
indicates the default horizontal scroll amount for the LEFT and RIGHT commands. The following values are valid:

n
sets the default scroll amount to *n* columns.

HALF
sets the default scroll amount to half the window width.

PAGE
sets the default scroll amount to the full window width.

The default is HSCROLL=HALF. You can use the HSCROLL command in the FSLIST window to change the default scroll amount.

NOBORDER
suppresses the sides and bottom of the FSLIST window's border. When this option is used, text can appear in the columns and row that are normally occupied by the border.

NUM | NONUM
controls the display of line sequence numbers in files that have a record length of 80 and contain sequence numbers in columns 73 through 80. NUM displays the line sequence numbers; NONUM suppresses them. The default is NONUM.

OVP | NOOVP
indicates whether the carriage-control code for overprinting is in effect:

OVP
causes the procedure to honor the overprint code and print the current line over the previous line when the code is encountered.

NOOVP
causes the procedure to ignore the overprint code and print each line from the file on a separate line of the display.

FSLIST Command

Initiates an FSLIST session from any SAS window. The command enables you to use either a fileref or a filename to specify the file to browse. It also enables you to specify how carriage-control information is interpreted.

Syntax

FSLIST <* | ? | *file-specification*<*carriage-control-option*<*overprinting-option*>>>

Without Arguments

If you do not specify any of these three arguments, then a selection window appears that enables you to select an external filename.

Required Arguments

*****

opens a dialog box in which you can specify the name of the file to browse, along with various FSLIST procedure options. In the dialog box, you can specify either a physical filename, a fileref, or a directory name. If you specify a directory name, then a selection list of the files in the directory appears, from which you can choose the desired file.

?

opens a selection window from which you can choose the external file to browse. The selection list in the window includes all external files that are identified in the current SAS session (all files with defined filerefs).

To select a file, position the cursor on the corresponding fileref and press ENTER.

Notes:

Only filerefs that are defined within the current SAS session appear in the selection list. Under some operating environments, it is possible to allocate filerefs outside of SAS. Such filerefs do not appear in the selection list that is displayed by the FSLIST command.

The selection window is not opened if no filerefs have been defined in the current SAS session. Instead, an error message is printed, instructing you to enter a filename with the FSLIST command.

file-specification

identifies the external file to browse. *file-specification* can be one of the following:

'external-file'

the complete operating environment file specification (called the fully qualified pathname under some operating environments) for the external file. You must enclose the name in quotation marks.

If the specified file is not found, then a selection window appears that shows all available filerefs.

fileref

a fileref that is currently assigned to an external file. If you specify a fileref that is not currently defined, then a selection window appears that shows all available filerefs. An error message in the selection window indicates that the specified fileref is not defined.

Optional Arguments

If you specify *file-specification* with the FSLIST command, then you can also use the following options. These options are not valid with the ? argument, or when no argument is used:

CC | FORTCC | NOCC

indicates whether carriage-control characters are used to format the display.

If the FSLIST procedure can determine from the file's attributes that the file contains carriage-control information, then that carriage-control information is used to format the displayed text (the CC option is the default). Otherwise, the entire contents of the file are treated as text (the NOCC option is the default).

You can specify one of the following values for this option:

CC
uses the native carriage-control characters of the operating environment.

FORTCC
uses FORTRAN-style carriage control. See the discussion of the PROC FSLIST statement's FORTCC option for details.

NOCC
treats carriage-control characters as regular text.

OVP | NOOVP
indicates whether the carriage-control code for overprinting is honored. OVP causes the overprint code to be honored; NOOVP causes it to be ignored. The default is NOOVP. The OVP option is ignored if NOCC is in effect.

Using the FSLIST Window

Overview of the FSLIST Window

The FSLIST window displays files for browsing only. You cannot edit files in the FSLIST window. However, you can copy text from the FSLIST window into a paste buffer by doing one of the following, depending on your operating environment:

- Use a mouse to select text, and select **Copy** from the **Edit** menu.
- Use the global MARK and STORE commands.

Depending on your operating environment, the text that you copy can then be pasted into any SAS window that uses the SAS text editor, including the FSLETTER window in SAS/FSP software, or into any other application that allows pasting of text.

You can use commands in the command window or command line to control the FSLIST window.

FSLIST Window Commands

Global Commands

In the FSLIST window, you can use any of the global commands that are described in the "Global Commands" chapter in the *SAS/FSP Procedures Guide*.

Scrolling Commands

n
scrolls the window so that line *n* of text is at the top of the window. Type the desired line number in the command window or on the command line and press ENTER. If *n* is greater than the number of lines in the file, then the last few lines of the file are displayed at the top of the window.

BACKWARD <*n*|HALF|PAGE|MAX>
scrolls vertically toward the first line of the file. The following scroll amounts can be specified:

n
scrolls upward by the specified number of lines.

HALF
scrolls upward by half the number of lines in the window.

PAGE
scrolls upward by the number of lines in the window.

MAX
scrolls upward until the first line of the file is displayed.

If the scroll amount is not explicitly specified, then the window is scrolled by the amount that was specified in the most recent VSCROLL command. The default VSCROLL amount is PAGE.

BOTTOM
scrolls downward until the last line of the file is displayed.

FORWARD <*n*|HALF|PAGE|MAX>
scrolls vertically toward the end of the file. The following scroll amounts can be specified:

n
scrolls downward by the specified number of lines.

HALF
scrolls downward by half the number of lines in the window.

PAGE
scrolls downward by the number of lines in the window.

MAX
scrolls downward until the first line of the file is displayed.

If the scroll amount is not explicitly specified, then the window is scrolled by the amount that was specified in the most recent VSCROLL command. The default VSCROLL amount is PAGE. Regardless of the scroll amount, this command does not scroll beyond the last line of the file.

HSCROLL <*n*|HALF|PAGE>
sets the default horizontal scrolling amount for the LEFT and RIGHT commands. The following scroll amounts can be specified:

n
sets the default scroll amount to the specified number of columns.

HALF
sets the default scroll amount to half the number of columns in the window.

PAGE
sets the default scroll amount to the number of columns in the window.

The default HSCROLL amount is HALF.

LEFT <*n*|HALF|PAGE|MAX>
scrolls horizontally toward the left margin of the text. This command is ignored unless the file width is greater than the window width. The following scroll amounts can be specified:

n
scrolls left by the specified number of columns.

HALF
scrolls left by half the number of columns in the window.

PAGE
scrolls left by the number of columns in the window.

MAX
scrolls left until the left margin of the text is displayed at the left edge of the window.

If the scroll amount is not explicitly specified, then the window is scrolled by the amount that was specified in the most recent HSCROLL command. The default HSCROLL amount is HALF. Regardless of the scroll amount, this command does not scroll beyond the left margin of the text.

RIGHT <*n*|HALF|PAGE|MAX>
scrolls horizontally toward the right margin of the text. This command is ignored unless the file width is greater than the window width. The following scroll amounts can be specified:

n
scrolls right by the specified number of columns.

HALF
scrolls right by half the number of columns in the window.

PAGE
scrolls right by the number of columns in the window.

MAX
scrolls right until the right margin of the text is displayed at the left edge of the window.

If the scroll amount is not explicitly specified, then the window is scrolled by the amount that was specified in the most recent HSCROLL command. The default HSCROLL amount is HALF. Regardless of the scroll amount, this command does not scroll beyond the right margin of the text.

TOP
scrolls upward until the first line of text from the file is displayed.

VSCROLL <*n*|HALF|PAGE>
sets the default vertical scrolling amount for the FORWARD and BACKWARD commands. The following scroll amounts can be specified:

n
sets the default scroll amount to the specified number of lines.

HALF
sets the default scroll amount to half the number of lines in the window.

PAGE
sets the default scroll amount to the number of lines in the window.

The default VSCROLL amount is PAGE.

Searching Commands

BFIND <*search-string* <PREFIX|SUFFIX|WORD>>
locates the previous occurrence of the specified string in the file, starting at the current cursor position and proceeding backward toward the beginning of the file. The *search-string* value must be enclosed in quotation marks if it contains embedded blanks.

If a FIND command has previously been issued, then you can use the BFIND command without arguments to repeat the search in the opposite direction.

The CAPS option in the PROC FSLIST statement and the CAPS ON command cause search strings to be converted to uppercase for the purposes of the search,

unless the strings are enclosed in quotation marks. See the discussion of the FIND command for details.

By default, the BFIND command locates any occurrence of the specified string, even where the string is embedded in other strings. You can use any one of the following options to alter the command's behavior:

PREFIX
causes the search string to match the text string only when the text string occurs at the beginning of a word.

SUFFIX
causes the search string to match the text string only when the text string occurs at the end of a word.

WORD
causes the search string to match the text string only when the text string is a distinct word.

You can use the RFIND command to repeat the most recent BFIND command.

CAPS <ON|OFF>
controls how the FIND, BFIND, and RFIND commands locate matches for a search string. By default, the FIND, BFIND, and RFIND commands locate only those text strings that exactly match the search string as it was entered. When you issue the CAPS command, the FIND, BFIND, and RFIND commands convert search strings into uppercase for the purposes of searching (displayed text is not affected), unless the strings are enclosed in quotation marks. Strings in quotation marks are not affected.

For example, after you issue a CAPS ON command, both of the following commands locate occurrences of **NC** but not occurrences of **nc**: **find NC find nc**If you omit the ON or OFF argument, then the CAPS command acts as a toggle, turning the attribute on if it was off or off if it was on.

FIND *search-string* <NEXT|FIRST|LAST|PREV|ALL> <PREFIX|SUFFIX|WORD>
locates an occurrence of the specified *search-string* in the file. The *search-string* must be enclosed in quotation marks if it contains embedded blanks.

The text in the *search-string* must match the text in the file in terms of both characters and case. For example, the command **find raleigh**will locate not the text **Raleigh** in the file. You must instead use **find Raleigh**.

When the CAPS option is used with the PROC FSLIST statement or when a CAPS ON command is issued in the window, the search string is converted to uppercase for the purposes of the search, unless the string is enclosed in quotation marks. In that case, the command **find raleigh**will locate only the text **RALEIGH** in the file. You must instead use the command **find 'Raleigh'** to locate the text **Raleigh**.

You can modify the behavior of the FIND command by adding any one of the following options:

ALL
reports the total number of occurrences of the string in the file in the window's message line and moves the cursor to the first occurrence.

FIRST
moves the cursor to the first occurrence of the string in the file.

LAST
moves the cursor to the last occurrence of the string in the file.

NEXT
moves the cursor to the next occurrence of the string in the file.

PREV
moves the cursor to the previous occurrence of the string in the file.

The default option is NEXT.

By default, the FIND command locates any occurrence of the specified string, even where the string is embedded in other strings. You can use any one of the following options to alter the command's behavior:

PREFIX
causes the search string to match the text string only when the text string occurs at the beginning of a word.

SUFFIX
causes the search string to match the text string only when the text string occurs at the end of a word.

WORD
causes the search string to match the text string only when the text string is a distinct word.

After you issue a FIND command, you can use the RFIND command to repeat the search for the next occurrence of the string, or you can use the BFIND command to repeat the search for the previous occurrence.

RFIND
repeats the most recent FIND command, starting at the current cursor position and proceeding forward toward the end of the file.

Display Commands

COLUMN <ON|OFF>
displays a column ruler below the message line in the FSLIST window. The ruler is helpful when you need to determine the column in which a particular character is located. If you omit the ON or OFF specification, then the COLUMN command acts as a toggle, turning the ruler on if it was off and off if it was on.

HEX <ON|OFF>
controls the special hexadecimal display format of the FSLIST window. When the hexadecimal format is turned on, each line of characters from the file occupies three lines of the display. The first is the line displayed as characters; the next two lines of the display show the hexadecimal value of the operating environment's character codes for the characters in the line of text. The hexadecimal values are displayed vertically, with the most significant byte on top. If you omit the ON or OFF specification, then the HEX command acts as a toggle, turning the hexadecimal format on if it was off and off if it was on.

NUMS <ON|OFF>
controls whether line numbers are shown at the left side of the window. By default, line numbers are not displayed. If line numbers are turned on, then they remain at the left side of the display when text in the window is scrolled right and left. If you omit the ON or OFF argument, then the NUMS command acts as a toggle, turning line numbering on if it was off or off if it was on.

Other Commands

BROWSE *fileref*|'*actual-filename*' <CC|FORTCC|NOCC <OVP|NOOVP>>
closes the current file and displays the specified file in the FSVIEW window. You can specify either a fileref previously associated with a file or an actual filename

enclosed in quotation marks. The BROWSE command also accepts the same carriage-control options as the FSLIST command. See "Optional Arguments" on page 709 for details.

END
closes the FSLIST window and ends the FSLIST session.

HELP <*command*>
opens a Help window that provides information about the FSLIST procedure and about the commands available in the FSLIST window. To get information about a specific FSLIST window command, follow the HELP command with the name of the desired command.

KEYS
opens the KEYS window for browsing and editing function key definitions for the FSLIST window. The default key definitions for the FSLIST window are stored in the FSLIST.KEYS entry in the SASHELP.FSP catalog.

If you change any key definitions in the KEYS window, then a new FSLIST.KEYS entry is created in your personal PROFILE catalog (SASUSER.PROFILE, or WORK.PROFILE if the SASUSER library is not allocated).

When the FSLIST procedure is initiated, it looks for function key definitions first in the FSLIST.KEYS entry in your personal PROFILE catalog. If that entry does not exist, then the default entry in the SASHELP.FSP catalog is used.

Chapter 25

GROOVY Procedure

Overview: GROOVY Procedure

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 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.

Note:

- 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 output by Java to the SAS log is reserved by SAS. If you need to output a percent character in the first byte of a Java text line, then you must immediately follow it with another percent character (%%).

Syntax: GROOVY Procedure

Restriction: Does not run if the NOXCMD option is turned on.

PROC GROOVY <*classpath options*>;
  ADD *classpath options*;
  EVALUATE <(LOAD | PARSEONLY | NORUN)> *"Groovy statement string"* <*arguments*>;
  EXECUTE <(LOAD | PARSEONLY | NORUN)> *Groovy file name* | *fileref* <*arguments*>;
  SUBMIT <(LOAD | PARSEONLY | NORUN)> <*arguments*>;
    Groovy statements
  ENDSUBMIT;
  CLEAR;
QUIT;

Statement	Task	Example
"PROC GROOVY Statement"	Enable SAS code to run Groovy code on the JVM	Ex. 1

PROC GROOVY Statement

Enables SAS code to run Groovy code on the JVM.

Syntax

PROC GROOVY <*classpath options*>;

Optional Argument

classpath options
can be one of the following:

CLASSPATH=
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 jars 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 at the time the process started
2. paths added with the ADD CLASSPATH and ADD SASJAR statements in the order in which they were executed

ADD Command

Syntax

ADD *classpath options* ;

Required Arguments

classpath options
can be one of the following:

CLASSPATH=
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 jars 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 command 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 Command

Syntax

EVALUATE <(LOAD | PARSEONLY | NORUN)> "*Groovy statement string*" <*arguments*>;

Required Argument

Groovy statement string
specifies a Groovy statement string that is to be parsed by the EVALUATE command.

Optional Arguments

LOAD | PARSEONLY | NORUN
parses the Groovy statement into a groovy.lang.Script object, but does not run it. The arguments are aliases for each other.

arguments
specifies arguments that are passed to the code that is being evaluated.

Details

The EVALUATE command 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 command 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 command.

EXECUTE Command

Syntax

EXECUTE <(LOAD | PARSEONLY | NORUN)> *Groovy file name* | *fileref* <*arguments*>;

Required Arguments

Groovy file name
specifies the name of the Groovy file that is to be parsed by the EXECUTE command.

fileref
specifies the name of a fileref that is to be parsed by the EXECUTE command.

Optional Arguments

LOAD | PARSEONLY | NORUN
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.

arguments
specifies arguments that are passed to the code that is being executed.

Details

The EXECUTE command 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 command 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 command.

Note: If you used an EXEC PARSEONLY command to compile a file into a Class, then you must submit a CLASS command so that changes to that file are honored by future EXEC PARSEONLY commands. If you do not submit the CLEAR command, then any changes that you made to the file after you issued the EXEC PARSEONLY command are not included by subsequent submissions of the EXEC PARSEONLY command. You can use the GroovyScriptEngine Class if you need to use reloadable scripts.

SUBMIT Command

Syntax

SUBMIT <(LOAD | PARSEONLY | NORUN)> <*arguments*>;
Groovy statements
ENDSUBMIT;

Required Argument

Groovy statements
specifies Groovy statements that are to be parsed by the SUBMIT command into a groovy.lang.Script object.

Optional Arguments

LOAD | PARSEONLY | NORUN
parses the Groovy statements into a groovy.lang.Script object, but does not run it. The arguments are aliases for each other.

arguments
specifies arguments that are passed to the code that is being submitted.

Details

The SUBMIT command 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 command 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 Command

Syntax

ENDSUBMIT;

Details

Ends the Groovy statements that begin with the SUBMIT command.

Note: The ENDSUBMIT statement must be on a line by itself, and preceded by only blank space.

CLEAR Command

Syntax

CLEAR;

Details

The CLEAR command 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 command.

Note: Neither the CLEAR command nor the RESET command resets the System.Properties collection or the CLASSPATH.

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 command.

```
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.

```
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.

```
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 command before changes that were made to the execution.groovy file in this example are reflected in subsequent runs of the code.

```
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.

Example: Define Classes

Features: PROC GROOVY statement option
CLASSPATH
SUBMIT command option
PARSEONLY
SUBMIT command
ENDSUBMIT command

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;
```

Chapter 26

HADOOP Procedure

Overview: HADOOP Procedure

PROC HADOOP enables SAS to run Apache Hadoop code against Hadoop data. Apache Hadoop is an open-source technology, written in Java, that provides data storage and distributed 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
- Pig language code

Concepts: HADOOP Procedure

Using PROC HADOOP

The HADOOP procedure enables you to submit HDFS commands, MapReduce programs, and Pig language code against Hadoop data. To connect to the Hadoop server, a Hadoop configuration file is required that specifies the name and JobTracker addresses for the specific server. **fs.default.name** is the URI (protocol specifier, host name, and port) that describes the NameNode for the cluster. Here is an example of a Hadoop configuration file:

```
<configuration>
   <property>
      <name>fs.default.name</name>
      <value>hdfs://xxx.us.company.com:8020</value>
   </property>
   <property>
      <name>mapred.job.tracker</name>
      <value>xxx.us.company.com:8021</value>
   </property>
</configuration>
```

The PROC HADOOP statement supports several options that control access to the Hadoop server. For example, you can identify the Hadoop configuration file and specify the user ID and password on the Hadoop server. You can specify the server options on all PROC HADOOP statements. For the list of server options, see "Hadoop Server Options" on page 729.

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 and provide access to the data to many clients that are distributed across a network.

The PROC HADOOP HDFS statement submits HDFS commands to the Hadoop server that are like the Hadoop shell commands that interact with HDFS and manipulate files. For the list of HDFS commands, see "HDFS Statement" on page 730.

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: the Map function that separates the data to be processed into independent chunks and the Reduce function that performs analysis on that data.

The PROC HADOOP MAPREDUCE statement submits a MapReduce program into a Hadoop cluster. See "MAPREDUCE Statement" on page 731.

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. See "PIG Statement" on page 733.

Syntax: HADOOP Procedure

Restriction: PROC HADOOP is not supported in the z/OS operating environment.

Requirement: Java Runtime Environment (JRE) 1.6

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*>;

Statement	Task	Example
"PROC HADOOP Statement"	Control access to the Hadoop server	Ex. 1, Ex. 2, Ex. 3
"HDFS Statement"	Submit Hadoop Distributed File System (HDFS) commands	Ex. 1
"MAPREDUCE Statement"	Submit MapReduce programs into a Hadoop cluster	Ex. 2
"PIG Statement"	Submit Pig language code into a Hadoop cluster	Ex. 3

PROC HADOOP Statement

Controls access to the Hadoop server.

Syntax

PROC HADOOP <*hadoop-server-options*>;

Hadoop Server Options

These options control access to the Hadoop server and can be specified on all HADOOP procedure statements.

AUTHDOMAIN=*auth-domain*
specifies the name of an authentication domain metadata object in order to connect to the Hadoop server. The authentication domain references credentials (user ID and

password) without your having to explicitly specify the credentials. The *auth-domain* name is case sensitive, and it must be enclosed in double quotation marks.

An administrator creates authentication domain definitions while creating a user definition with the User Manager in SAS Management Console. The authentication domain is associated with one or more login metadata objects that provide access to the Hadoop server. The metadata objects are resolved by SAS calling the SAS Metadata Server and returning the authentication credentials.

Requirement: The authentication domain and the associated login definition must be stored in a metadata repository, and the metadata server must be running to resolve the metadata object specification.

Interaction: If you specify AUTHDOMAIN=, do not specify USERNAME= and PASSWORD=.

See: For more information about creating and using authentication domains, see the discussion on credential management in the *SAS Intelligence Platform: Security Administration Guide*.

OPTIONS=*fileref* | '*external-file*'
identifies the Hadoop configuration file to use for the associated PROC statement. The file must be an XML document.

fileref
specifies the SAS fileref that is assigned to the Hadoop configuration file. To assign a fileref, use the FILENAME statement.

'*external-file*'
is the physical location of the 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.

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 OPTIONS=.

Requirement: 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 OPTIONS=.

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.

HDFS Statement

Submits Hadoop Distributed File System (HDFS) commands.

Restriction: The HDFS statement supports only one operation per invocation.

Example: "Example 1: Submitting HDFS Commands" on page 733

Syntax

HDFS <*hadoop-server-options*> <*hdfs-command-options*>;

HDFS Command Options

These options support commands that interact with the HDFS. Include only one operation per HDFS statement.

COPYFROMLOCAL='*local-file*'
copies the specified local file to an HDFS path output location. Specify the complete pathname and filename.

Requirement: Use the OUT= option to specify the HDFS path output location.

COPYTOLOCAL='*HDFS-file*'
copies the specified HDFS file to a local file output location. Specify the complete HDFS directory and filename.

Requirement: Use the OUT= option to specify the local file output location.

DELETE='*HDFS-file*'
deletes the specified HDFS file. Specify the complete HDFS directory and filename.

DELETESOURCE
deletes the input source file after a copy command.

Restriction: Use DELETESOURCE with the COPYFROMLOCAL= or COPYTOLOCAL= options.

KEEPCRC
saves the Cyclic Redundancy Check (CRC) file after a copy command to a local file 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.

Default: The CRC file is deleted.

Restriction: Use KEEPCRC with the COPYTOLOCAL= option.

MKDIR='*HDFS-path*'
creates the specified HDFS path. Specify the complete HDFS directory.

OUT='*output-location*'
specifies the output location for an HDFS operation. When copying a local file to HDFS, specify the HDFS path. When copying an HDFS file to a local file, specify the external file for your machine. When renaming an HDFS file, specify the new HDFS path and filename.

OVERWRITE
overwrites the output file after a copy command.

Restriction: Use OVERWRITE with the COPYFROMLOCAL= or COPYTOLOCAL= options.

RENAME='*HDFS-file*'
renames the specified HDFS file. Specify the complete HDFS directory and filename.

Requirement: Use the OUT= option to specify the new HDFS path and filename.

MAPREDUCE Statement

Submits MapReduce programs into a Hadoop cluster.

Example: "Example 2: Submitting a MapReduce Program" on page 734

Syntax

MAPREDUCE <*hadoop-server-options*> <*mapreduce-options*>;

MapReduce Options

COMBINE=*class-name*
specifies the name of the combiner class in dot notation.

DELETERESULTS
deletes the MapReduce results.

GROUPCOMPARE=*class-name*
specifies the name of the grouping comparator (GroupComparator) class in dot notation.

INPUT=*HDFS-path*
specifies the HDFS path 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. 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-path*
specifies a new HDFS path 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. 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.

WORKINGDIR=*HDFS-path*
specifies the name of the HDFS working directory path.

PIG Statement

Submits Pig language code into a Hadoop cluster.

Example: "Example 3: Submitting Pig Language Code" on page 736

Syntax

PIG <*hadoop-server-options*> <*pig-code-options*>;

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. Include the complete pathname and the filename. Enclose the physical name in single or double quotation marks.

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. Include the complete pathname and the filename. 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. Include the complete pathname and the filename. Enclose each location in single or double quotation marks.

Examples: HADOOP Procedure

Example 1: Submitting HDFS Commands

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

```
filename cfg "C:\Users\sasabc\hadoop\sample_config.xml";

proc hadoop options=cfg 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

Assign a file reference to the Hadoop configuration file. The FILENAME statement assigns the file reference CFG to the physical location of a Hadoop configuration file that is named sample_config.xml, which is shown in "Using PROC HADOOP" on page 728.

```
filename cfg "C:\Users\sasabc\hadoop\sample_config.xml";
```

Execute the PROC HADOOP statement. The PROC HADOOP statement controls access to the Hadoop server by referencing the Hadoop configuration file with the OPTIONS= option, identifying the user ID and password on the Hadoop server, and specifying the option VERBOSE, which enables additional messages to the SAS log.

```
proc hadoop options=cfg username="sasabc" password="sasabc" verbose;
```

Create an HDFS path. The first HDFS statement specifies the MKDIR= option to create an HDFS path.

```
   hdfs mkdir="/user/sasabc/new_directory";
```

Delete an HDFS path. The second HDFS statement specifies the DELETE= option to delete an HDFS path.

```
   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 a MapReduce Program

Details

This PROC HADOOP example submits a MapReduce program to a Hadoop server. 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

```
filename cfg 'C:\Users\sasabc\Hadoop\sample_config.xml';

proc hadoop options=cfg username="sasabc" password="sasabc" verbose;

   mapreduce input='/user/sasabc/architectdoc.txt'

      output='/user/sasabc/outputtest'

      jar='C:\Users\sasabc\Hadoop\jars\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

Assign a file reference to the Hadoop configuration file. The FILENAME statement assigns the file reference CFG to the physical location of a Hadoop configuration file that is named sample_config.xml, which is shown in "Using PROC HADOOP" on page 728.

```
filename cfg 'C:\Users\sasabc\Hadoop\sample_config.xml';
```

Execute the PROC HADOOP statement. The PROC HADOOP statement controls access to the Hadoop server by referencing the Hadoop configuration file with the OPTIONS= option, identifying the user ID and password on the Hadoop server, and specifying the option VERBOSE, which enables additional messages to the SAS log.

```
proc hadoop options=cfg username="sasabc" password="sasabc" verbose;
```

Submit a MapReduce program. The MAPREDUCE statement includes several options. INPUT= specifies the HDFS path of the input Hadoop file named architectdoc.txt.

```
   mapreduce input='/user/sasabc/architectdoc.txt'
```

Create an HDFS path. OUTPUT= creates the HDFS path 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='C:\Users\sasabc\Hadoop\jars\WordCount.jar'
```

Specify an output key class. OUTPUTKEY= specifies the name of the output key class. The org.apache.hadoop.io.Text class stores text using standard UTF8 encoding and provides methods to compare 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 3: Submitting Pig Language Code

Details

This PROC HADOOP example submits Pig language code into a Hadoop cluster. This is the Pig language code to be executed:

```
A = LOAD '/user/sasabc/testdata.txt' USING PigStorage(',')
    AS (customer_number,account_number,tax_id,date_of_birth,status,
        residence_country_code,marital_status,email_address,phone_number,
        annual_income,net_worth_amount,risk_classification);
B = FILTER A BY marital_status == 'Single';
store B into '/user/sasabc/output_customer' USING PigStorage(',');
```

Program

```
filename cfg "C:\Users\sasabc\hadoop\sample_config.xml";

filename code "C:\Users\sasabc\hadoop\sample_pig.txt";

proc hadoop options=cfg username="sasabc" password="sasabc" verbose;

   pig code=code registerjar="C:\Users\sasabc\Hadoop\jars\myudf.jar";
run;
```

Program Description

Assign a file reference to the Hadoop configuration file. The first FILENAME statement assigns the file reference CFG to the physical location of a Hadoop configuration file that is named sample_config.xml, which is shown in "Using PROC HADOOP" on page 728.

```
filename cfg "C:\Users\sasabc\hadoop\sample_config.xml";
```

Assign a file reference to the Pig language code. The second FILENAME statement assigns the file reference CODE to the physical location of the file that contains the Pig language code that is named sample_pig.txt, which is shown above.

```
filename code "C:\Users\sasabc\hadoop\sample_pig.txt";
```

Execute the PROC HADOOP statement. The PROC HADOOP statement controls access to the Hadoop server by referencing the Hadoop configuration file with the OPTIONS= option, identifying the user ID and password on the Hadoop server with the USERNAME= and PASSWORD= options, and specifying the VERBOSE option, which enables additional messages to the SAS log.

```
proc hadoop options=cfg username="sasabc" password="sasabc" verbose;
```

Execute the PIG statement. The PIG statement includes the CODE= option to specify the SAS fileref CODE that is assigned to the physical location of the file that contains the Pig language code and the REGISTERJAR= option to specify the JAR file that contains the Pig scripts to execute.

```
   pig code=code registerjar="C:\Users\sasabc\Hadoop\jars\myudf.jar";
run;
```

Chapter 27

HTTP Procedure

Overview: HTTP Procedure

PROC HTTP issues Hypertext Transfer Protocol (HTTP) requests. PROC HTTP reads as input the entire body from a fileref and writes output to a fileref. PROC HTTP can also read custom request headers from a fileref and write response headers to a fileref.

Syntax: HTTP Procedure

PROC HTTP *options*;

Statement	Task	Example
"PROC HTTP Statement"	Issues HTTP requests	Ex. 1, Ex. 2, Ex. 3, Ex. 4

PROC HTTP Statement

Invokes a Web service that issues requests.

Examples:

Syntax

PROC HTTP *options*;

Summary of Optional Arguments

CT
: specifies the HTTP content-type to be set in the request headers.

HEADERIN
: specifies a fileref to a text file that contains one line per request header in the format *key:value*.

HEADEROUT
: specifies a fileref to a text file to which the response headers are in the format *key:value*.

IN
: specifies a fileref to input data.

METHOD
: specifies the HTTP method.

OUT
: specifies a fileref where output is written.

PROXYHOST
: specifies an HTTP proxy server host name.

PROXYPASSWORD
: specifies an HTTP proxy server password.

PROXYPORT
: specifies an HTTP proxy server port.

PROXYUSERNAME
: specifies an HTTP proxy server user name.

URL
: specifies the endpoint for the HTTP request.

WEBAUTHDOMAIN
: specifies the Web authentication domain.

WEBPASSWORD
: specifies a password for basic authentication.

WEBUSERNAME
: specifies a user name for basic authentication.

Optional Arguments

CT
specifies the HTTP content-type to be set in the request headers.

HEADERIN
specifies a fileref to a text file that contains one line per request header in the format *key:value*.

In the z/OS operating environment, HEADERIN files must be created with a variable record length.

HEADEROUT
specifies a fileref to a text file to which the response headers are in the format *key:value*.

IN
specifies a fileref to input data.

METHOD
specifies the HTTP method.

OUT
specifies a fileref that indicates where output is written.

PROXYHOST
specifies an HTTP proxy server host name.

PROXYPASSWORD
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
specifies an HTTP proxy server port.

PROXYUSERNAME
specifies an HTTP proxy server user name.

Tip: The user name is required only if your proxy server requires credentials.

URL
specifies the endpoint for the HTTP request.

WEBAUTHDOMAIN
specifies the Web authentication domain. If specified, a user name and password are retrieved from metadata for the specified authentication domain.

WEBPASSWORD
specifies a password for basic authentication.

Tip: Encodings that are produced by PROC PWENCODE are supported.

WEBUSERNAME
specifies a user name for basic authentication.

Using Hypertext Transfer Protocol Secure (HTTPS)

HTTP Security: SSL and Data Encryption

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.

Making PROC HTTP Calls by Using the HTTPS Protocol

In order to make PROC HTTP calls by using the HTTPS protocol, you must configure a trust source that contains the certificate of the service to be trusted. This trust source and its password must be provided for the SAS session by setting Java system options using jreoptions. You can provide this information on the SAS command line or in a SAS configuration file. Use the following syntax. Be sure to enter the following entry on one line:

```
-jreoptions
(-Djavax.net.ssl.trustStore=full-path-to-the-trust-store
-Djavax.net.ssl.trustStorePassword=trustStorePassword)
```

The following example shows how to use the entry on the SAS command line. This example uses the Windows operating environment. Be sure to enter the following entry on one line:

```
"C:\Program Files\SAS\SASFoundation\9.2\sas.exe" -CONFIG
"C:\Program Files\SAS\SASFoundation\9.2\nls\en\SASV9.CFG"
-jreoptions (- Djavax.net.ssl.trustStore=C:\Documents and
Settings\mydir\.keystore
-Djavax.net.ssl.trustStorePassword=trustpassword)
```

Using Encodings with PROC HTTP

Responses are not encoded to session encodings. You must supply the request with the encoding you want to use, and set the content type.

Examples: HTTP Procedure

Example 1: A Simple POST Request

Details

This example makes a POST method call to a server on the local network. Parameters to the POST are read from IN and the response is written to OUT.TXT.

Program

```
filename in "u:\prochttp\Testware\in";
filename out "u:\prochttp\Testware\out.txt";
data _null_;
   file in;
   input;
   put _infile_;
   datalines4;

server=localhost&action=list&type=store&user=your-user-name&password
```

```
=your-password&details=true&x=1
;;;;

 proc http in=in out=out url="http://localhost.com/rsm/rstool"
     method="post"
      ct="application/x-www-form-urlencoded";
run;
```

Example 2: A POST Through a Proxy

Details

This example makes a POST method call to an external server and, therefore, requires the use of a proxy server. Parameters to the POST are read from ProxyTest_in and the response is written to ProxyTest_out.txt.

Program

```
filename in "u:\prochttp\Testware\ProxyTest_in";
filename out "u:\prochttp\Testware\ProxyTest_out.txt";
data _null_;
   file in;
   input;
   put _infile_;
   datalines4;
appid=restbook&query=jellyfish
;;;;

proc http
   in=in
   out=out
   url="http://api.search.yahoo.com/WebSearchService/V1/webSearch?"
   method="post"
   ct="application/x-www-form-urlencoded"
   proxyhost="proxygw.abc.sas.com"
   proxyport=80;
run;
```

Example 3: A POST Through a Proxy That Requires Authentication

Details

This example makes the same POST request as in "Example 2: A POST Through a Proxy" on page 743 but uses a proxy server that requires authentication.

Program

```
filename in "u:\prochttp\Testware\ProxyAuthTest_in";
filename out "u:\prochttp\Testware\ProxyAuthTest_out.txt";
data _null_;
   file in;
   input;
```

```
   put _infile_;
   datalines4;
appid=restbook&query=jellyfish
;;;;

proc http
   in=in
   out=out
   url="http://api.search.yahoo.com/WebSearchService/V1/webSearch?"
   method="post"
   ct="application/x-www-form-urlencoded"
   proxyhost="proxyhost.company.com"
   proxyusername="your-user-name"
   proxypassword="your-password"
   proxyport=80;
run;
```

Example 4: A POST That Captures the Response Headers

Details

This example makes the same POST request as in "Example 2: A POST Through a Proxy" on page 743 but captures the response headers in a file called headerOut.txt.

Program

```
filename in "u:\prochttp\Testware\ProxyTest_in";
filename out "u:\prochttp\Testware\ProxyTest_out.txt";
filename hdrout "u:\prochttp\Testware\headerOut.txt";
data _null_;
   file in;
   input;
   put _infile_;
   datalines4;
appid=restbook&query=jellyfish
;;;;

 proc http
    in=in
    out=out
    headerout=hdrout
    url="http://api.search.yahoo.com/WebSearchService/V1/webSearch?"
    method="post"
    ct="application/x-www-form-urlencoded"
    proxyhost="inetgw.unx.sas.com"
    proxyport=80;
run;
```

Chapter 28

IMPORT Procedure

Overview: IMPORT Procedure

PROC IMPORT reads data from an external data source and writes it to a SAS data set. Base SAS can import JMP files and delimited 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 such files as Microsoft Access Database, Microsoft Excel files, and Lotus spreadsheets. See *SAS/ACCESS Interface to PC Files: Reference* for more information.

With JMP files, you can import value labels to a SAS format catalog and metadata information to a data set.

When you run the IMPORT procedure, it reads the input file and writes the data to a SAS data set. The SAS variable definitions are based on the input records. When the IMPORT procedure reads a delimited file, it generates a DATA step code to import the data.

You control the results with options and statements that are specific to the input data source. The IMPORT procedure generates the specified output SAS data set and writes information about the import to the SAS log. The log displays the DATA step code that the IMPORT procedure generates.

To import data, you can also use the Import Wizard or the External File Interface (EFI) to guide you through the steps to import an external data source. You can use the Import

Wizard to generate IMPORT procedure statements, which you can save to a file for subsequent use. To open the Import Wizard or EFI from the SAS windowing environment, select **File ⇨ Import Data**. For more information about the Import Wizard or EFI, see the Base SAS online Help and documentation.

Syntax: IMPORT Procedure

Restriction: The IMPORT procedure is available for the following operating environments:

- Windows
- UNIX

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.

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* | '*nn*'x;

GETNAMES=YES | NO;

GUESSINGROWS=*n*;

statements for importing from JMP files

DBENCODING=*12-byte SAS encoding-value* ;

FMTLIB=<*libref.*>*format-catalog*;

META=*libref.member-data-set*;

Statement	Task	Example
"PROC IMPORT Statement"	Import an external data file to a SAS data set	Ex. 1, Ex. 2, Ex. 3, Ex. 4
"DATAROW Statement"	Start reading data from a specific row in the delimited text file	Ex. 3
"DBENCODING Statement"	Indicate the encoding character set to use for the JMP file	
"DELIMITER Statement"	Specify the delimiter that separates columns of data in the input file	Ex. 1, Ex. 3, Ex. 4
"FMTLIB Statement"	Save value labels to the specified SAS format catalog	
"GETNAMES Statement"	Generate SAS variable names from the data values in the first record in the input file	Ex. 1, Ex. 2

Statement	Task	Example
"GUESSINGROWS Statement"	Specify the number of rows of the input file to scan to determine the appropriate data type and length for the columns	
"META Statement"	Save JMP metadata information to the specified SAS metadata set	

PROC IMPORT Statement

Imports an external data file to a SAS data set.

Syntax

PROC IMPORT

DATAFILE="*filename*" | TABLE="*tablename*"
OUT=<*libref*.>*SAS data set*<(*SAS data set options*)>
<DBMS=*identifier*> <REPLACE> ;

Required Arguments

DATAFILE="*filename*"

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. 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. For more information about the FILENAME statement, see *SAS Statements: Reference*. For more information about PC file formats, see *SAS/ACCESS Interface to PC Files: Reference*.

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. 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:

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 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 scanned using the GUESSINGROWS data source statement. All values are read in as character

strings. If a Date and Time format, or numeric informat can be applied to the data value, the type is declared as numeric. Otherwise, the type remains character.

Examples:

"Example 1: Importing a Delimited File" on page 752

"Example 2: Importing a Specific Delimited File Using a Fileref" on page 755

"Example 3: Importing a Tab-Delimited File" on page 756

"Example 4: Importing a Comma-Delimited File with a CSV Extension" on page 757

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).

Examples:

"Example 1: Importing a Delimited File" on page 752

"Example 2: Importing a Specific Delimited File Using a Fileref" on page 755

"Example 3: Importing a Tab-Delimited File" on page 756

"Example 4: Importing a Comma-Delimited File with a CSV Extension" on page 757

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.

Optional Arguments

DBMS=*identifier*

specifies the type of data to import. You can import JMP files (DBMS=JMP) or delimited files. 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.

See:

Table 18.1 on page 504 for more information about identifiers for this option.

SAS/ACCESS Interface to PC Files: Reference for a list of additional DBMS values when using SAS/ACCESS Interface to PC Files.

Examples:

"Example 1: Importing a Delimited File" on page 752

"Example 2: Importing a Specific Delimited File Using a Fileref" on page 755

"Example 3: Importing a Tab-Delimited File" on page 756

"Example 4: Importing a Comma-Delimited File with a CSV Extension" on page 757

REPLACE
overwrites an existing SAS data set. If you do not specify REPLACE, the IMPORT procedure does not overwrite an existing data set.

Examples:
"Example 1: Importing a Delimited File" on page 752
"Example 2: Importing a Specific Delimited File Using a Fileref" on page 755
"Example 3: Importing a Tab-Delimited File" on page 756
"Example 4: Importing a Comma-Delimited File with a CSV Extension" on page 757

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*.

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 only for delimited files.

See: "GETNAMES Statement" on page 751

Example: "Example 3: Importing a Tab-Delimited File" on page 756

Syntax

DATAROW=*n*;

Required Argument

n
specify the row number in the input file for the IMPORT procedure to start reading data.

Range: 1 to 2147483647

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-byte SAS encoding-value* ;

Required Argument

12-byte 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.

Default: Blank space

Interaction: The DELIMTER statement is valid only when DBMS=DLM.

Example: "Example 1: Importing a Delimited File" on page 752

Syntax

DELIMITER=*char* | '*nn*'x;

Required Argument

***char* | '*nn*'x**
specify 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='&'. If you do not specify DELIMITER=, the IMPORT procedure assumes that the delimiter is a space.

Default: Blank space

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 record in the input file.

Interactions: If the column names in the first record in the input file are not valid SAS names, then the IMPORT procedure uses default variable names.

The GETNAMES statement is valid only for delimited files.

Examples: "Example 1: Importing a Delimited File" on page 752

"Example 2: Importing a Specific Delimited File Using a Fileref" on page 755

"Example 4: Importing a Comma-Delimited File with a CSV Extension" on page 757

Syntax

GETNAMES=YES | NO;

Required Argument

YES | NO

Specify whether SAS variable names should be generated from the first record in the input file.

Note: If a data value in the first record 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 column name **`Occupancy Code`** would become the SAS variable name **`Occupancy_Code`**.

Default: YES

GUESSINGROWS Statement

Specifies the number of rows of the file to scan to determine the appropriate data type and length for the columns.

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*;

Required Argument

n

Indicate the number of rows the IMPORT procedure scans in the input file to determine the appropriate data type and length of columns. The scan data process scans from row 1 to the number that is specified by the GUESSINGROWS option.

Default: 20

Range: 1 to 2147483647

Note: You can change the default value in the SAS Registry under **SAS Registry** ⇨ **Products** ⇨ **Base** ⇨ **EFI** ⇨ **GuessingRows**.

META Statement

Saves JMP metadata information to the specified SAS metadata set.

Interaction: The META statement is valid only when DBMS=JMP.

Syntax

META=*libref.member-data-set*;

Required Argument

libref.member-data-set
specifies the data set where the metadata information is to be written.

Examples: IMPORT Procedure

Example 1: Importing a Delimited File

Features: PROC IMPORT Statement Arguments
DATAFILE=
DBMS=
GETNAMES=
OUT=
REPLACE
DELIMITER=

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

```
/*** Specify the input file.***/
/*** Identify the output SAS data set.***/
    /*** Specify the input file is a delimited file.***/
/*** Replace the data set if it exists.***/
    /*** Specify delimiter as an & (ampersand).***/
/*** Generate variable names from first row of data.***/

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;
```

SAS 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.

```
1    options nodate ps=60 ls=80; proc import datafile="C:\My
1  ! Documents\myfiles\delimiter.txt" dbms=dlm 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.
```

This output lists the output data set, MYDATA, created by the IMPORT procedure from the delimited external file.

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 Arguments
- DATAFILE=
- DBMS=
- GETNAMES=
- OUT=
- REPLACE

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
       getnames=yes;
   out=stateinfo
        replace;
run;
```

```
proc print;
run;
```

Example 3: Importing a Tab-Delimited File

Features: PROC IMPORT Statement Arguments
- DATAFILE=
- DATAROW=
- DBMS=
- OUT=
- REPLACE

Data Source Statement
- DELIMITER=

Details

This example imports the following tab-delimited file and creates a temporary SAS data set named WORK.CLASS. 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**. The first observation read will be observation 5 due to the DATAROW= specification. GETNAMES= defaults to 'yes'.

```
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:\temp\tab.txt'
            out=class
            dbms=dlm
            replace;
       datarow=5;
     delimiter='09'x;
run;
```

Example 4: Importing a Comma-Delimited File with a CSV Extension

Features: PROC IMPORT Statement Arguments
- DATAFILE=
- DBMS=
- GETNAMES=
- OUT=
- REPLACE

Details

This example imports the following comma-delimited file and creates a temporary SAS data set named WORK.SHOES. GETNAME= is set to 'no', so the variable names in record 1 are not used. DATAROW=2 begins reading data from record 2.

```
"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;
run;
```

Chapter 29

JAVAINFO Procedure

Overview: JAVAINFO Procedure

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
"PROC JAVAINFO Statement"	Display diagnostic information about the SAS Java environment

PROC JAVAINFO Statement

Displays diagnostic information about the SAS Java environment.

Syntax

PROC JAVAINFO <*options*>;

Optional Arguments

ALL
specifies current information about the SAS Java environment.

CLASSPATHS
specifies information about the classpaths that Java is using.

HELP
specifies usage assistance in using the JAVAINFO procedure.

JREOPTIONS
specifies 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
specifies information about the operating system that SAS is running under.

version
specifies the Java Runtime Environment (JRE) that SAS is using.

Chapter 30

MEANS Procedure

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 52, "SUMMARY Procedure," on page 1475 for an explanation of the differences.

What Types of Output Does PROC MEANS Produce?

PROC MEANS Default Output

The following output 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 30.1 *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 828.

Output 30.2 *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 Obs	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 combinations of School and Year 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.

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. 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 nonsubdivided 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 30.3 *Ordering Class Values*

```
                The SAS System                              1

            The MEANS Procedure

                                  N
Pet         Gender      Obs
---------------------------
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).

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

$$Nc_1(L\ c_1 + K) + Nc_2(L\ c_2 + K) + \ldots + Nc_n(L\ c_n + K)$$

where

Nc_i

is the number of unique values for the class variable.

$L\ c_i$
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, $L\ c_i$ is simply the unformatted length of c_i.

Each unique combination of class variables, $c_{1_i}\ c_{2_j}$ for a given type forms a level in that type. See "TYPES Statement" on page 792. 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

$$W * Nc_1 * Nc_2 * \ldots * Nc_n$$

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).

$Nc_1 \ldots Nc_n$
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 will 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= will have 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:

- DB2
- Netezza
- Oracle
- Teradata

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 will prevent in-database processing: OBS=, FIRSTOBS=, RENAME=, and DBCONDITION=. For a complete listing, refer to "In-Database Procedures in Teradata" in *SAS/ACCESS for Relational Databases: Reference*.

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 will be 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 database processing, see *SAS/ACCESS for Relational Databases: Reference*.

Syntax: MEANS 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 35. You can also use any global statement. For a list, see "Global Statements" in Chapter 1 of *SAS Statements: Reference*.

PROC MEANS <*option(s)*> <*statistic-keyword(s)*>;
BY <DESCENDING> *variable-1* <… <DESCENDING>*variable-n*><NOTSORTED>;
CLASS *variable(s)* </ *option(s)*>;
FREQ *variable*;
ID *variable(s)*;
OUTPUT <OUT=*SAS-data-set*> <*output-statistic-specification(s)*>
<*id-group-specification(s)*> <*maximum-id-specification(s)*>
<*minimum-id-specification(s)*> </ *option(s)*> ;
TYPES *request(s)*;
VAR *variable(s)* < / WEIGHT=*weight-variable*>;
WAYS *list*;
WEIGHT *variable*;

Statement	Task	Example
"PROC MEANS Statement"	Compute descriptive statistics for variables	Ex. 1, Ex. 2, Ex. 3, Ex. 4, Ex. 5, Ex. 6, Ex. 7, Ex. 8,

Statement	Task	Example
		Ex. 9, Ex. 10, Ex. 12, Ex. 13
"BY Statement"	Calculate separate statistics for each BY group	Ex. 3
"CLASS Statement"	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 Statement"	Identify a variable whose values represent the frequency of each observation	
"ID Statement"	Include additional identification variables in the output data set	
"OUTPUT Statement"	Create an output data set that contains specified statistics and identification variables	Ex. 8, Ex. 9, Ex. 10, Ex. 11, Ex. 12
"TYPES Statement"	Identify specific combinations of class variables to use to subdivide the data	Ex. 2, Ex. 5, Ex. 12
"VAR Statement"	Identify the analysis variables and their order in the results	Ex. 1
"WAYS Statement"	Specify the number of ways to make unique combinations of class variables	Ex. 6
"WEIGHT Statement"	Identify a variable whose values weight each observation in the statistical calculations	Ex. 6

PROC MEANS Statement

Computes descriptive statistics for variables.

See: Chapter 52, "SUMMARY Procedure," on page 1475

Examples:
"Example 1: Computing Specific Descriptive Statistics" on page 801
"Example 2: Computing Descriptive Statistics with Class Variables" on page 803
"Example 3: Using the BY Statement with Class Variables" on page 806
"Example 4: Using a CLASSDATA= Data Set with Class Variables" on page 808
"Example 5: Using Multilabel Value Formats with Class Variables" on page 811
"Example 6: Using Preloaded Formats with Class Variables" on page 816
"Example 7: Computing a Confidence Limit for the Mean" on page 820
"Example 8: Computing Output Statistics" on page 822
"Example 9: Computing Different Output Statistics for Several Variables" on page 824
"Example 10: Computing Output Statistics with Missing Class Variable Values" on page 826

"Example 12: Identifying the Top Three Extreme Values with the Output Statistics" on page 831
"Example 13: Using the STACKODSOUTPUT option to control data" on page 835

Syntax

PROC MEANS <*option(s)*> <*statistic-keyword(s)*>;

Summary of Optional Arguments

DATA=*SAS-data-set*
specifies the input data set.

NOTHREADS
overrides the SAS system option THREADS | NOTHREADS.

NOTRAP
disables floating point exception recovery.

PCTLDEF=
specifies the mathematical definition used to compute quantiles.

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.

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 | NOPRINT
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=*divisor*
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−*value*)×100. For example, 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 796

"Example 7: Computing a Confidence Limit for the Mean" on page 820

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 799

"Example 10: Computing Output Statistics with Missing Class Variable Values" on page 826

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 808

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 816

DATA=*SAS-data-set*

identifies the input SAS data set.

See: "Input Data Sets" on page 20

DESCENDTYPES

orders observations in the output data set by descending _TYPE_ value.

Alias: 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 799

"Example 9: Computing Different Output Statistics for Several Variables" on page 824

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 795

WEIGHT= option on the VAR statement on page 793

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 808

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 801
"Example 4: Using a CLASSDATA= Data Set with Class Variables" on page 808
"Example 5: Using Multilabel Value Formats with Class Variables" on page 811

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 785

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 803
"Example 4: Using a CLASSDATA= Data Set with Class Variables" on page 808

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 Language Reference: Concepts for a discussion of missing values that have special meaning.
"Example 6: Using Preloaded Formats with Class Variables" on page 816

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 799

"Example 5: Using Multilabel Value Formats with Class Variables" on page 811

"Example 6: Using Preloaded Formats with Class Variables" on page 816

NOPRINT

See the `"PRINT | NOPRINT" on page 775 option.

NOTHREADS

See the "THREADS | NOTHREADS" on page 778 option.

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 799

"Example 10: Computing Output Statistics with Missing Class Variable Values" on page 826

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.

Alias: 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.

Alias: UNFMT | INTERNAL

Default: UNFORMATTED

See: "Ordering the Class Values" on page 764

PCTLDEF=
PCTLDEF is an alias for QNTLDEF=.

See: QNTLDEF= option

PRINT | NOPRINT
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 822

"Example 12: Identifying the Top Three Extreme Values with the Output Statistics" on page 831

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 808

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 785

STACKODSOUTPUT
produces an ODS output object whose data set resembles the printed output.

Alias: STACKODS

See: "Example 13: Using the STACKODSOUTPUT option to control data" on page 835

QMARKERS=*number*
specifies the default number of markers to use for the P² 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 798

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 will 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 798

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 1671

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

Descriptive statistic keywords	
CLM	NMISS
CSS	RANGE
CV	SKEWNESS|SKEW
KURTOSIS|KURT	STDDEV|STD

LCLM	STDERR
MAX	SUM
MEAN	SUMWGT
MIN	UCLM
MODE	USS
N	VAR
Quantile statistic keywords	
MEDIAN|P50	Q3|P75
P1	P90
P5	P95
P10	P99
P20	P30
P40	P60
P70	P80
Q1|P25	QRANGE
Hypothesis testing keywords	
PROBT|PRT	T

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 1666.

"Example 1: Computing Specific Descriptive Statistics" on page 801

"Example 3: Using the BY Statement with Class Variables" on page 806

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|*n*K| *n*M| *n*G
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 | NOTHREADS
enables or disables parallel processing of the input data set. This option overrides the SAS system option THREADS | NOTHREADS unless the system option is restricted. (See Restriction.) See "Support for Parallel Processing" in Chapter 13 of *SAS Language Reference: Concepts* for more information.

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=*divisor*
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 30.1 *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	$(\Sigma_i w_i) - 1$
WEIGHT | WGT	Sum of weights	$\Sigma_i w_i$

The procedure computes the variance as $CSS / divisor$, where CSS is the corrected sums of squares and equals $\Sigma (x_i - \bar{x})^2$. When you weight the analysis variables, CSS equals $\Sigma w_i (x_i - \bar{x}_w)^2$, 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 1666

"Weighted Statistics Example" on page 44

BY Statement

Produces separate statistics for each BY group.

See: "BY" on page 36

"Comparison of the BY and CLASS Statements" on page 783

Example: "Example 3: Using the BY Statement with Class Variables" on page 806

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 omit the NOTSORTED option in the BY statement, then the observations in the data set either must be sorted by all the variables that you specify or 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 sorted 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. 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.

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 21. Also see "Suppressing the Default BY Line" on page 21.

CLASS Statement

Specifies the variables whose values define the subgroup combinations for the analysis.

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 26.

Examples:

"Example 2: Computing Descriptive Statistics with Class Variables" on page 803
"Example 3: Using the BY Statement with Class Variables" on page 806
"Example 4: Using a CLASSDATA= Data Set with Class Variables" on page 808
"Example 5: Using Multilabel Value Formats with Class Variables" on page 811
"Example 6: Using Preloaded Formats with Class Variables" on page 816
"Example 7: Computing a Confidence Limit for the Mean" on page 820
"Example 8: Computing Output Statistics" on page 822
"Example 9: Computing Different Output Statistics for Several Variables" on page 824
"Example 10: Computing Output Statistics with Missing Class Variable Values" on page 826
"Example 11: Identifying an Extreme Value with the Output Statistics" on page 828
"Example 12: Identifying the Top Three Extreme Values with the Output Statistics" on page 831

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 764

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 826

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 816

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 784

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 Language Reference: Concepts for a discussion of missing values with special meanings.

"Example 10: Computing Output Statistics with Missing Class Variable Values" on page 826

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 used 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 641.

"Example 5: Using Multilabel Value Formats with Class Variables" on page 811

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 826

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.

Alias: FMT | EXTERNAL

See: "Example 5: Using Multilabel Value Formats with Class Variables" on page 811

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 811

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.

Alias: 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 764

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:
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.

See: "Example 6: Using Preloaded Formats with Class Variables" on page 816

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 you previously sort BY variables.

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 806.

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 values that PROC MEANS allows depends on the amount of computer memory that is available. See "Computational Resources" on page 765 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.

FREQ Statement

Specifies a numeric variable that contains the frequency of each observation.

See: "FREQ" on page 40

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 785.

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 1428 for information about how PROC MEANS compares character values to determine the maximum value.

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 822

"Example 9: Computing Different Output Statistics for Several Variables" on page 824

"Example 10: Computing Output Statistics with Missing Class Variable Values" on page 826

"Example 11: Identifying an Extreme Value with the Output Statistics" on page 828

"Example 12: Identifying the Top Three Extreme Values with the Output Statistics" on page 831

Syntax

OUTPUT <OUT=*SAS-data-set*> <*output-statistic-specification(s)*> <*id-group-specification(s)*> <*maximum-id-specification(s)*> <*minimum-id-specification(s)*> </ *option(s)*>;

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.

output-statistic-specification(s)
specifies the statistics to store in the OUT= data set and names one or more variables that contain the statistics. The form of the *output-statistic-specification* is

statistic-keyword<(*variable-list*)>=<*name(s)*>
where

statistic-keyword
specifies which statistic to store in the output data set. The available statistic keywords are

Descriptive statistics keyword	
CSS	RANGE
CV	SKEWNESS | SKEW
KURTOSIS | KURT	STDDEV | STD
LCLM	STDERR
MAX	SUM
MEAN	SUMWGT
MIN	UCLM
MODE	USS
N	VAR
NMISS	
Quantile statistics keyword	
MEDIAN | P50	Q3 | P75
P1	P90

P5	P95
P10	P99
P20	P30
P40	P60
P70	P80
Q1 | P25	QRANGE
Hypothesis testing keyword	
PROBT | PRT	T

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 will not inherit the analysis variable's format because this format might be invalid for these statistics (for example, dollar or datetime formats).

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:

"Example 8: Computing Output Statistics" on page 822

"Example 9: Computing Different Output Statistics for Several Variables" on page 824

"Example 11: Identifying an Extreme Value with the Output Statistics" on page 828

"Example 12: Identifying the Top Three Extreme Values with the Output Statistics" on page 831

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

name(s)

specifies one or more names for the variables in output data set that will 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 822

id-group-specification

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. The form of the *id-group-specification* is

IDGROUP (<MIN|MAX (*variable-list-1*) <…MIN|MAX (*variable-list-n*)>> <<MISSING> <OBS><LAST>> OUT <[*n*]>
(*id-variable-list*)=<*name(s)*>)

MIN|MAX(*variable-list*)

specifies the selection criteria to determine the extreme values of one or more input data set variables specified in *variable-list*. Use MIN to determine the minimum extreme value and MAX to determine the maximum extreme value.

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.

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.

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 output 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 output 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 output 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 822

"Example 12: Identifying the Top Three Extreme Values with the Output Statistics" on page 831

name(s)
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.

maximum-id-specification(s)

specifies that one or more identification variables be associated with the maximum values of the analysis variables. The form of the *maximum-id-specification* is

MAXID <(*variable-1*<(*id-variable-list-1*)><…*variable-n*<(*id-variable-list-n*)>>)> = *name(s)*

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

name(s)

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 828

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.

minimum-id-specification

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. The form of the *minid-specification* is

MINID<(*variable-1*<(*id-variable-list-1*)><…*variable-n*<(*id-variable-list-n*)>>)> = *name(s)*

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 will correspond to the observation with the missing value not the observation with the value for the MIN statistic.

option

can be one of the following items:

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 831

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* 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 will 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.

See: "Example 12: Identifying the Top Three Extreme Values with the Output Statistics" on page 831

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 will match 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 799
"Example 8: Computing Output Statistics" on page 822

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 which 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 799
"WAYS Statement" on page 794
"Example 8: Computing Output Statistics" on page 822

TYPES Statement

Identifies which of the possible combinations of class variables to generate.

Requirement: CLASS statement

See: "Output Data Set" on page 799

Examples: "Example 2: Computing Descriptive Statistics with Class Variables" on page 803
"Example 5: Using Multilabel Value Formats with Class Variables" on page 811
"Example 12: Identifying the Top Three Extreme Values with the Output Statistics" on page 831

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 799.

VAR Statement

Identifies the analysis variables and their order in the output.

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 52, “SUMMARY Procedure,” on page 1475

Example: “Example 1: Computing Specific Descriptive Statistics” on page 801

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: Before 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 816

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 observations in the statistical calculations.

See: For information about how to calculate weighted statistics and for an example that uses the WEIGHT statement, see "WEIGHT" on page 43.

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 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.

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.

Restrictions:

To compute weighted quantiles, use QMETHOD=OS in the PROC statement.

Skewness and kurtosis are not available with the WEIGHT statement.

PROC MEANS will 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.

Note: Before Version 7 of SAS, the procedure did not exclude the observations with missing weights from the count of observations.

Tip: When you use the WEIGHT statement, consider which value of the VARDEF= option is appropriate. See the discussion of VARDEF= and the calculation of weighted statistics in "Keywords and Formulas" on page 1666 for more information.

Statistical Computations: 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 1666 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 $\sqrt{\frac{1}{n-1}\Sigma\ (x_i - \bar{x})^2}$ 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

$$\bar{x} + t_{(1-\alpha;n-1)}\frac{s}{\sqrt{n}} \quad \text{(upper)}$$

$$\bar{x} - t_{(1-\alpha;n-1)}\frac{s}{\sqrt{n}} \quad \text{(lower)}$$

A two-sided $100(1 - \alpha)$ % confidence interval for the standard deviation has lower and upper limits

$$s\sqrt{\frac{n-1}{\chi^2_{(1-\alpha/2;n-1)}}},\ s\sqrt{\frac{n-1}{\chi^2_{(\alpha/2;n-1)}}}$$

where $\chi^2_{(1-\alpha/2;n-1)}$ and $\chi^2_{(\alpha/2;n-1)}$ are the $(1 - \alpha/2)$ and $\alpha/2$ critical values of the chi-square statistic with $n - 1$ degrees of freedom. A one-sided $100(1 - \alpha)$ % confidence interval is computed by replacing $\alpha/2$ with α.

A $100(1 - \alpha)$ % confidence interval for the variance has upper and lower limits that are equal to the squares of the corresponding upper and lower limits for the standard deviation.

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 ith 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

$$t_w = \frac{\bar{y}_w - \mu_0}{s_w \sqrt{\sum_{i=1}^{n} w_i}}$$

where $\bar{y}_w$ is the weighted mean, s_w is the weighted standard deviation, and w_i is the weight for *ith* 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 1671. 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 will take 10 times as much memory as it does for data set A. If QMETHOD=P2, then both data sets A and B will require 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 you request has 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

$$m \cdot \left(2^{k} - 1\right) \cdot n + 1$$

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 30.1 *The Effect of Class Variables on the OUTPUT Data Set*

C	B	A	_WAY_	_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	Total	1	
0	0	1	1	1	A	a	1+a
0	1	0	1	2	B	b	
0	1	1	2	3	A*B	a*b	1+a+b+a*b
1	0	0	1	4	C	c	
1	0	1	2	5	A*C	a*c	
1	1	0	2	6	B*C	b*c	1+a+b+a*b+c
1	1	1	3	7	A*B*C	a*b*c	+a*c+b*c+a*b*c
Character binary equivalent of _TYPE_ (CHARTYPE option)					A ,B ,C=CLASS variables	a, b, c,=number of levels of A, B, C, respectively	

Examples: MEANS Procedure

Example 1: Computing Specific Descriptive Statistics

Features: PROC MEANS statement options
statistic keywords
FW=
VAR statement

Data set: CAKE

Details

This example does the following:

- 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 analyses and the analysis 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 30.4 *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

Other features: 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 30.5 *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 30.6 *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

Other features: 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 30.7 *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

Other features: 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 30.8 *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  2007 Allison 31.65 19
Monroe  2007 Barry   23.76 16
Monroe  2007 Candace 21.11  5


     . . . more data lines . . .

Kennedy 2009 Sid     27.45 25
Kennedy 2009 Will    28.88 21
Kennedy 2009 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 '2007-09';
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  2007 Allison 31.65 19
Monroe  2007 Barry   23.76 16
Monroe  2007 Candace 21.11  5


     . . . more data lines . . .

Kennedy 2009 Sid     27.45 25
Kennedy 2009 Will    28.88 21
Kennedy 2009 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 '2007-09';
run;
```

Output

PROC MEANS displays the lower and upper confidence limits for both variables for each year.

Output 30.9 *Confidence Limits*

Confidence Limits for Fund Raising Statistics 2007-09

The MEANS Procedure

Year	N Obs	Variable	Lower 90% CL for Mean	Upper 90% CL for Mean	Mean	Std Dev
2007	34	MoneyRaised	25.69	32.29	28.99	11.37
		HoursVolunteered	17.68	22.73	20.21	8.71
2008	29	MoneyRaised	24.61	31.61	28.11	11.09
		HoursVolunteered	15.67	20.19	17.93	7.17
2009	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

Other features: 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 are 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 30.10 *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

Other features: 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 30.11 *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

Other features: 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 outputs 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 30.12 *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

Other features: 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 30.13 *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 30.14 *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

Other features:
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 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;
```

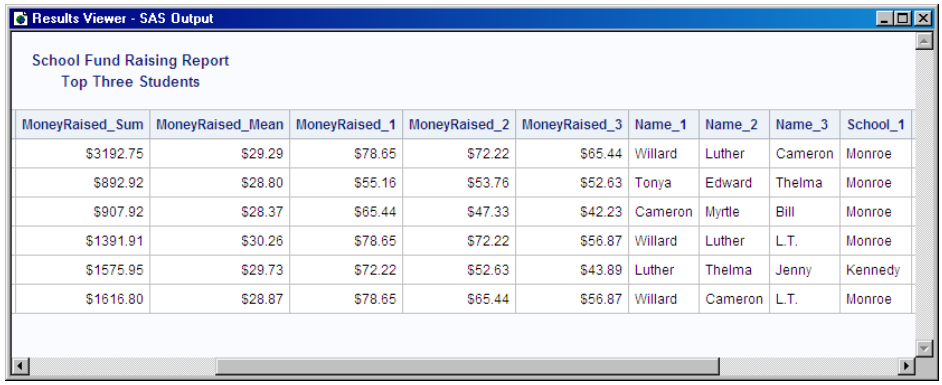
Display information about the TOP3LIST data set. PROC DATASETS displays the contents of the TOP3LIST data set. NOLIST suppresses the directory listing for the WORK data library.

```
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 30.15 *School Fund Raising*


Results Viewer - SAS Output

School Fund Raising Report
Top Three Students

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 definition for this output data set.

Output 30.16 *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	$SCHFMT.	
13	School_1	Char	7	$SCHFMT.	
14	School_2	Char	7	$SCHFMT.	
15	School_3	Char	7	$SCHFMT.	
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 CONTENT

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 30.17 *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_H
1	F	9	Weight	9	90.111111111	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 STACKODS;
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 30.18 *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 Chlamtac I. “The P² Algorithm for Dynamic Calculation of Quantiles and Histograms Without Sorting Observations.” 1985. *Communications of the Association of Computing Machinery* 28 (10): 1076–0185.

Chapter 31

MIGRATE Procedure

Overview: MIGRATE Procedure

What Does the MIGRATE Procedure Do?

The MIGRATE procedure migrates members in a SAS library to the current SAS version.

The procedure migrates a library from most SAS 6, SAS 7, SAS 8, and SAS®9 operating environments to the 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.

The procedure migrates the following library members:

- data sets with alternate collating sequence, audit trails, compression, created and modified datetimes, deleted observations, encryption, generations, indexes, integrity constraints, and passwords
- in many cases, views, catalogs, item stores, and MDDBs (See "Considerations for Each Member Type" on page 842.)

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 SPD engine (Scalable Performance Data engine) data sets. If you are staying on the same operating environment, the SPD engine data sets are compatible. If you are changing to a different operating environment where the data sets are not compatible, use PROC CPORT and PROC CIMPORT to convert them. See *SAS Scalable Performance Data Engine: Reference*.

Best Practices

First use the Compatibility Calculator on the SAS Web site at **`http://support.sas.com/rnd/migration/planning/files/calculator/`** to determine whether you must migrate your libraries. Then use the PROC MIGRATE Calculator on the SAS Web site at **`http://support.sas.com/rnd/migration/planning/files/migratecalc/`**, which provides the PROC MIGRATE syntax for your migration.

To document and validate the migration of your libraries, use the MIGRATE procedure's validation tools. The validation tools are located on the SAS Web site at **`http://support.sas.com/rnd/migration/procmigrate/validtools.html`**. For more information, see "Migrating a Library with Validation Tools " on page 851.

Considerations for Each Member Type

Here are the considerations for each member type. Remember that a SAS data set can be a data file or a data view. See also several important restrictions at the beginning of the syntax section.

Data Files

PROC MIGRATE retains compression, created and modified datetimes, deleted observations, encryption, indexes, integrity constraints, and passwords. The audit trail and generations are also migrated. Indexes and integrity constraints are rebuilt on the member in the target library. Migrated data files take on the data representation and encoding attributes of the target library. See "Migrating a Data File with Audit Trails, Generations, Indexes, or Integrity Constraints" on page 848.

Views

As with data files, migrated data views take on the data representation and encoding attributes of the target library. When you migrate a library that contains DATA step views to a different operating environment, and the views were created before SAS 9.2, setting the proper encoding might be necessary. In releases before SAS 9.2, DATA step views did not save encoding information. Therefore, if the view has a different encoding than the target session, you must specify the INENCODING= option for the source library's LIBNAME statement. Here is an example:

```
libname srclib 'c:\mysource' inencoding="OPEN_ED-1047";
libname lib1 'c:\mytarget';
proc migrate in=srclib out=lib1;
run;
```

A note is printed to the log that recommends using the validation tools to recompile the DATA step view and determine whether the migration was successful. See "Migrating a Library with Validation Tools " on page 851.

In addition, 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 file 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 file LIB1.MYDATA, not LIB2.MYDATA. The following example fails with an error message that LIB1 cannot be found:

```
proc print data=LIB2.MYVIEW;
run;
```

PROC MIGRATE supports three types of views: DATA step views, SQL views, and SAS/ACCESS views:

DATA Step Views
: Beginning with SAS 8, 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 before 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
PROC MIGRATE supports PROC SQL views with no known issues.

SAS/ACCESS Views
PROC MIGRATE supports SAS/ACCESS views that were written with the Oracle, Sybase, or DB2 engine. PROC MIGRATE automatically uses the CV2VIEW procedure, which converts SAS/ACCESS views into SQL views. Migrating SAS/ACCESS views to a different operating environment is not supported. For more information about the conversion, see the overview of the CV2VIEW procedure in *SAS/ACCESS for Relational Databases: Reference*.

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. For example, catalogs in sequential libraries are not migrated.

Restriction	PROC MIGRATE is not supported for SAS 6 AIX catalogs. Use PROC CPORT and PROC CIMPORT instead. See “Additional Steps for Unsupported Catalogs.”
Requirement	If cross-environment data access (CEDA) processing is invoked, and if the IN= source library contains catalogs, then you must specify a SAS/CONNECT or SAS/SHARE libref in the IN= option or in the SLIBREF= option. See “Using the SLIBREF= Option” to determine whether to specify the libref in the IN= or SLIBREF= argument. If the catalogs were created in SAS 6 or SAS 8, SLIBREF= must be assigned through a SAS 8 server.

MDDBs

PROC MIGRATE supports MDDBs with no known issues.

Program Files

PROC MIGRATE does not support program files. If a program file exists in the library, a message is written to the SAS log.

Items Stores

PROC MIGRATE supports item stores unless you migrate from a 32-bit to a 64-bit environment. Migrations from 32-bit to 64-bit environments use Remote Library Services (RLS), which does not support item stores. In that case, an error message is not written to the SAS log, but item stores might not work correctly in the target library.

Syntax: MIGRATE Procedure

Restrictions: SAS data set options are not supported.

The source and target libraries must be in different physical locations.

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 associated with the user ID of the person who ran PROC MIGRATE.

SAS 8.2 source libraries from the following operating environments are not supported: CMS, OS/2, OpenVMS VAX, or 64-bit AIX. See “Alternatives to PROC MIGRATE.”

Source libraries that were created under a 64-bit Windows source environment are supported only for a 64-bit Windows target environment. To migrate to a different target environment, see “Alternatives to PROC MIGRATE.”

Catalogs that were created under a Tru64 UNIX source environment are not supported for a Linux for x64 or a Solaris for x64 target environment. To migrate catalogs under those conditions, see “Additional Steps for Unsupported Catalogs.”

SAS files that were created before SAS 6.12 (SAS 6.09E for z/OS) must be converted to SAS 6.12 before they can be migrated to the current release of SAS. Some SAS 6 source environments are not supported; see “Migrating a SAS 6 Library.”

Tips: For encoding and transcoding issues, see the topic about national language support within the migration topics at **`http://support.sas.com/rnd/migration/`** or see the *SAS National Language Support (NLS): Reference Guide*.

Assign the OUT= 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. Note that members in a sequential library are an exception, because PROC MIGRATE does not read the entire tape to determine existence.

PROC MIGRATE IN=*libref-1* OUT=*libref-2*
<BUFSIZE=*n*>
<MOVE>
<SLIBREF=*libref*>
<KEEPNODUPKEY>;

Statement	Task	Example
“PROC MIGRATE Statement”	Migrate members in a SAS library to the current SAS version	Ex. 1, Ex. 2, Ex. 3, Ex. 4, Ex. 5

PROC MIGRATE Statement

Migrates a SAS library forward to the current release of SAS.

Syntax

PROC MIGRATE IN=*libref-1* OUT=*libref-2*
<BUFSIZE=*n*>
<MOVE>
<SLIBREF=*libref*>
<KEEPNODUPKEY>;

Required Arguments

IN=*libref-1*
names the source SAS library from which to migrate members.

Requirements:
If you use a server, such as SAS/CONNECT or SAS/SHARE, the server must be SAS 9.1.3 or later.

If cross-environment data access (CEDA) processing is invoked, and if the IN= source library contains catalogs, then you must specify a SAS/CONNECT or SAS/SHARE libref in the IN= option or in the SLIBREF= option. See "Using the SLIBREF= Option" to determine whether to specify the libref in the IN= or SLIBREF= argument.

If CATALOG is the only member type in the library and you are using the SLIBREF= option, then omit the IN= argument.

OUT=*libref-2*
names the target SAS library to contain the migrated members.

Restriction: Do not assign the OUT= target library to a server, such as SAS/CONNECT or SAS/SHARE. 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 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. Note that members in a sequential library are an exception, because PROC MIGRATE does not read the entire tape to determine existence.

Optional Arguments

BUFSIZE=*n* | *n*K | *n*M | *n*G
specifies the buffer page size of the members that are written to the target library. For example, a value of **8** specifies a page size of 8 bytes, and a value of **4k** specifies a page size of 4096 bytes.

n
specifies the number of bytes.

*n*K
specifies the number of kilobytes.

*n*M
specifies the number of megabytes.

*n*G
specifies the number of gigabytes.

Default: the original buffer page size that was used to create the source library member

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 SAS log. Specifying MOVE reduces the scope of the validation tools. The validation tools are located on the SAS Web site at **`http://support.sas.com/rnd/migration/procmigrate/validtools.html`**. For more information, see "Migrating a Library with Validation Tools " on page 851.

Restriction: The engine that is associated with the IN= source library must support the deletion of tables. Sequential engines do not support the deletion of tables.

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 or SAS/SHARE server. If cross-environment data access (CEDA) processing is invoked, and if the IN= source library contains catalogs, then you must specify a SAS/CONNECT or SAS/SHARE libref in the IN= option or in the SLIBREF= option. See "Using the SLIBREF= Option" on page 852 to determine whether to specify the libref in the IN= or SLIBREF= argument.

Requirements:
If the catalogs were created in SAS 6 or SAS 8, SLIBREF= must be assigned through a SAS 8 server.

The SLIBREF= server must be running on the same type of operating environment as the source library. For example, if the source session is running under UNIX, then the server must be running under UNIX.

Interactions:
If CATALOG is the only member type in the library and you are using the SLIBREF= option, then omit the IN= argument.

If you are migrating from a SAS®9 release or later (for example, migrating from SAS 9.1.3 to SAS 9.2), then SLIBREF= is not required. If you have incompatible catalogs, specify the SAS/CONNECT or SAS/SHARE server libref in the IN= argument and omit the SLIBREF= argument. For the IN= argument, the server must be SAS 9.1.3 or later.

KEEPNODUPKEY
specifies to retain the NODUPKEY sort order. See "Migrating a SAS Data Set with NODUPKEY Sort Indicator" on page 848.

Migrating a Data File with Audit Trails, Generations, Indexes, or Integrity Constraints

In all cases, the data file 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 file, the data file 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 recreate 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 recreating the index. If the data set is incompatible with the session encoding or data representation, recreating the index produces an error. To resolve the error, recreate 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 file, or while an audit trail or generations are migrated, the data file 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 file from the source library.
- For a data file 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 KEEPNODUPKEY option.

Under the default behavior (without the KEEPNODUPKEY 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 KEEPNODUPKEY 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.

Migrating a SAS 6 Library

For SAS 6 source libraries, the following operating environments are supported. Find your source operating environment in the first column and read across the row for information about the supported target operating environments.

If your libraries are not supported, see "Example 7: Alternatives to PROC MIGRATE" on page 861. If only your catalogs are not supported, see "Example 6: Additional Steps for Unsupported Catalogs" on page 860. For important information about the SLIBREF= option, see "Using the SLIBREF= Option" on page 852.

Table 31.1 *Supported SAS 6 Libraries*

Source Library	Target Library	Instructions for Libraries with Catalogs
SAS 6.12 AIX, HP-UX, or Solaris	AIX, HP-UX, or Solaris	PROC MIGRATE does not support catalogs from SAS 6 AIX. For HP-UX or Solaris libraries that contain catalogs, specify the SLIBREF= option.
SAS 6.12 Windows	32-bit Windows	Catalogs are supported. Do not use the SLIBREF= option.
SAS 6.12 Windows	64-bit Windows	For libraries that contain catalogs, specify the SLIBREF= option.
SAS 6.09E z/OS	z/OS	Catalogs are supported. Do not use the SLIBREF= option.

SAS files that were created before SAS 6.12 (SAS 6.09E for z/OS) must be converted to SAS 6.12 before they can be migrated to the current release of SAS. See also "Example 7: Alternatives to PROC MIGRATE" on page 861.

Migrating a Data Set that Contains Non-English Characters

For SAS data sets that use the ASCII-OEM character set, PROC MIGRATE does not translate non-English characters. To migrate a SAS data set with ASCII-OEM characters to the current release of SAS, use the CPORT and CIMPORT procedures with the TRANTAB option. Specify the appropriate TRANTAB values for the source and target data sets. (See Chapter 15, "CPORT Procedure," on page 349 and Chapter 11, "CIMPORT Procedure," on page 251.)

Migrating Files with Short Extensions on PC Operating Environments

Overview

In SAS 7 and 8, the SHORTFILEEXT option creates a file with a shortened, three-character extension on PC operating environments only. This feature is necessary for operating systems that use a FAT (file allocation table) file system. The FAT file system is also referred to as 8.3 because a filename can include up to eight characters and a file extension can include up to three characters. These files are created on PC environments. They are not usable by SAS on other environments.

Note: SAS 6 files all have three-character extensions but are not affected by this issue. You can distinguish SAS 6 files because their extensions do not contain the number 7.

Below is a table of the short and standard extensions for SAS 7 and 8 files. To determine whether a library contains files with short extensions, look at the filenames in the SAS Explorer or use the file management tools of your operating environment.

Table 31.2 *Short and Standard File Extensions for SAS 7 and 8 Files*

Memtype	Short Extension	Standard Extension
ACCESS	sa7	sas7bacs
AUDIT	st7	sas7baud
CATALOG	sc7	sas7bcat
DATA	sd7	sas7bdat
DMDB	s7m	sas7bdmd
FDB	sf7	sas7bfdb
INDEX	si7	sas7bndx
ITEMSTOR	sr7	sas7bitm
MDDB	sm7	sas7bmdb
PROGRAM	ss7	sas7bpgm
PUTILITY	sp7	sas7bput
UTILITY	su7	sas7butl
VIEW	sv7	sas7bvew

SAS®9 Compatibility with Short-Extension Files

SAS 9.0
: supports the SHORTFILEEXT option, although it is not documented. If you use this option, you have Read and Write access to your existing short-extension files.

SAS 9.1 and 9.1.2
: do not support any access to short-extension files. You can use PROC COPY in SAS 8 to copy the short file extensions to standard file extensions before migrating your files to SAS®9.

  As an alternative to PROC COPY, if you are comfortable using operating system commands or file management tools, you can manually change the file's extension to the standard extension. The content of a short-extension file is identical to the content of a standard-extension file. The use of operating system commands is not supported by SAS.

SAS 9.1.3 and later
: support Read-Only access to existing short-extension files. You can migrate your library to the current release of SAS by using PROC MIGRATE. You must specify the SHORTFILEEXT option in the LIBNAME statement for the source library. The files are written to the target library with standard extensions; the files support full access.

For example, a library named MYLIB contains two files with short extensions: a SAS data set named **`mydata.sd7`** and a catalog named **`mycat.sc7`**. Use the following code to migrate the library to SAS®9:

```
libname mylib v8 'source-library-pathname' shortfileext;
libname newlib v9 'target-library-pathname';

proc migrate in=mylib out=newlib;
run;
```

After migration, the target library NEWLIB contains two files with standard extensions: a SAS data set named **`mydata.sas7bdat`** and a catalog named **`mycat.sas7bcat`**.

If your library also contains standard-extension files, then perform an additional migration without the SHORTFILEEXT option in the LIBNAME statement to migrate those files. Make sure that no short-extension files have the same name as a standard-extension file. In the target library, all files have a standard extension. If a short-extension file and a standard-extension file have the same name and same member type in the target library, the second one fails to migrate.

Migrating a Library with Validation Tools

When you run PROC MIGRATE with validation tools, you generate Output Delivery System (ODS) reports that validate a successful migration. The validation tools are located on the SAS Web site at **`http://support.sas.com/rnd/migration/procmigrate/validtools.html`**. The validation tools support a basic migration or a migration with the SLIBREF option (which uses RLS).

The validation tools consist of a set of macros and a template of example code. Some of the macros run before the migration to record the expected behavior of PROC MIGRATE. Another group of macros runs after the migration to record the actual behavior. The final group of macros compares expected and actual behavior.

If you use the MOVE option with PROC MIGRATE, the validation tools can produce validation output only for the members that were migrated. The MOVE option deletes the source library after it has been moved to the target library. For these reasons, the MOVE option significantly limits the validation tools. For example code, see the validation tools topics on the SAS Web site (the URL above).

Using the SLIBREF= Option

When to Use the SLIBREF= Option

The SLIBREF= option is required if following two conditions are both met: if the source library contains catalogs that were created prior to SAS 9.1.3 and if the processing invokes CEDA on the target session. (In general, CEDA is invoked when you migrate to an incompatible operating environment or with an incompatible session encoding.)

For SAS 8 files, you can run a test to determine whether CEDA processing will be used by PROC MIGRATE. Starting in SAS®9, by default, SAS writes a message to the log when CEDA is used (prior to SAS®9, set MSGLEVEL=I). In the target session, try processing a data set that was created under the source session. Submit something simple like the CONTENTS procedure. Check the SAS log for a message. If CEDA processing was used, then you need the SLIBREF= option.

For SAS 6 files, only use the SLIBREF= option for SAS 6 HP-UX or Solaris libraries that contain catalogs. For more information, see “Migrating a SAS 6 Library” on page 849.

If you are uncertain whether you must specify SLIBREF=, use the PROC MIGRATE Calculator on the SAS Web site at **`http://support.sas.com/rnd/migration/planning/files/migratecalc/`**.

For sample code, see “Example 2: Migrating with Incompatible Catalogs across Computers” on page 855 and “Example 4: Migrating with Incompatible Catalogs on the Same Computer ” on page 857. For more information about CEDA, see the *SAS Language Reference: Concepts*.

When to Not Use the SLIBREF= Option

- If you are migrating from SAS®9 or later (for example, migrating from SAS 9.1.3 to SAS 9.2), then SLIBREF= is not required. If you have incompatible catalogs, specify the SAS/CONNECT or SAS/SHARE server libref in the IN= argument and omit the SLIBREF= argument.
- Do not use the SLIBREF= option if the library contains no catalogs.
- If the library does contain catalogs, do not use the SLIBREF= option if CEDA processing will not be used by PROC MIGRATE.

If you are uncertain whether you must specify SLIBREF=, use the PROC MIGRATE Calculator on the SAS Web site at **`http://support.sas.com/rnd/migration/planning/files/migratecalc/`**.

Requirements for the SAS/CONNECT or SAS/SHARE Server

To use the SLIBREF= option, you must have access to a SAS/CONNECT or SAS/SHARE server that is running on the same type of operating environment as the source library. For example, if the source session is running under UNIX, then the server must be running under UNIX.

If the catalogs were created in SAS 6 or SAS 8, SLIBREF= must be assigned through a SAS 8 server. (Note that this is not the same server that you assign through the IN= argument. If you assign a server through the IN= argument, the IN= server must be SAS 9.1.3 or later.)

If you cannot meet these requirements, use the alternate method described in "Example 6: Additional Steps for Unsupported Catalogs" on page 860.

Restrictions for the SLIBREF= Option

If CATALOG is the only member type in the library and you are using the SLIBREF= option, then omit the IN= argument.

When you use the SLIBREF= option for a SAS 8.2 library, multilabel formats are not supported. If a catalog contains a multilabel format, the format is not created on the target and an error is printed to the log. See SAS Note 20052, which is available from SAS customer support at **`http://support.sas.com`**.

Examples: MIGRATE Procedure

Example 1: Migrating across Computers

Features: IN=

OUT=

Notes: You are encouraged to run PROC MIGRATE with validation tools. See "Migrating a Library with Validation Tools " on page 851.

To learn whether this is the correct PROC MIGRATE example for your migration, use the PROC MIGRATE Calculator on the SAS Web site at **`http://support.sas.com/rnd/migration/planning/files/migratecalc/`**

Details

In this example, the following is demonstrated:

- The source and target libraries are on different computers.
- The SLIBREF= argument is not used. (See "Using the SLIBREF= Option" on page 852 to learn whether SLIBREF= is required.)
- The IN= argument accesses all of the supported file types in the source library. You can assign the source library to the IN= argument in one of the following two ways:
  - directly via a Network File System (NFS)

- via a SAS 9 SAS/CONNECT or SAS/SHARE server

The direct method is possible only if you can access the library via an NFS, which is a standard protocol of UNIX operating environments. If you want to use that method, see the documentation for NFS and for your operating environment.

This example uses SAS/CONNECT software. The SAS/CONNECT or SAS/SHARE server that you assign to the IN= argument must be SAS 9.1.3 or later.

Program

```
signon serv-ID sascmd='my-sas-invocation-command';

rsubmit;
libname source <engine> 'source-library-pathname';
endrsubmit;

libname source <engine> server=serv-ID;

libname target <engine> 'target-library-pathname';

proc migrate in=source out=target <options>;
run;
```

Program Description

From a session in the current release of SAS, submit the SIGNON command to invoke a SAS/CONNECT server session. Note that because you are working across computers, you might specify a machine name in the server ID.

```
signon serv-ID sascmd='my-sas-invocation-command';
```

Within this remote session, assign a libref to the source library that contains the library members to be migrated. Use the RSUBMIT and ENDRSUBMIT commands for SAS/CONNECT.

```
rsubmit;
libname source <engine> 'source-library-pathname';
endrsubmit;
```

In the local (client) session in the current release, assign the same source libref as in step 2. But do not assign the libref to a physical location. Instead, specify the SERVER= option with the server ID (in this example, serv-ID) that you assigned in the SIGNON command in step 1.

```
libname source <engine> server=serv-ID;
```

Assign the target library.

```
libname target <engine> 'target-library-pathname';
```

Use PROC MIGRATE. If your library contains catalogs, see "Example 2: Migrating with Incompatible Catalogs across Computers" on page 855 instead.

```
proc migrate in=source out=target <options>;
run;
```

Example 2: Migrating with Incompatible Catalogs across Computers

Features: IN=
OUT=
SLIBREF=

Notes: You are encouraged to run PROC MIGRATE with validation tools. See "Migrating a Library with Validation Tools " on page 851.

To learn whether this is the correct PROC MIGRATE example for your migration, use the PROC MIGRATE Calculator on the SAS Web site at **http://support.sas.com/rnd/migration/planning/files/migratecalc/**

Details

In this example, the following is demonstrated:

- The source and target libraries are on different computers.
- The SLIBREF= argument accesses the catalogs in the source library. (See "Using the SLIBREF= Option" on page 852 to learn whether SLIBREF= is required.) The SLIBREF= argument must be assigned to a SAS/CONNECT or SAS/SHARE server running in a session of SAS that can access the catalogs. For example, if the source library contains SAS 8.2 catalogs created by 32-bit Solaris, SLIBREF= must be assigned to a SAS 8.2 32-bit Solaris server. If catalogs were created in SAS 6, SLIBREF= must be assigned through a SAS 8.2 server that is compatible with the data representation of the SAS 6 catalogs.
- The IN= argument accesses the rest of the supported file types in the source library. You can assign the source library to the IN= argument in one of the following two ways:
  - directly via a Network File System (NFS)
  - via a SAS/CONNECT or SAS/SHARE server

  This example uses NFS, which is a standard protocol of UNIX operating environments. See the documentation for NFS and for your operating environment.

Program

```
signon v8srv sascmd='my-v8-sas-invocation-command';

rsubmit;
libname srclib <engine> 'source-library-pathname';
endrsubmit;

libname source <engine> '/nfs/v8machine-name/source-library-pathname';

libname srclib <engine> server=v8srv;

libname target <engine> 'target-library-pathname';

proc migrate in=source out=target slibref=srclib <options>;
run;

proc migrate out=target slibref=srclib <options>;
run;
```

Program Description

From a session in the current release of SAS, submit the SIGNON command to invoke a SAS/CONNECT server session. Note that because you are working across computers, you might specify a machine name in the server ID.

```
signon v8srv sascmd='my-v8-sas-invocation-command';
```

Within this remote SAS 8.2 session, assign a libref to the source library that contains the library members to be migrated. Use the RSUBMIT and ENDRSUBMIT commands for SAS/CONNECT.

```
rsubmit;
libname srclib <engine> 'source-library-pathname';
endrsubmit;
```

In the local (client) session in the current release, assign to the same source library through NFS.

```
libname source <engine> '/nfs/v8machine-name/source-library-pathname';
```

Assign the same libref to the same source libref as in step 2 (in this example, SRCLIB). But do not assign the libref to a physical location. Instead, specify the SERVER= option with the server ID (in this example, V8SRV) that you assigned in the SIGNON command in step 1.

```
libname srclib <engine> server=v8srv;
```

Assign the target library.

```
libname target <engine> 'target-library-pathname';
```

Use PROC MIGRATE with the SLIBREF= option. For the IN= and OUT= options, specify the usual source and target librefs (in this example, SOURCE and TARGET, respectively). Set SLIBREF= to the libref that uses the SERVER= option (in this example, SRCLIB).

```
proc migrate in=source out=target slibref=srclib <options>;
run;
```

Alternatively, if CATALOG is the only member type in the library and you are using the SLIBREF= option, then omit the IN= argument.

```
proc migrate out=target slibref=srclib <options>;
run;
```

Example 3: Migrating on the Same Computer

Features: IN=

OUT=

Notes: You are encouraged to run PROC MIGRATE with validation tools. See “Migrating a Library with Validation Tools ” on page 851.

To learn whether this is the correct PROC MIGRATE example for your migration, use the PROC MIGRATE Calculator on the SAS Web site at **http://support.sas.com/rnd/migration/planning/files/migratecalc/**

Details

In this example, the following is demonstrated:

- The source and target libraries are on one computer.
- The SLIBREF= argument is not used. (See "Using the SLIBREF= Option" on page 852 to learn whether SLIBREF= is required.)
- The IN= argument accesses all of the supported file types in the source library. Because the source and target are on one computer, NFS is not used.

Program

```
libname source <engine> 'source-library-pathname';
libname target base 'target-library-pathname';

proc migrate in=source out=target;
run;
```

Program Description

From a session in the current release of SAS, submit the following.

```
libname source <engine> 'source-library-pathname';
libname target base 'target-library-pathname';

proc migrate in=source out=target;
run;
```

Example 4: Migrating with Incompatible Catalogs on the Same Computer

Features: IN=
OUT=
SLIBREF=

Notes: You are encouraged to run PROC MIGRATE with validation tools. See "Migrating a Library with Validation Tools " on page 851.

To learn whether this is the correct PROC MIGRATE example for your migration, use the PROC MIGRATE Calculator on the SAS Web site at **`http://support.sas.com/rnd/migration/planning/files/migratecalc/`**

Details

In this example, the following is demonstrated:

- The source and target libraries are on one computer.
- The SLIBREF= argument accesses the catalogs in the source library. (See "Using the SLIBREF= Option" on page 852 for important information and to learn whether SLIBREF= is required.) The SLIBREF= argument must be assigned to a SAS/CONNECT or SAS/SHARE server running in a session of SAS that can access the catalogs. For example, if the source library contains SAS 8.2 catalogs created by 32-bit Solaris, SLIBREF= must be assigned to a SAS 8.2 32-bit Solaris server. If catalogs were created in SAS 6, SLIBREF= must be assigned through a SAS 8.2 server that is compatible with the data representation of the SAS 6 catalogs.

- The IN= argument accesses the rest of the supported file types in the source library. Because the source and target libraries are on one computer, NFS is not used.

Program

```
signon v8srv sascmd='my-v8-sas-invocation-command';

rsubmit;
libname srclib <engine> 'source-library-pathname';
endrsubmit;

libname srclib <engine> server=v8srv;

libname source <engine> 'source-library-pathname';
libname target <engine> 'target-library-pathname';

proc migrate in=source out=target slibref=srclib <options>;
run;

proc migrate out=target slibref=srclib <options>;
run;
```

Program Description

From a session in the current release of SAS, submit the SIGNON command to invoke a SAS/CONNECT server session.

```
signon v8srv sascmd='my-v8-sas-invocation-command';
```

Within this remote SAS 8.2 session, assign a libref to the source library that contains the library members to be migrated. Use the RSUBMIT and ENDRSUBMIT commands for SAS/CONNECT.

```
rsubmit;
libname srclib <engine> 'source-library-pathname';
endrsubmit;
```

In the local (client) session in the current release, assign to the same source library as in step 2. But do not assign the libref to a physical location. Instead, specify the SERVER = option with the server ID (in this example, V8SRV) that you assigned in the SIGNON command in step 1.

```
libname srclib <engine> server=v8srv;
```

Assign two librefs: one to the source library and another to the target library. The source library is the same one that is assigned in step 2, but you must use a different libref.

```
libname source <engine> 'source-library-pathname';
libname target <engine> 'target-library-pathname';
```

Use PROC MIGRATE with the SLIBREF= option. For the IN= and OUT= options, specify the librefs that you assigned in step 4 (in this example, SOURCE and TARGET, respectively). Set SLIBREF= to the libref that uses the SERVER= option (in this example, SRCLIB).

```
proc migrate in=source out=target slibref=srclib <options>;
run;
```

Alternatively, if CATALOG is the only member type in the library and you are using the SLIBREF= option, then omit the IN= argument.

```
proc migrate out=target slibref=srclib <options>;
run;
```

Example 5: Migrating from a SAS®9 Release with Incompatible Catalogs

Features: IN=

OUT=

Notes: You are encouraged to run PROC MIGRATE with validation tools. See "Migrating a Library with Validation Tools " on page 851 .

To learn whether this is the correct PROC MIGRATE example for your migration, use the PROC MIGRATE Calculator on the SAS Web site at **http://support.sas.com/rnd/migration/planning/files/migratecalc/**

Details

If you are migrating from SAS®9 or later (for example, migrating from SAS 9.1.3 to SAS 9.2), then SLIBREF= is not required. If you have incompatible catalogs, specify the SAS/CONNECT or SAS/SHARE server libref in the IN= argument and omit the SLIBREF= argument.

In this example, the following is demonstrated:

- The source and target libraries can be on the same computer or on different computers.
- The SLIBREF= argument is not used. (See "Using the SLIBREF= Option" on page 852 for important information and to learn whether SLIBREF= is required.)
- The IN= argument accesses all of the supported file types in the source library, including incompatible catalogs, by using SAS/CONNECT or SAS/SHARE software. This example uses SAS/CONNECT software. The SAS/CONNECT or SAS/SHARE server that you assign to the IN= argument must be SAS 9.1.3 or later.

Program

```
signon serv-ID sascmd='my-sas-invocation-command';

rsubmit;
libname source <engine> 'source-library-pathname';
endrsubmit;

libname source <engine> server=serv-ID;

libname target <engine> 'target-library-pathname';

proc migrate in=source out=target <options>;
run;
```

Program Description

From a session in the current release of SAS, submit the SIGNON command to invoke a SAS/CONNECT server session. Note that if you are working across computers, you might specify a machine name in the server ID.

```
signon serv-ID sascmd='my-sas-invocation-command';
```

Within this remote session, assign a libref to the source library that contains the library members to be migrated. Use the RSUBMIT and ENDRSUBMIT commands for SAS/CONNECT.

```
rsubmit;
libname source <engine> 'source-library-pathname';
endrsubmit;
```

In the local (client) session in the current release, assign the same source libref as in step 2. But do not assign the libref to a physical location. Instead, specify the SERVER= option with the server ID (in this example, serv-ID) that you assigned in the SIGNON command in step 1.

```
libname source <engine> server=serv-ID;
```

Assign the target library.

```
libname target <engine> 'target-library-pathname';
```

Use PROC MIGRATE.

```
proc migrate in=source out=target <options>;
run;
```

Example 6: Additional Steps for Unsupported Catalogs

Details

PROC MIGRATE is not supported for migrating catalogs under the following circumstances:

- SAS 6 AIX catalogs to any target library.
- Tru64 UNIX catalogs to either Linux for x64 or Solaris for x64 target library.
- any catalogs if you must use SLIBREF but you do not have access to either SAS/CONNECT or SAS/SHARE software. To learn whether SLIBREF= is required, see “Using the SLIBREF= Option” on page 852.

This example is the best practice to convert catalogs so that they are native with the target environment and with the current release of SAS.

You can use this process for just the catalogs or for all members of a library. You might want to limit this method to catalogs for the following reasons:

- FTP can be time-consuming for large libraries.
- By using PROC MIGRATE for other members of the library, you can retain those members' attributes and use validation tools.

For more information, see *Moving and Accessing SAS Files*.

Program

1. In the source session, create a transport file with PROC CPORT. (See Chapter 15, "CPORT Procedure," on page 349.)
2. Move the transport file to the target environment. Do not use RLS (a feature of SAS/CONNECT and SAS/SHARE software) to move catalogs, or you will encounter errors. You must use binary FTP, the DOWNLOAD procedure, NFS (Network File System), or another method of directly accessing files.
3. In the target session, use CIMPORT to import the transport file. (See Chapter 11, "CIMPORT Procedure," on page 251.)

Example 7: Alternatives to PROC MIGRATE

Details

PROC MIGRATE is not supported for libraries that were created under the following source environments:

- SAS 8.2 libraries from CMS, OS/2, OpenVMS VAX, or 64-bit AIX.
- any unsupported SAS 6 operating environment. For a list of supported SAS 6 operating environments, see "Migrating a SAS 6 Library" on page 849.

Program

To migrate across operating environments that cannot use PROC MIGRATE, use the following process:

1. Under the source installation of SAS, use one of the conversion procedures, PROC COPY, PROC CPORT, or PROC UPLOAD, to create a transport file containing your source library. See *Moving and Accessing SAS Files* .
2. Under the target installation of SAS, use PROC COPY, PROC CIMPORT, or PROC DOWNLOAD to convert the transport file to a SAS®9 library.

Chapter 32

OPTIONS Procedure

Overview: OPTIONS Procedure

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

For additional information about SAS system options, see *SAS System Options: Reference*.

Syntax: OPTIONS Procedure

See: "OPTIONS Procedure: UNIX" in *SAS Companion for UNIX Environments*
"OPTIONS Procedure: Windows" in *SAS Companion for Windows*
"OPTIONS Procedure: z/OS" in *SAS Companion for z/OS*

PROC OPTIONS <*option(s)*>;

Statement	Task	Example
"PROC OPTIONS Statement"	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 878
"Example 2: Displaying the Setting of a Single Option" on page 879
"Example 3: Displaying Expanded Path Environment Variables" on page 880
"Example 4: List the Options That Can Be Specified by the INSERT and APPEND Options" on page 881

Syntax

PROC OPTIONS <*option(s)*>;

Summary of Optional Arguments

LISTGROUPS
displays system option groups as well as a description of each group.

Choose the format of the listing

DEFINE
displays the short description of the option, the option group, and the option type.

EXPAND
when displaying a character option, 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.

HEXVALUE
displays system option character values as hexadecimal values.

LOGNUMBERFORMAT
displays numeric system option values using locale-specific punctuation.

LONG
lists each system option on a separate line with a description.

NOEXPAND
when displaying a path, displays the path using environment variable(s) and not the value of the environment variable(s). This is the default.

NOLOGNUMBERFORMAT
displays numeric system option values without using punctuation, such as a comma or a period. This is the default.

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.

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 879

EXPAND
when displaying a character option, 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.

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 867 option to view paths that display the environment variable

Example: "Example 3: Displaying Expanded Path Environment Variables" on page 880

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: "Displaying Information about System Option Groups" on page 872

HEXVALUE
displays system option character values as hexadecimal values.

HOST
displays only host options.

See: "NOHOST" on page 867 option to display only portable options.

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* and
"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 881

LISTGROUPS
displays system option groups as well as a description of each group.

See: "Displaying Information about System Option Groups" on page 872

LISTRESTRICT
lists the system options that can be restricted by your site administrator.

See: "RESTRICT" on page 867 option to list options that have been restricted by the site administrator

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 867 option to produce a compressed listing without descriptions

Example: "Example 1: Producing the Short Form of the Options Listing" on page 878

LOGNUMBERFORMAT
displays numeric system option values using locale-specific punctuation.

See: "NOLOGNUMBERFORMAT" on page 867 option to display numeric option values without using commas

Example: "Example 2: Displaying the Setting of a Single Option" on page 879

NOEXPAND
when displaying a path, displays the path using environment variable(s) and not the value of the environment variable(s). This is the default.

See: "EXPAND" on page 865 option to display a path by expanding the value of environment variables

NOHOST
displays only portable options.

Alias: PORTABLE or PORT

See: "HOST" on page 866 option to display only host options

NOLOGNUMBERFORMAT
displays numeric system option values without using punctuation, such as a comma or a period. This is the default.

See: "LOGNUMBERFORMAT" on page 867 option to display numeric system options using commas

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 879

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 866 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 866 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.

Interaction: This option has no effect when SHORT is specified.

Note: SAS options that are passwords, such as EMAILPW and METAPASS, return the value xxxxxxxx and not the actual password.

Example: "Example 2: Displaying the Setting of a Single Option" on page 879

Displaying 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;
```

Log 32.1 *The SAS Log Showing a Partial Listing of SAS System Options*

```
Portable Options:

 APPEND=           Append at the end of the option value
 APPLETLOC=site-specific-path
                   Location of Java applets
 ARMAGENT=         ARM Agent to use to collect ARM records
 ARMLOC=ARMLOG.LOG Identify location where ARM records are to be written
 ARMSUBSYS=(ARM_NONE)
                   Enable/Disable ARMing of SAS subsystems
 AUTOCORRECT       Perform auto-correction for misspelled procedure names,
keywords or global statement names
 AUTOEXEC=         Identifies AUTOEXEC files used during initialization
 AUTOSAVELOC=      Identifies the location where program editor contents are
auto saved
 NOAUTOSIGNON      SAS/CONNECT remote submit will not automatically attempt to
SIGNON
 BINDING=DEFAULT   Controls the binding edge for duplexed output
 BOMFILE           Add Byte Order Mark when creating Unicode files
 BOTTOMMARGIN=0.000 IN
                   Bottom margin for printed output
 BUFNO=1           Number of buffers for each SAS data set
 BUFSIZE=0         Size of buffer for page of SAS data set
 BYERR             Set the error flag if a null data set is input to the SORT
procedure
 BYLINE            Print the BY line at the beginning of each BY group
 BYSORTED          Require SAS data set observations to be sorted for BY
processing
 NOCAPS            Do not translate source input to uppercase
 NOCARDIMAGE       Do not process SAS source and data lines as 80-byte records
 CATCACHE=0        Number of SAS catalogs to keep in cache memory
 CBUFNO=0          Number of buffers to use for each SAS catalog
 CENTER            Center SAS procedure output
 CGOPTIMIZE=3      Control code generation optimization
 NOCHARCODE        Do not use character combinations as substitute for special
characters not on the keyboard
 CLEANUP           Attempt recovery from out-of-resources condition
 NOCMDMAC          Do not support command-style macros
 CMPLIB=           Identify previously compiled libraries of CMP subroutines to
use when linking
 CMPMODEL=BOTH     Identify CMP model storage type
 CMPOPT=(NOEXTRAMATH NOMISSCHECK NOPRECISE NOGUARDCHECK NOGENSYMNAMES
NOFUNCDIFFERENCING)
                   Enable SAS compiler performance optimizations
```

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;
```

Log 32.2 *The SAS Log Showing a Partial List of Host Options*

```
1    proc options host;
2    run;

    SAS (r) Proprietary Software Release xxx  TS1B0


Host Options:

 ACCESSIBILITY=STANDARD
                   Enable Extended Accessibility
 ALTLOG=           Specifies the destination for a copy of the SAS log
 ALTPRINT=         Specifies the destination for a copy of the SAS procedure
output file
 AUTHPROVIDERDOMAIN=
                   Authentication providers associated with domain suffixes
 AUTHSERVER=       Specify the authentication server or domain.
 AWSCONTROL=(SYSTEMMENU MINMAX TITLE)
                   Used to customize the appearance for the SAS AWS. Valid
parameters are: TITLE/ NOTITLE
                   SYSTEMMENU/NOSYSTEMMENU MINMAX/NOMINMAX
 AWSDEF=(0 0 80 80)
                   Specify the initial size and position of the SAS AWS. This
should be specified as follows: 0 0 100 100
 AWSMENU           Show the main window's (AWS) menu.
```

Displaying 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;
```

Log 32.3 *The Setting of a Single SAS System Option*

```
11    proc options option=errorcheck define;
2    run;

    SAS (r) Proprietary Software Release xxx  TS1B0

 ERRORCHECK=NORMAL
Option Definition Information for SAS Option ERRORCHECK
    Group= ERRORHANDLING
    Group Description: Error messages and error conditions settings
    Description: Level of special error processing to be performed
    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;
```

Log 32.4 *The Settings of Two SAS System Options*

```
7    proc options option=(pdfsecurity pdfpassword) define;
8    run;

    SAS (r) Proprietary Software Release xxx  TS1B0

 PDFSECURITY=NONE
Option Definition Information for SAS Option PDFSECURITY
    Group= PDF
    Group Description: PDF settings
    Group= SECURITY
    Group Description: Security settings
    Description: Identify the level of PDF document encryption
    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=xxxxxxxx
Option Definition Information for SAS Option PDFPASSWORD
    Group= PDF
    Group Description: PDF settings
    Group= SECURITY
    Group Description: Security settings
    Description: PDF open and owner passwords
    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.
          Password Option Value: Can not Print or Display
          Expansion: Environment variables, within the option value, are not
expanded
    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
```

Displaying 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;
```

Log 32.5 *List of SAS System Option Groups*

```
26   proc options listgroups;
27   run;

    SAS (r) Proprietary Software Release xxx  TS1B0


    Option Groups
       GROUP=ADABAS                ADABAS

       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
```

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;
```

Log 32.6 *Sample Output Using the GROUP= Option*

```
5    proc options group=(svg graphics);
6    run;

    SAS (r) Proprietary Software Release xxx  TS1B0


Group=SVG
 NOSVGCONTROLBUTTONS
                  Do not display paging control buttons in multi-page SVG
documents
 SVGHEIGHT=       Identifies the height of the SVG outermost element and height
of the initial viewport
 SVGPRESERVEASPECTRATIO=
                  Identifies the preserveAspectRatio attribute for the
outermost SVG element
 SVGTITLE=        Identifies SVG title element
 SVGVIEWBOX=      Identifies the viewBox attribute for the outermost SVG element
 SVGWIDTH=        Identifies width of the SVG outermost element and width of
the initial viewport
 SVGX=            Identifies the x-axis attribute for the outermost SVG element
 SVGY=            Identifies the y-axis attribute for the outermost SVG element

Group=GRAPHICS
 DEVICE=          Graphics device driver
 GSTYLE           Use ODS styles in the generation of graphs. GSTYLE does not
affect ODS styles for graphs generated
                  with Java, ActiveX drivers or Statistical Graphics
 GWINDOW          Display SAS/GRAPH output in the GRAPH window of Display
Manager
 MAPS=("!sasroot\your-site-path\en\maps")
                  Location of maps for use with SAS/GRAPH
 FONTALIAS=       Assigns a new host font facename to a portable SAS font
family. The first parameter is the SAS font
                  family and the second is the host-specific font facename.
```

The following table lists the values that are available in all operating environments when you use the GROUP= option with PROC OPTIONS.

Values for Use with GROUP=		
CODEGEN	HELP	META
COMMUNICATIONS	INPUTCONTROL	ODSPRINT
DATAQUALITY	INSTALL	PDF
EMAIL	LANGUAGECONTROL	PERFORMANCE
ENVDISPLAY	LISTCONTROL	SASFILES
ENVFILES	LOG_LISTCONTROL	SECURITY
ERRORHANDLING	LOGCONTROL	SORT
EXECMODES	MACRO	SQL
EXTFILES	MEMORY	SVG
GRAPHICS		

The following table lists operating environment–specific values that might be available when you use the GROUP= option with PROC OPTIONS.

Possible Operating Environment-Specific Values for Use with GROUP=			
ADABAS	DB2	ISPF	SMF
CODEGEN	IDMS	ORACLE	
DATACOM	IMS	REXX	

Operating Environment Information
Refer to the SAS documentation for your operating environment for more information about these host-specific options.

Displaying Restricted Options

Your site administrator can restrict some system options so that your SAS session adheres 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 Table 1.1, “System Options That Cannot Be Restricted,” 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.

Log 32.7 *A List of Options That Have Been Restricted by the Site Administrator*

```
1
   proc options restrict;
2    run;
    SAS (r) Proprietary Software Release xxx  TS1B0

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
```

Log 32.8 *A Partial Log That Lists Options That Can Be Restricted*

```
33   proc options listrestrict;
34   run;

    SAS (r) Proprietary Software Release xxx  TS1B0


    Your Site Administrator can restrict the ability to modify the following
Portable Options:

     APPEND             Append at the end of the option value
     APPLETLOC          Location of Java applets
     ARMAGENT           ARM Agent to use to collect ARM records
     ARMLOC             Identify location where ARM records are to be written
     ARMSUBSYS          Enable/Disable ARMing of SAS subsystems
     AUTOCORRECT        Perform auto-correction for misspelled procedure names,
keywords or global statement names
     AUTOSAVELOC        Identifies the location where program editor contents
are auto saved
```

Results: OPTIONS Procedure

SAS writes the options list to the SAS log. SAS system options of the form ***option*** | **NO*option*** are listed as either ***option*** or **NO*option***, depending on the current setting. They are always sorted by the positive form. For example, NOCAPS would be listed under the Cs.

Operating Environment Information
PROC OPTIONS produces additional information that is specific to the environment under which you are running the SAS System. Refer to the SAS documentation for your operating environment for more information about this and for descriptions of host-specific options.

See Also

- *SAS Companion for UNIX Environments*
- *SAS Companion for Windows*
- *SAS Companion for z/OS*

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 "Displaying a List of System Options" on page 868.

Program

```
proc options short;
run;
```

Program Description

List all options and their settings. SHORT lists the SAS system options and their settings without any descriptions.

```
proc options short;
run;
```

Log

Log 32.9 *Partial Listing of the SHORT Option*

```
6    proc options short;
7    run;
    SAS (r) Proprietary Software Release xxx  TS1B0

Portable Options:

 APPEND= APPLETLOC=your-directory 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
CATCACHE=0
CBUFNO=0 CENTER CGOPTIMIZE=3 NOCHARCODE NOCHKPTCLEAN CLEANUP NOCMDMAC CMPLIB=
CMPMODEL=BOTH
CMPOPT=(NOEXTRAMATH NOMISSCHECK NOPRECISE NOGUARDCHECK NOGENSYMNAMES
NOFUNCDIFFERENCING) NOCOLLATE
COLORPRINTING COMAMID=TCP COMPRESS=NO CONNECTMETACONNECTION
CONNECTOUTPUT=BUFFERED
CONNECTPERSIST CONNECTREMOTE=CONNECTSTATUS CONNECTWAIT COPIES=1 CPUCOUNT=2 CPUID
DATASTMTCHK=COREKEYWORDS DATE DATESTYLE=MDY NODBFMTIGNORE NODBIDIRECTEXEC
DBSLICEPARM=
(THREADED_APPS, 2) DBSRVTP=NONE DEFLATION=6 NODETAILS DEVICE= DFLANG=ENGLISH
DKRICOND=ERROR
DKROCOND=WARN NODLCREATEDIR DLDMGACTION=REPAIR NODMR DMS DMSEXP DMSLOGSIZE=99999
DMSOUTSIZE=99999
DMSPGMLINESIZE=136
```

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 MEMBLKSZ. The DEFINE and VALUE options display additional information. The LOGNUMBERFORMAT displays the value using commas.

Program

```
proc options option=memblksz define value lognumberformat;
run;
```

Program Description

Specify the MEMBLKSZ SAS system option. OPTION=MEMBLKSZ displays option value information. DEFINE and VALUE display additional information. LOGNUMBERFORMAT specifies to format the value using commas.

```
proc options option=memblksz define value lognumberformat;
run;
```

Log

Log 32.10 *Log Output from Specifying the MEMBLKSZ Option*

```
13   proc options option=memblksz define value lognumberformat;
14   run;

    SAS (r) Proprietary Software Release xxx

Option Value Information For SAS Option MEMBLKSZ
    Value: 16,777,216
    Scope: Default
    How option value set: Shipped Default


Option Definition Information for SAS Option MEMBLKSZ
    Group= MEMORY
    Group Description: Memory settings
    Description: Size of memory blocks allocated to support MEMLIB and MEMCACHE
options.
    Type: The option value is of type INTMAX
          Range of Values: The minimum is 0 and the maximum is
9223372036854775807
          Valid Syntax(any casing): MIN|MAX|n|nK|nM|nG|nT|hexadecimal
    Numeric Format: Usage of LOGNUMBERFORMAT does not impact the value format
    When Can Set: Session startup (command line or config) only
    Restricted: Your Site Administrator can restrict modification of this option
    Optsave: PROC Optsave or command Dmoptsave will not 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

```
proc options option=msg expand;
run;
```

```
proc options option=msg noexpand;
run;
```

Program Description

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

Log 32.11 *Displaying an Expanded and Nonexpanded Pathname Using the OPTIONS Procedure*

```
6    proc options option=msg expand;
7    run;
    SAS (r) Proprietary Software Release xxx  TS1B0

 MSG=( 'C:\Program File\SAS\SAS9.3\sasmsg' )
                  The path to the sasmsg directory
NOTE: PROCEDURE OPTIONS used (Total process time):
      real time           0.01 seconds
      cpu time            0.00 seconds

8    proc options option=msg noexpand;
9    run;
    SAS (r) Proprietary Software Release xxx  TS1B0

 MSG=( '!sasroot\sasmsg')
                  The path to the sasmsg directory
```

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

```
proc options listinsertappend;
run;
```

Program Description

List all options that can be specified by the INSERT and APPEND options. The LISTINSERTAPPEND option provides a list and a description of these options.

```
proc options listinsertappend;
run;
```

Log

Log 32.12 *Displaying the Options That Can Be Specified by the INSERT and APPEND Options*

```
9    proc options listinsertappend;
10   run;

    SAS (r) Proprietary Software Release xxx   TS1B0


    Core options that can utilize INSERT and APPEND

     AUTOEXEC           Identifies AUTOEXEC files used during initialization
     CMPLIB             Identify previously compiled libraries of CMP
subroutines to use when linking
     FMTSEARCH          List of catalogs to search for formats and informats
     MAPS               Location of maps for use with SAS/GRAPH
     SASAUTOS           Search list for autocall macros
     SASHELP            Location of the SASHELP library
     SASSCRIPT          Location of SAS/CONNECT script files

    Host options that can utilize INSERT and APPEND

     HELPLOC            Location of help environment text and index files
     MSG                The path to the sasmsg directory
     SET                Defines an environment variable
```

Chapter 33

OPTLOAD Procedure

Overview: OPTLOAD Procedure

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 one of these methods:

- the DMOPTLOAD command from a command line in the SAS windowing environment. For example, DMOPTLOAD key= “core\options”.
- 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 Statement”	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".

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".

Example: Load a Data Set of Saved System Options

Features: PROC OPTLOAD 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 "c:\mysas";

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 "c:\mysas";
```

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

The following is the SAS log output after submitting the previous statements and procedures:

```
1    libname mysas "c:\mysas";
NOTE: Libref MYSAS was successfully assigned as follows:
      Engine:        V9
      Physical Name: c:\mysas

2    proc options option=yearcutoff;
3    run;

    SAS (r) Proprietary Software Release xxx  TS1B0

 YEARCUTOFF=1920   Cutoff year for DATE and DATETIME informats and functions
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 xxx  TS1B0

 YEARCUTOFF=2000   Cutoff year for DATE and DATETIME informats and functions
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 xxx  TS1B0

 YEARCUTOFF=1920   Cutoff year for DATE and DATETIME informats and functions
NOTE: PROCEDURE OPTIONS used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

Chapter 34

OPTSAVE Procedure

Overview: OPTSAVE Procedure

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 one of these methods:

- the DMOPTSAVE command from a command line in the SAS windowing environment. Use the command like this: DMOPTSAVE <*save-location*>.
- the PROC OPTSAVE statement.

Syntax: OPTSAVE Procedure

PROC OPTSAVE <*options* >;

Statement	Task	Example
"PROC OPTSAVE Statement"	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".

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.

Determining 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;
```

```
8    proc options option=pageno define;
9    run;

     SAS (r) Proprietary Software Release 9.3  TS1B0

 PAGENO=1
Option Definition Information for SAS Option PAGENO
    Group= LISTCONTROL
    Group Description: Procedure output and display settings
    Description: Beginning page number for the next page of output produced by
the SAS System
    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 does not impact 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
```

Creating a List of Options That Can Be Saved

Some system options cannot be saved. To create a list of options that can be saved, submit this SAS code:

```
proc optsave;
run;

proc print;
   var optname;
run;
```

The PRINT procedure uses the value of the _LAST_ system option to determine the input data set. The default data set name for the OPTSAVE procedure is SASUSER.MYOPTS.

Example: Saving System Options in a Data Set

Features: PROC OPTSAVE option:
OUT=

Details

This example saves the current system option settings using the OPTSAVE procedure.

Program

```
libname mysas "c:\mysas";


proc optsave out=mysas.options;
run;
```

Program Description

Create a libref.

```
libname mysas "c:\mysas";
```

Save the current system option settings.

```
proc optsave out=mysas.options;
run;
```

Log

The following is the SAS log output:

```
1    libname mysas "c:\mysas";
NOTE: Libref MYSAS was successfully assigned as follows:
      Engine:        V9
      Physical Name: c:\mysas

2    proc optsave out=mysas.options;
3    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
```

Chapter 35

PLOT Procedure

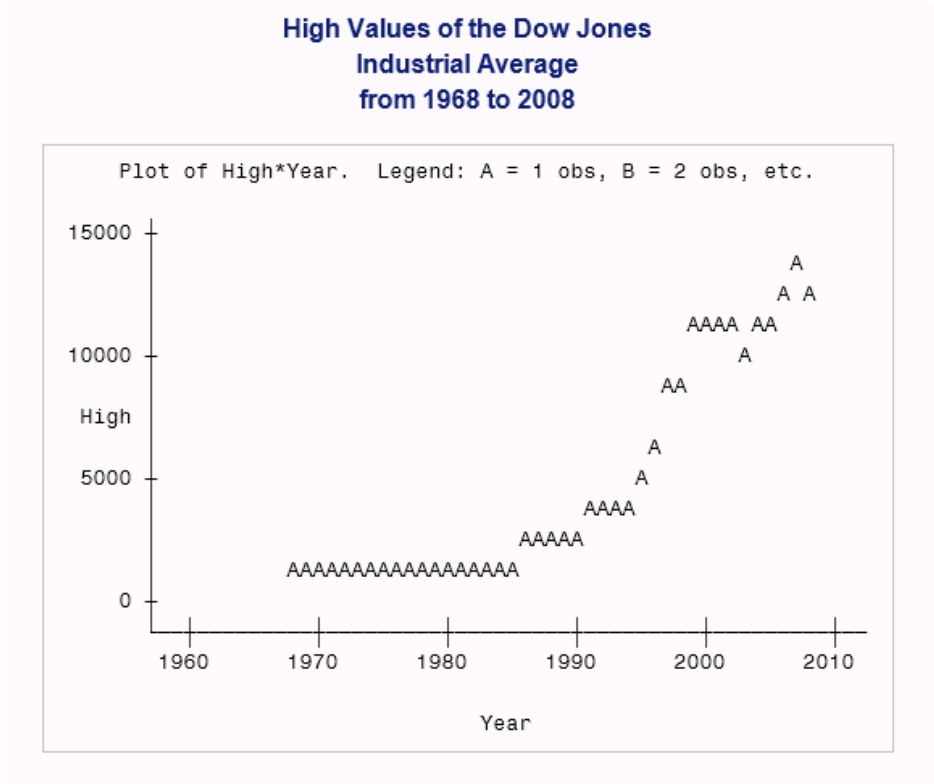
Overview: PLOT Procedure

The PLOT procedure plots the values of two variables for each observation in an input SAS data set. The coordinates of each point on the plot correspond to the two variables' values in one or more observations of the input data set.

The following output is a simple plot of the high values of the Dow Jones Industrial Average (DJIA) between 1968 and 2008. PROC PLOT determines the plotting symbol and the scales for the axes. Here are the statements that produce the output:

```
options nodate pageno=1 linesize=64
   pagesize=25;

proc plot data=djia;
   plot high*year;
   title 'High Values of the Dow Jones';
   title2 'Industrial Average';
   title3 'from 1968 to 2008';
run;
```

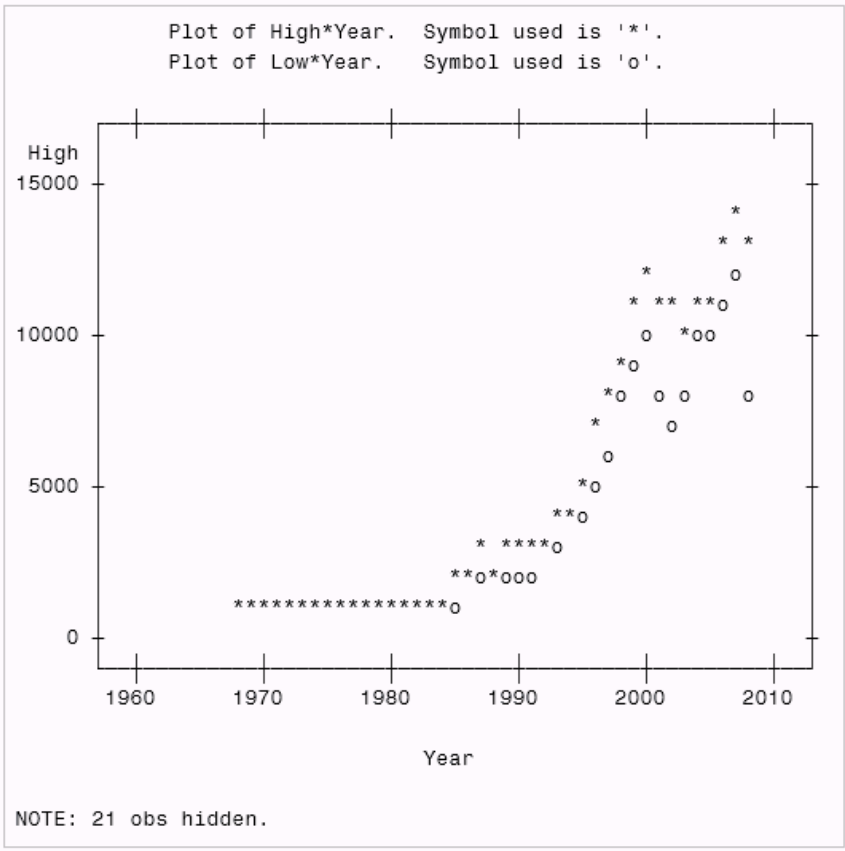
Output 35.1 *a Simple Plot*



You can also overlay two plots, as shown in the following output. One plot shows the high values of the DJIA; the other plot shows the low values. The plot also shows that you can specify plotting symbols and put a box around a plot. The statements that produce the following output are shown in "Example 3: Overlaying Two Plots" on page 924.

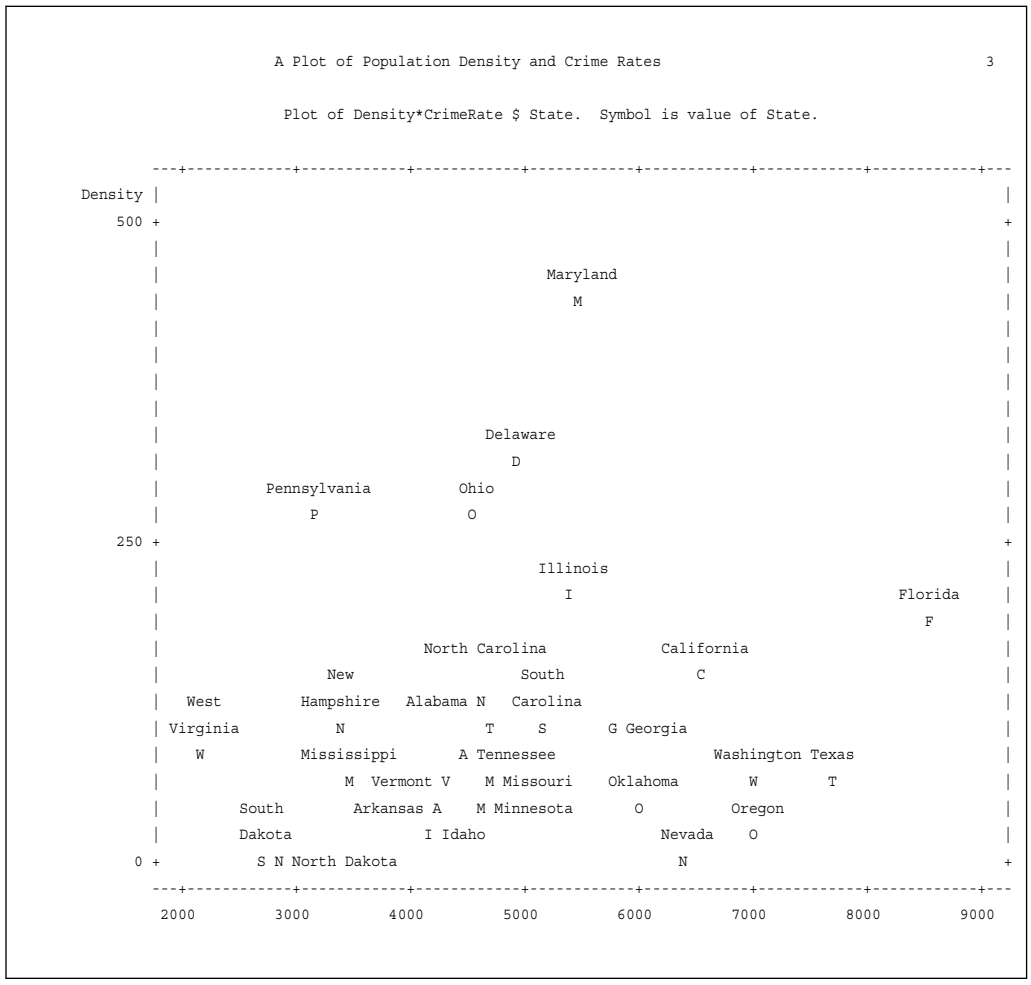
Output 35.2 *Plotting Two Sets of Values at Once*

Plot of Highs and Lows
for the Dow Jones Industrial Average



PROC PLOT can also label points on a plot with the values of a variable, as shown in the following output. The plotted data represents population density and crime rates for selected U.S. states. The SAS code that produces the following output is shown in "Example 11: Adjusting Labels on a Plot with the PLACEMENT= Option" on page 945.

Output 35.3 *Labeling Points on a Plot*



Concepts: PLOT Procedure

RUN Groups

PROC PLOT is an interactive procedure. It remains active after a RUN statement is executed. Usually, SAS terminates a procedure after executing a RUN statement. When you start the PLOT procedure, you can continue to submit any valid statements without resubmitting the PROC PLOT statement. Thus, you can easily experiment with changing labels, values of tick marks, and so on. Any options submitted in the PROC PLOT statement remain in effect until you submit another PROC PLOT statement.

When you submit a RUN statement, PROC PLOT executes all the statements submitted since the last PROC PLOT or RUN statement. Each group of statements is called a *RUN group*. With each RUN group, PROC PLOT begins a new page and begins with the first item in the VPERCENT= and HPERCENT= lists, if any.

To terminate the procedure, submit a QUIT statement, a DATA statement, or a PROC statement. Like the RUN statement, each of these statements completes a RUN group. If

you do not want to execute the statements in the RUN group, then use the RUN CANCEL statement, which terminates the procedure immediately.

You can use the BY statement interactively. The BY statement remains in effect until you submit another BY statement or terminate the procedure.

See "Example 11: Adjusting Labels on a Plot with the PLACEMENT= Option" on page 945 for an example of using RUN-group processing with PROC PLOT.

Generating Data with Program Statements

When you generate data to be plotted, a good rule is to generate fewer observations than the number of positions on the horizontal axis. PROC PLOT then uses the increment of the horizontal variable as the interval between tick marks.

Because PROC PLOT prints one character for each observation, using SAS program statements to generate the data set for PROC PLOT can enhance the effectiveness of continuous plots. For example, suppose that you want to generate data in order to plot the following equation, for x ranging from 0 to 100:

$$y = 2.54 + 3.83x$$

You can submit these statements:

```
options linesize=80;
 data generate;
    do x=0 to 100 by 2;
       y=2.54+3.83*x;
       output;
    end;
 run;
 proc plot data=generate;
    plot y*x;
 run;
```

If the plot is printed with a LINESIZE= value of 80, then about 75 positions are available on the horizontal axis for the X values. Thus, 2 is a good increment: 51 observations are generated, which is fewer than the 75 available positions on the horizontal axis.

However, if the plot is printed with a LINESIZE= value of 132, then an increment of 2 produces a plot in which the plotting symbols have space between them. For a smoother line, a better increment is 1, because 101 observations are generated.

Labeling Plot Points with Values of a Variable

Pointer Symbols

When you are using a label variable and do not specify a plotting symbol or if the value of the variable that you use as the plotting symbol is null ('00'x), PROC PLOT uses pointer symbols as plotting symbols. Pointer symbols associate a point with its label by pointing in the general direction of the label placement. PROC PLOT uses four different pointer symbols based on the value of the S= and V= suboptions in the PLACEMENT= option. The table below shows the pointer symbols:

Table 35.1 Pointer Symbols

S=	V=	Symbol
LEFT	any	<
RIGHT	any	>
CENTER	>0	∘
CENTER	<=0	v

If you are using pointer symbols and multiple points coincide, then PROC PLOT uses the number of points as the plotting symbol if the number of points is between 2 and 9. If the number of points is more than 9, then the procedure uses an asterisk (*).

Note: Because of character set differences among operating environments, the pointer symbol for S=CENTER and V>0 might differ from the one shown here.

Understanding Penalties

PROC PLOT assesses the quality of placements with penalties. If all labels are plotted with zero penalty, then no labels collide and all labels are near their symbols. When it is not possible to place all labels with zero penalty, PROC PLOT tries to minimize the total penalty. The following table gives a description of the penalty, the default value of the penalty, the index that you use to reference the penalty, and the range of values that you can specify if you change the penalties. Each penalty is described in more detail in Table 35.3 on page 899.

Table 35.2 Penalties Table

Penalty	Default Penalty	Index	Range
Not placing a blank	1	1	0-500
Bad split, no split character specified	1	2	0-500
Bad split with split character	50	3	0-500
Free horizontal shift, *fhs*	2	4	0-500
Free vertical shift, *fvs*	1	5	0-500
Vertical shift weight, *vsw*	2	6	0-500
Vertical or horizontal shift denominator, *vhsd*	5	7	1-500
Collision state	500	8	0-10,000
(Reserved for future use)		9-14	
Not placing the first character	11	15	0-500
Not placing the second character	10	16	0-500

Penalty	Default Penalty	Index	Range
Not placing the third character	8	17	0-500
Not placing the fourth character	5	18	0-500
Not placing the fifth through 200th character	2	19-214	0-500

The following table contains the index values from the previous table with a description of the corresponding penalty.

Table 35.3 *Index Values for Penalties*

1	A nonblank character in the plot collides with an embedded blank in a label, or there is not a blank or a plot boundary before or after each label fragment.
2	A split occurs on a nonblank or nonpunctuation character when you do not specify a split character.
3	A label is placed with a different number of lines than the L= suboption specifies, when you specify a split character.
4-7	A label is placed far away from the corresponding point. PROC PLOT calculates the penalty according to this (integer arithmetic) formula: $[MAX(\lvert H \rvert - fhs, 0) + vsw \times MAX(\lvert V \rvert - (L + fvs + (V > 0))/2, 0)]/vhsd$ Notice that penalties 4 through 7 are actually just components of the formula used to determine the penalty. Changing the penalty for a free horizontal or free vertical shift to a large value such as 500 removes any penalty for a large horizontal or vertical shift. "Example 6: Plotting Date Values on an Axis" on page 930 illustrates a case in which removing the horizontal shift penalty is useful.
8	A label might collide with its own plotting symbol. If the plotting symbol is blank, then a collision state cannot occur. See "Collision States" on page 900 for more information.
15-214	A label character does not appear in the plot. By default, the penalty for not printing the first character is greater than the penalty for not printing the second character, and so on. By default, the penalty for not printing the fifth and subsequent characters is the same.

Note: Labels can share characters without penalty.

Changing Penalties

You can change the default penalties with the PENALTIES= option in the PLOT statement. Because PROC PLOT considers penalties when it places labels, changing the default penalties can change the placement of the labels. For example, if you have labels that all begin with the same two-letter prefix, then you might want to increase the default penalty for not printing the third, fourth, and fifth characters and decrease the penalties for not printing the first and second characters . In the following example, the PENALTIES= option increases the default penalty for not printing the third, fourth, and fifth characters to 11, 10, and 8 and it decreases the penalties for not printing the first and second characters to 2

```
penalties(15 to 20)=2 2 11 10 8 2
```

This example extends the penalty list. The 20th penalty of 2 is the penalty for not printing the sixth through 200th character. When the last index i is greater than 18, the last penalty is used for the $(i - 14)$th character and beyond.

You can also extend the penalty list by just specifying the starting index. For example, the following PENALTIES= option is equivalent to the one above:

```
penalties(15)=2 2 11 10 8 2
```

Collision States

Collision states are placement states that can cause a label to collide with its own plotting symbol. PROC PLOT usually avoids using collision states because of the large default penalty of 500 that is associated with them. PROC PLOT does not consider the actual length or splitting of any particular label when determining if a placement state is a collision state. The following are the rules that PROC PLOT uses to determine collision states:

- When S=CENTER, placement states that do not shift the label up or down sufficiently so that all of the label is shifted onto completely different lines from the symbol are collision states.
- When S=RIGHT, placement states that shift the label zero or more positions to the left without first shifting the label up or down onto completely different lines from the symbol are collision states.
- When S=LEFT, placement states that shift the label zero or more positions to the right without first shifting the label up or down onto completely different lines from the symbol are collision states.

Note: A collision state cannot occur if you do not use a plotting symbol.

Reference Lines

PROC PLOT places labels and computes penalties before placing reference lines on a plot. The procedure does not attempt to avoid rows and columns that contain reference lines.

Hidden Label Characters

In addition to the number of hidden observations and hidden plotting symbols, PROC PLOT prints the number of hidden label characters. Label characters can be hidden by plotting symbols or other label characters.

Overlaying Label Plots

When you overlay a label plot and a nonlabel plot, PROC PLOT tries to avoid collisions between the labels and the characters of the nonlabel plot. When a label character collides with a character in a nonlabel plot, PROC PLOT adds the usual penalty to the penalty sum.

When you overlay two or more label plots, all label plots are treated as a single plot in avoiding collisions and computing hidden character counts. Labels of different plots never overprint, even with the OVP system option in effect.

Computational Resources Used for Label Plots

This section uses the following variables to discuss how much time and memory PROC PLOT uses to construct label plots:

n
number of points with labels.

len
constant length of labels.

s
number of label pieces, or fragments.

p
number of placement states specified in the PLACE= option.

Time

For a given plot size, the time that is required to construct the plot is approximately proportional to $n \times len$. The amount of time required to split the labels is approximately proportional to ns^2 . Generally, the more placement states that you specify, the more time that PROC PLOT needs to place the labels. However, increasing the number of horizontal and vertical shifts gives PROC PLOT more flexibility to avoid collisions, often resulting in less time used to place labels.

Memory

PROC PLOT uses $24p$ bytes of memory for the internal placement state list. PROC PLOT uses $n(84 + 5len + 4s(1 + 1.5(s + 1)))$ bytes for the internal list of labels. PROC PLOT builds all plots in memory; each printing position uses one byte of memory. If you run out of memory, then request fewer plots in each PLOT statement and put a RUN statement after each PLOT statement.

Specifying Variable Lists in Plot Requests

You can use SAS variable lists in plot requests. For example, the following are valid plot requests:

Table 35.4 *Plot Requests*

Plot Request	What is Plotted
`(a - - d)`	`a*b a*c a*d b*c b*d c*d`
`(x1 - x4)`	`x1*x2 x1*x3 x1*x4 x2*x3 x2*x4 x3*x4`
`(_numeric_)`	All combinations of numeric variables
`y*(x1 - x4)`	`y*x1 y*x2 y*x4 y*x4`

If both the vertical and horizontal specifications request more than one variable and if a variable appears in both lists, then it will not be plotted against itself. For example, the following statement does not plot B*B and C*C:

```
plot (a b c)*(b c d);
```

Specifying Combinations of Variables

The operator in request is either an asterisk (*) or a colon (:). An asterisk combines the variables in the lists to produce all possible combinations of *x* and *y* variables. For example, the following plot requests are equivalent:

```
plot (y1-y2) * (x1-x2);

plot y1*x1 y1*x2 y2*x1 y2*x2;
```

A colon combines the variables pairwise. Thus, the first variables of each list combine to request a plot, as do the second, third, and so on. For example, the following plot requests are equivalent:

```
plot (y1-y2) : (x1-x2);

plot y1*x1 y2*x2;
```

Syntax: PLOT Procedure

Requirement: At least one PLOT statement is required.

Tips: Supports RUN-group processing.

Supports the Output Delivery System. See Chapter 3, "Output Delivery System: Basic Concepts," in *SAS Output Delivery System: User's Guide*.

You can use the ATTRIB, FORMAT, LABEL, and WHERE statements. See "Statements with the Same Function in Multiple Procedures" on page 35 for details. You can also use any global statements. See "Global Statements" on page 20 for a list.

PROC PLOT <*option(s)*>;
  BY <DESCENDING> *variable-1*
  <…<DESCENDING>*variable-n*>
  <NOTSORTED>;
  PLOT *plot-request(s)* </ *option(s)*>;

Statement	Task	Example
"PROC PLOT Statement"	Request the plots be produced	Ex. 10
"BY Statement"	Produce a separate plot for each BY group	Ex. 8
"PLOT Statement"	Describe the plots that you want	Ex. 1, Ex. 2, Ex. 3, Ex. 4, Ex. 5, Ex. 6, Ex. 7, Ex. 8, Ex. 9, Ex. 11, Ex. 12, Ex. 13

PROC PLOT Statement

Requests the plots be produced.

Tip: You can use data set options with the DATA= option. See "Data Set Options" on page 19 for a list.

Example: "Example 10: Excluding Observations That Have Missing Values" on page 943

Syntax

PROC PLOT <*option(s)*>;

Summary of Optional Arguments

DATA=*SAS-data-set*
specifies the input data set.

Control the appearance of the plot

FORMCHAR <(*position(s)*)>='*formatting-character(s)*'
specifies the characters that construct the borders of the plot.

NOLEGEND
suppresses the legend at the top of the plot.

VTOH=*aspect-ratio*
specifies the aspect ratio of the characters on the output device.

Control the axes

MISSING
includes missing character variable values.

NOMISS
excludes observations with missing values.

UNIFORM
uniformly scales axes across BY groups.

Control the size of the plot

HPERCENT=*percent(s)*
specifies the percentage of the available horizontal space for each plot.

VPERCENT=*percent(s)*
specifies the percentage of the available vertical space for each plot.

Optional Arguments

DATA=*SAS-data-set*
specifies the input SAS data set.

See: Chapter 2, "Fundamental Concepts for Using Base SAS Procedures," on page 17

FORMCHAR <(*position(s)*)>='*formatting-character(s)*'
defines the characters to use for constructing the borders of the plot.

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 twenty possible SAS formatting characters, in order.

Range: PROC PLOT uses formatting characters 1, 2, 3, 5, 7, 9, and 11. The following table shows the formatting characters that PROC PLOT uses.

Table 35.5 Character Positions

Position	Default	Used to Draw
1	|	Vertical separators
2	-	Horizontal separators
3 5 9 1 1	-	Corners
7	+	Intersection of vertical and horizontal separators

formatting-character(s)
lists the characters to use for the specified positions. PROC PLOT 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 pound sign (#) to the seventh character, and does not alter the remaining characters: `formchar(3,7)='*#'`

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 2D to the third formatting character, 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 plots with no borders, for example: `formchar (1,2,7)=''`

HPERCENT=*percent(s)*
specifies one or more percentages of the available horizontal space to use for each plot. HPERCENT= enables you to put multiple plots on one page. PROC PLOT tries to fit as many plots as possible on a page. After using each of the *percent(s)*, PROC PLOT cycles back to the beginning of the list. A zero in the list forces PROC PLOT to go to a new page even if it could fit the next plot on the same page.

HPERCENT=33
prints three plots per page horizontally; each plot is one-third of a page wide.

HPERCENT=50 25 25
prints three plots per page; the first is twice as wide as the other two.

HPERCENT=33 0
produces plots that are one-third of a page wide,; each plot is on a separate page.

HPERCENT=300
produces plots three pages wide.

At the beginning of every BY group and after each RUN statement, PROC PLOT returns to the beginning of the *percent(s)* and starts printing a new page.

Alias: HPCT=

Default: 100

Example: "Example 4: Producing Multiple Plots per Page" on page 926

MISSING
includes missing character variable values in the construction of the axes. It has no effect on numeric variables.

Interaction: overrides the NOMISS option for character variables.

NOLEGEND
suppresses the legend at the top of each plot. The legend lists the names of the variables being plotted and the plotting symbols used in the plot.

NOMISS
excludes observations for which either variable is missing from the calculation of the axes. Normally, PROC PLOT draws an axis based on all the values of the variable being plotted, including points for which the other variable is missing.

Interactions:
The HAXIS= option overrides the effect of NOMISS on the horizontal axis. The VAXIS= option overrides the effect on the vertical axis.

NOMISS is overridden by MISSING for character variables.

Example: "Example 10: Excluding Observations That Have Missing Values" on page 943

UNIFORM
uniformly scales axes across BY groups. Uniform scaling enables you to directly compare the plots for different values of the BY variables.

Restriction: You cannot use PROC PLOT with the UNIFORM option with an engine that supports concurrent access if another user is updating the data set at the same time.

VPERCENT=*percent(s)*
specifies one or more percentages of the available vertical space to use for each plot. If you use a percentage greater than 100, then PROC PLOT prints sections of the plot on successive pages.

Alias: VPCT=

Default: 100

See: HPERCENT=

Example: "Example 4: Producing Multiple Plots per Page" on page 926

VTOH=*aspect-ratio*
specifies the aspect ratio (vertical to horizontal) of the characters on the output device. *aspect-ratio* is a positive real number. If you use the VTOH= option, then PROC PLOT spaces tick marks so that the distance between horizontal tick marks is nearly equal to the distance between vertical tick marks. For example, if characters are twice as high as they are wide, then specify VTOH=2.

Interaction: VTOH= has no effect if you use the HSPACE= and the VSPACE= options in the PLOT statement.

Note: The minimum value allowed is 0.

See: "HAXIS=axis-specification " on page 909 for a way to equate axes so that the given distance represents the same data range on both axes.

BY Statement

Produces a separate plot and starts a new page for each BY group.

See: "BY" on page 36

Example: "Example 8: Plotting BY Groups" on page 936

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 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 data is 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.

PLOT Statement

Requests the plots to be produced by PROC PLOT.

Tip: You can use multiple PLOT statements.

Examples:
"Example 1: Specifying a Plotting Symbol" on page 919
"Example 2: Controlling the Horizontal Axis and Adding a Reference Line" on page 922
"Example 3: Overlaying Two Plots" on page 924
"Example 4: Producing Multiple Plots per Page" on page 926
"Example 5: Plotting Data on a Logarithmic Scale" on page 929
"Example 6: Plotting Date Values on an Axis" on page 930
"Example 7: Producing a Contour Plot" on page 932
"Example 8: Plotting BY Groups" on page 936
"Example 9: Adding Labels to a Plot" on page 940

Syntax

PLOT *plot-request(s)* </ *option(s)*>;

Summary of Optional Arguments

BOX
: puts a box around the plot.

OVERLAY
: overlays plots.

Control the axes

HAXIS=*axis-specification*
: specifies the tick-mark values for the horizontal axis.

HEXPAND
: expands the horizontal axis.

HPOS=*axis-length*
: specifies the number of print positions on the horizontal axis.

HREVERSE
: reverses the order of the values on the horizontal axis.

HSPACE=*n*
: specifies the distance between tick marks on the horizontal axis.

HZERO
: assigns a value of zero to the first tick mark on the horizontal axis.

VAXIS=*axis-specification*
: specifies the tick-mark values for the vertical axis.

VEXPAND
: expands the vertical axis.

VPOS=*axis-length*
: specifies the number of print positions on the vertical axis.

VREVERSE
: reverses the order of the values on the vertical axis.

VSPACE=*n*
: specifies the distance between tick marks on the vertical axis.

VZERO
: assigns a value of zero to the first tick mark on the vertical axis.

Label points on a plot

LIST<=*penalty-value*>
: lists the penalty and the placement state of the points.

OUTWARD='*character*'
: forces the labels away from the origin.

PENALTIES<(*index-list*)>=*penalty-list*
: changes default penalties.

PLACEMENT=(*expression(s)*)

specifies locations for the placement of the labels.

SPLIT='*split-character*'
specifies a split character for the label.

STATES
lists all placement states in effect.

Produce a contour plot

CONTOUR<=*number-of-levels*>
draws a contour plot.

S*contour-level*='*character-list*'
specifies the plotting symbol for one contour level.

Produce a countour plot

SLIST='*character-list-1*' <…'*character-list-n*'>
specifies the plotting symbol for multiple contour levels.

Specify reference lines

HREF=*value-specification*
draws a line perpendicular to the specified values on the horizontal axis.

HREFCHAR='*character*'
specifies a character to use to draw the horizontal reference line.

VREF=*value-specification*
draws a line perpendicular to the specified values on the vertical axis.

VREFCHAR='*character*'
specifies a character to use to draw the vertical reference line.

Required Argument

plot-request(s)
specifies the variables (vertical and horizontal) to plot and the plotting symbol to use to mark the points on the plot.

Each form of *plot-request(s)* supports a label variable. A label variable is preceded by a dollar sign ($) and specifies a variable whose values label the points on the plot. For example:

```
plot y*x $ label-variable

plot y*x='*' $ label-variable
```

For more information, see "Labeling Plot Points with Values of a Variable" on page 897. In addition, see "Example 9: Adding Labels to a Plot" on page 940 and all the examples that follow it.

The *plot-request(s)* can be one or more of the following:

*vertical*horizontal* <$ *label-variable*>
specifies the variable to plot on the vertical axis and the variable to plot on the horizontal axis.

For example, the following statement requests a plot of Y by X:

```
plot y*x;
```

Y appears on the vertical axis, X on the horizontal axis.

This form of the plot request uses the default method of choosing a plotting symbol to mark plot points. When a point on the plot represents the values of one observation in the data set, PROC PLOT puts the character A at that point. When

a point represents the values of two observations, the character B appears. When a point represents values of three observations, the character C appears, and so on, through the alphabet. The character Z is used for the occurrence of 26 or more observations at the same printing position.

*vertical*horizontal='character'* <$ *label-variable*>
specifies the variables to plot on the vertical and horizontal axes and specifies a plotting symbol to mark each point on the plot. A single character is used to represent values from one or more observations.

For example, the following statement requests a plot of Y by X, with each point on the plot represented by a plus sign (+):

```
plot y*x='+';
```

*vertical*horizontal=variable* <$ *label-variable*>
specifies the variables to plot on the vertical and horizontal axes and specifies a variable whose values are to mark each point on the plot. The variable can be either numeric or character. The first (left-most) nonblank character in the formatted value of the variable is used as the plotting symbol (even if more than one value starts with the same letter). When more than one observation maps to the same plotting position, the value from the first observation marks the point. For example, in the following statement GENDER is a character variable with values of **FEMALE** and **MALE**; the values **F** and **M** mark each observation on the plot.

```
plot height*weight=gender;
```

See: "Specifying Variable Lists in Plot Requests" on page 901 and "Specifying Combinations of Variables" on page 901

Optional Arguments

BOX
draws a border around the entire plot, rather than just on the left side and bottom.

Example: "Example 3: Overlaying Two Plots" on page 924

CONTOUR<=*number-of-levels*>
draws a contour plot using plotting symbols with varying degrees of shading where *number-of-levels* is the number of levels for dividing the range of *variable*. The plot request must be of the form *vertical*horizontal=variable* where *variable* is a numeric variable in the data set. The intensity of shading is determined by the values of this variable.

When you use CONTOUR, PROC PLOT does not plot observations with missing values for *variable*.

Overprinting, if it is enabled by the OVP system option, is used to produce the shading. Otherwise, single characters varying in darkness are used. The CONTOUR option is most effective when the plot is dense.

Default: 10

Range: 1-10

Example: "Example 7: Producing a Contour Plot" on page 932

HAXIS=*axis-specification*
specifies the tick-mark values for the horizontal axis.

- For numeric values, *axis-specification* is either an explicit list of values, a BY increment, or a combination of both:

  n <…*n*>

BY *increment*

n TO *n* BY *increment*

The values must be in either ascending or descending order. Use a negative value for *increment* to specify descending order. The specified values are spaced evenly along the horizontal axis even if the values are not uniformly distributed. Numeric values can be specified in the following ways:

HAXIS= value	Comments
`10 to 100 by 5`	Values appear in increments of 5, starting at 10 and ending at 100.
`by 5`	Values are incremented by 5. PROC PLOT determines the minimum and maximum values for the tick marks.
`10 100 1000 10000`	Values are not uniformly distributed. This specification produces a logarithmic plot. If PROC PLOT cannot determine the function implied by the axis specification, it uses simple linear interpolation between the points. To determine whether PROC PLOT correctly interpolates a function, you can use the DATA step to generate data that determines the function and see whether it appears linear when plotted. See "Example 5: Plotting Data on a Logarithmic Scale" on page 929 for an example.
`1 2 10 to 100 by 5`	A combination of the previous specifications.

- *For character variables*, *axis-specification* is a list of unique values that are enclosed in quotation marks:

  '*value-1*' <…'*value-n*'>

  For example,

  ```
  haxis='Paris' 'London' 'Tokyo'
  ```

  The character strings are case sensitive. If a character variable has an associated format, then *axis-specification* must specify the formatted value. The values can appear in any order.

- *For axis variables that contain date-time values*, *axis-specification* is either an explicit list of values or a starting and an ending value with an increment specified:

  '*date-time-value*'*i* <…'*date-time-value*'*i*>

  '*date-time-value*'*i* TO <…'*date-time-value*'*i*>
  <BY *increment*>

'date-time-value'i
any SAS date, time, or datetime value described for the SAS functions INTCK and INTNX. The suffix *i* is one of the following:

D date

T time

DT datetime

increment
one of the valid arguments for the INTCK or INTNX functions. *increment* can be one of the following:

For dates	For datetimes	For times
DAY	DTDAY	HOUR
WEEK	DTWEEK	MINUTE
MONTH	DTMONTH	SECOND
QTR	DTQTR	
YEAR	DTYEAR	

For example,

```
haxis='01JAN95'd to '01JAN96'd
by month

haxis='01JAN95'd to '01JAN96'd
by qtr
```

Note: You must use a FORMAT statement to print the tick-mark values in an understandable form.

Interaction: You can use the HAXIS= and VAXIS= options with the VTOH= option to equate axes. If your data is suitable, then use HAXIS=BY *n* and VAXIS=BY *n* with the same value for *n* and specify a value for the VTOH= option. The number of columns that separate the horizontal tick marks is nearly equal to the number of lines that separate the vertical tick marks times the value of the VTOH= option. In some cases, PROC PLOT cannot simultaneously use all three values and changes one or more of the values.

Examples:
"Example 2: Controlling the Horizontal Axis and Adding a Reference Line" on page 922

"Example 5: Plotting Data on a Logarithmic Scale" on page 929

"Example 6: Plotting Date Values on an Axis" on page 930

HEXPAND
expands the horizontal axis to minimize the margins at the sides of the plot and to maximize the distance between tick marks, if possible.

HEXPAND causes PROC PLOT to ignore information about the spacing of the data. Plots produced with this option waste less space but can obscure the nature of the relationship between the variables.

HPOS=*axis-length*
specifies the number of print positions on the horizontal axis. The maximum value of *axis-length* that allows a plot to fit on one page is three positions less than the value of the LINESIZE= system option because there must be space for the procedure to print information next to the vertical axis. The exact maximum depends on the number of characters that are in the vertical variable's values. If *axis-length* is too large to fit on a line, then PROC PLOT ignores the option.

HREF=*value-specification*
draws lines on the plot perpendicular to the specified values on the horizontal axis. PROC PLOT includes the values that you specify with the HREF= option on the horizontal axis unless you specify otherwise with the HAXIS= option.

For the syntax for *value-specification*, see "HAXIS=axis-specification " on page 909.

Example: "Example 8: Plotting BY Groups" on page 936

HREFCHAR='*character*'
specifies the character to use to draw the horizontal reference line.

Default: vertical bar (|)

See: "FORMCHAR <(position(s))>='formatting-character(s)' " on page 903 and "HREF=value-specification " on page 912

HREVERSE
reverses the order of the values on the horizontal axis.

HSPACE=*n*
specifies that a tick mark will occur on the horizontal axis at every *n*th print position, where *n* is the value of HSPACE=.

HZERO
assigns a value of zero to the first tick mark on the horizontal axis.

Interaction: PROC PLOT ignores HZERO if the horizontal variable has negative values or if the HAXIS= option specifies a range that does not begin with zero.

LIST<=*penalty-value*>
lists the horizontal and vertical axis values, the penalty, and the placement state of all points plotted with a penalty greater than or equal to *penalty-value*. If no plotted points have a penalty greater than or equal to *penalty-value*, then no list is printed.

Tip: LIST is equivalent to LIST=0.

See: "Understanding Penalties" on page 898

Example: "Example 11: Adjusting Labels on a Plot with the PLACEMENT= Option" on page 945

OUTWARD='*character*'
tries to force the point labels outward, away from the origin of the plot, by protecting positions next to symbols that match *character* that are in the direction of the origin (0,0). The algorithm tries to avoid putting the labels in the protected positions, so they usually move outward.

Tip: This option is useful only when you are labeling points with the values of a variable.

OVERLAY
overlays all plots that are specified in the PLOT statement on one set of axes. The variable names, or variable labels if they exist, from the first plot are used to label the axes. Unless you use the HAXIS= or the VAXIS= option, PROC PLOT automatically scales the axes in the way that best fits all the variables.

When the SAS system option OVP is in effect and overprinting is allowed, the plots are superimposed. Otherwise, when NOOVP is in effect, PROC PLOT uses the plotting symbol from the first plot to represent points that appear in more than one plot. In such a case, the output includes a message telling you how many observations are hidden.

Example: "Example 3: Overlaying Two Plots" on page 924

PENALTIES<(*index-list*)>=*penalty-list*
changes the default penalties. The *index-list* provides the positions of the penalties in the list of penalties. The *penalty-list* contains the values that you are specifying for the penalties that are indicated in the *index-list*. The *index-list* and the *penalty-list* can contain one or more integers. In addition, both *index-list* and *penalty-list* accept the form: **`value TO value`**

See: "Understanding Penalties" on page 898

Example: "Example 13: Changing a Default Penalty" on page 951

PLACEMENT=(*expression(s)*)
controls the placement of labels by specifying possible locations of the labels relative to their coordinates. Each *expression* consists of a list of one or more suboptions (H=, L=, S=, or V=) that are joined by an asterisk (*) or a colon (:). PROC PLOT uses the asterisk and colon to expand each expression into combinations of values for the four possible suboptions. The asterisk creates every possible combination of values in the expression list. A colon creates only pairwise combinations. The colon takes precedence over the asterisk. With the colon, if one list is shorter than the other, then the values in the shorter list are reused as necessary.

Use the following suboptions to control the placement:

H=*integer(s)*
specifies the number of horizontal spaces (columns) to shift the label relative to the starting position. Both positive and negative integers are valid. Positive integers shift the label to the right; negative integers shift it to the left. For example, you can use the H= suboption in the following way:

```
place=(h=0 1 -1 2 -2)
```

You can use the keywords BY ALT in this list. BY ALT produces a series of numbers whose signs alternate between positive and negative and whose absolute values change by one after each pair. For example, the following PLACE= specifications are equivalent:

```
place=(h=0 -1 to -3 by alt)
place=(h=0 -1 1 -2 2 -3 3)
```

If the series includes zero, then the zero appears twice. For example, the following PLACE= options are equivalent:

```
place=(h= 0 to 2 by alt)
place=(h=0 0 1 -1 2 -2)
```

Default: H=0

Range: −500 to 500

L=*integer(s)*
specifies the number of lines onto which the label can be split.

Default: L=1

Range: 1-200

S=*start-position(s)*
specifies where to start printing the label. The value for *start-position* can be one or more of the following:

CENTER
the procedure centers the label around the plotting symbol.

RIGHT
the label starts at the plotting symbol location and continues to the right.

LEFT
the label starts to the left of the plotting symbol and ends at the plotting symbol location.

Default: CENTER

V=*integer(s)*
specifies the number of vertical spaces (lines) to shift the label relative to the starting position. V= behaves the same as the H= suboption, described earlier.

A new expression begins when a suboption is not preceded by an operator. Parentheses around each expression are optional. They make it easier to recognize individual expressions in the list. However, the entire expression list must be in parentheses, as shown in the following example. The following table shows how this expression is expanded and describes each placement state.

```
place=((v=1)
    (s=right left : h=2 -2)
    (v=-1)
    (h=0 1 to 2 by alt * v=1 -1)
    (l=1 to 3 * v=1 to 2 by alt *
     h=0 1 to 2 by alt))
```

Each combination of values is a *placement state*. The procedure uses the placement states in the order in which they appear in the placement states list, so specify your most preferred placements first. For each label, the procedure tries all states, then it uses the first state that places the label with minimum penalty. When all labels are initially placed, the procedure cycles through the plot multiple times, systematically refining the placements. The refinement step tries to both minimize the penalties and to use placements nearer to the beginning of the states list. However, PROC PLOT uses a heuristic approach for placements, so the procedure does not always find the best set of placements.

Table 35.6 *Expanding an Expression List into Placement States*

Expression	Placement State	Meaning
(V=1)	S=CENTER L=1 H=0 V=1	Center the label, relative to the point, on the line above the point. Use one line for the label.
(S=RIGHT LEFT : H=2 −2)	S=RIGHT L=1 H=2 V=0	Begin the label in the second column to the right of the point. Use one line for the label.
	S=LEFT L=1 H=−2 V=0	End the label in the second column to the left of the point. Use one line for the label.
(V=−1)	S=CENTER L=1 H=0 V=− 1	Center the label, relative to the point, on the line below the point. Use one line for the label.
(H=0 1 to 2 BY ALT * V=1 −1)	S=CENTER L=1 H=0 V=1	Center the label, relative to the point, on the line above the point.
	S=CENTER L=1 H=0 V=−1	Center the label, relative to the point, on the line below the point.

Expression	Placement State	Meaning
	S=CENTER L=1 H=1 V=1	From center, shift the label one column to the right on the line above the point.
	S=CENTER L=1 H=1 V=−1	From center, shift the label one column to the right on the line below the point.
	S=CENTER L=1 H=−1 V=1	From center, shift the label one column to the left on the line above the point.
	S=CENTER L=1 H=− 1 V=−1	From center, shift the label one column to the left on the line below the point.
	S=CENTER L=1 H=2 V=1 S=CENTER L=1 H=2 V=−1	From center, shift the labels two columns to the right, first on the line above the point, then on the line below.
	S=CENTER L=1 H=−2 V=1 S=CENTER L=1 H=−2 V=−1	From center, shift the labels two columns to the left, first on the line above the point, then on the line below.
(L=1 to 3 * V=1 to 2 BY ALT * H=0 1 to 2 BY ALT)	S=CENTER L=1 H=0 V=1	Center the label, relative to the point, on the line above the point. Use one line for the label.
	S=CENTER L=1 H=1 V=1 S=CENTER L=1 H=−1 V=1 S=CENTER L=1 H=2 V=1 S=CENTER L=1 H=−2 V=1	From center, shift the label one or two columns to the right or left on the line above the point. Use one line for the label.
	S=CENTER L=1 H=0 V=−1	Center the label, relative to the point, on the line below the point. Use one line for the label.
	S=CENTER L=1 H=1 V=−1 S=CENTER L=1 H=−1 V=−1 S=CENTER L=1 H=2 V=−1 S=CENTER L=1 H=−2 V=−1	From center, shift the label one or two columns to the right and the left on the line below the point.
	.	Use the same horizontal shifts on the line two lines above the point and on the line two lines below the point.
	S=CENTER L=1 H=− 2 V=−2	
	S=CENTER L=2 H=0 V=1	Repeat the whole process splitting the label over two lines. Then repeat it splitting the label over three lines.

Expression	Placement State	Meaning
	S=CENTER L=3 H=− 2 V=−2	

Alias: PLACE=

Default: There are two defaults for the PLACE= option. If you are using a blank as the plotting symbol, then the default placement state is PLACE=(S=CENTER : V=0 : H=0 : L=1), which centers the label. If you are using anything other than a blank, then the default is PLACE=((S=RIGHT LEFT : H=2 −2) (V=1 −1 * H=0 1 -1 2 -2)). The default for labels placed with symbols includes multiple positions around the plotting symbol so the procedure has flexibility when placing labels on a crowded plot.

Tip: Use the STATES option to print a list of placement states.

See: "Labeling Plot Points with Values of a Variable" on page 897

Examples:

"Example 11: Adjusting Labels on a Plot with the PLACEMENT= Option" on page 945

"Example 12: Adjusting Labeling on a Plot with a Macro" on page 949

S*contour-level*='*character-list*'

specifies the plotting symbol to use for a single contour level. When PROC PLOT produces contour plots, it automatically chooses the symbols to use for each level of intensity. You can use the S= option to override these symbols and specify your own. You can include up to three characters in *character-list*. If overprinting is not allowed, then PROC PLOT uses only the first character.

For example, to specify three levels of shading for the Z variable, use the following statement:

```
plot y*x=z /
    contour=3 s1='A' s2='+' s3='X0A';
```

You can also specify the plotting symbols as hexadecimal constants:

```
plot y*x=z /
    contour=3 s1='7A'x  s2='7F'x s3='A6'x;
```

This feature was designed especially for printers where the hexadecimal constants can represent gray scale fill characters.

Range: 1 to the highest contour level (determined by the CONTOUR option).

See: SLIST= and CONTOUR

SLIST='*character-list-1*' <…'*character-list-n*'>

specifies plotting symbols for multiple contour levels. Each *character-list* specifies the plotting symbol for one contour level: the first *character-list* for the first level, the second *character-list* for the second level, and so on. For example:

```
plot y*x=z /
     contour=5  slist='.' ':' '!' '=' '+O';
```

If you omit a plotting symbol for each contour level, then PROC PLOT uses the default symbols:

```
slist='.' ',' '-' '=' '+' 'O' 'X'       'W' '*' '#'
```

Restriction: If you use the SLIST= option, then it must be listed last in the PLOT statement.

See: S*contour-level*= and CONTOUR=

SPLIT='*split-character*'
when labeling plot points, specifies where to split the label when the label spans two or more lines. The label is split onto the number of lines that is specified in the L= suboption to the PLACEMENT= option. If you specify a split character, then the procedure always splits the label on each occurrence of that character, even if it cannot find a suitable placement. If you specify L=2 or more but do not specify a split character, then the procedure tries to split the label on blanks or punctuation but will split words if necessary.

PROC PLOT shifts split labels as a block, not as individual fragments (a *fragment* is the part of the split label that is contained on one line). For example, to force `This is a label` to split after the `a` , change it to `This is a*label` and specify `SPLIT='*'` .

See: "Labeling Plot Points with Values of a Variable" on page 897

STATES
lists all the placement states in effect. STATES prints the placement states in the order in which you specify them in the PLACE= option.

VAXIS=*axis-specification*
specifies tick mark values for the vertical axis. VAXIS= follows the same rules as the HAXIS= option.

Examples:
"Example 7: Producing a Contour Plot" on page 932
"Example 12: Adjusting Labeling on a Plot with a Macro" on page 949

VEXPAND
expands the vertical axis to minimize the margins above and below the plot and to maximize the space between vertical tick marks, if possible.

See: "HEXPAND " on page 911

VPOS=*axis-length*
specifies the number of print positions on the vertical axis. The maximum value for *axis-length* that allows a plot to fit on one page is eight lines less than the value of the SAS system option PAGESIZE= because you must allow room for the procedure to print information under the horizontal axis. The exact maximum depends on the titles that are used, whether plots are overlaid, and whether CONTOUR is specified. If the value of *axis-length* specifies a plot that cannot fit on one page, then the plot spans multiple pages.

See: "HPOS=axis-length " on page 911

VREF=*value-specification*
draws lines on the plot perpendicular to the specified values on the vertical axis. PROC PLOT includes the values that you specify with the VREF= option on the vertical axis unless you specify otherwise with the VAXIS= option. For the syntax for *value-specification*, see "HAXIS=axis-specification " on page 909.

Example: "Example 2: Controlling the Horizontal Axis and Adding a Reference Line" on page 922

VREFCHAR='*character*'
specifies the character to use to draw the vertical reference lines.

Default: horizontal bar (-)

See: "FORMCHAR <(position(s))>='formatting-character(s)' " on page 903, "HREFCHAR='character' " on page 912, and "VREF=value-specification " on page 917

VREVERSE
reverses the order of the values on the vertical axis.

VSPACE=*n*
specifies that a tick mark will occur on the vertical axis at every *n*th print position, where *n* is the value of VSPACE=.

VZERO
assigns a value of zero to the first tick mark on the vertical axis.

Interaction: PROC PLOT ignores the VZERO option if the vertical variable has negative values or if the VAXIS= option specifies a range that does not begin with zero.

Results: PLOT Procedure

Scale of the Axes

Normally, PROC PLOT looks at the minimum difference between each pair of the five lowest ordered values of each variable (the *delta*) and ensures that there is no more than one of these intervals per print position on the final scaled axis, if possible. If there is not enough room for this interval arrangement, and if PROC PLOT guesses that the data was artificially generated, then it puts a fixed number of deltas in each print position. Otherwise, PROC PLOT ignores the value.

Printed Output

Each plot uses one full page unless the plot's size is changed by the VPOS= and HPOS= options in the PLOT statement, the VPERCENT= or HPERCENT= options in the PROC PLOT statement, or the PAGESIZE= and LINESIZE= system options. Titles, legends, and variable labels are printed at the top of each page. Each axis is labeled with the variable's name or, if it exists, the variable's label.

Normally, PROC PLOT begins a new plot on a new page. However, the VPERCENT= and HPERCENT= options enable you to print more than one plot on a page. VPERCENT= and HPERCENT= are described earlier in "PROC PLOT Statement" on page 902.

PROC PLOT always begins a new page after a RUN statement and at the beginning of a BY group.

ODS Table Names

The PLOT 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 Appendix 1, "ODS Table Names and the Base SAS Procedures That Produce Them," in *SAS Output Delivery System: User's Guide*.

Table 35.7 ODS Tables Produced by the PLOT Procedure

Table Name	Description	Conditions When Table Is Generated
Plot	A single plot	When you do not specify the OVERLAY option.
Overlaid	Two or more plots on a single set of axes	When you specify the OVERLAY option.

Portability of ODS Output with PROC PLOT

Under certain circumstances, using PROC PLOT with the Output Delivery System produces files that are not portable. If the SAS system option FORMCHAR= in your SAS session uses nonstandard line-drawing characters, then the output might include strange characters instead of lines in operating environments in which the SAS Monospace font is not installed. To avoid this problem, specify the following OPTIONS statement before executing PROC PLOT:

```
options formchar="|----|+|---+=|-/\<>*";
```

Missing Values

If values of either of the plotting variables are missing, then PROC PLOT does not include the observation in the plot. However, in a plot of Y*X, values of X with corresponding missing values of Y are included in scaling the X axis, unless the NOMISS option is specified in the PROC PLOT statement.

Hidden Observations

By default, PROC PLOT uses different plotting symbols (A, B, C, and so on) to represent observations whose values coincide on a plot. However, if you specify your own plotting symbol or if you use the OVERLAY option, then you might not be able to recognize coinciding values.

If you specify a plotting symbol, then PROC PLOT uses the same symbol regardless of the number of observations whose values coincide. If you use the OVERLAY option and overprinting is not in effect, then PROC PLOT uses the symbol from the first plot request. In both cases, the output includes a message telling you how many observations are hidden.

Examples: PLOT Procedure

Example 1: Specifying a Plotting Symbol

Features: PLOT statement
plotting symbol in plot request

Data set: DJIA

This example expands on Output 35.1 on page 894 by specifying a different plotting symbol.

Program

```
options formchar="|----|+|---+=|-/\<>*";

data djia;
      input Year HighDate date7. High LowDate date7. Low;
      format highdate lowdate date7.;
      datalines;
1968 03DEC68   985.21 21MAR68   825.13
1969 14MAY69   968.85 17DEC69   769.93
...more data lines...
2006 27DEC06 12510.57 20JAN06 10667.39
2007 09OCT07 14164.53 05MAR07 12050.41
2008 02MAY08 13058.20 10OCT08  8451.19
;

proc plot data=djia;
      plot high*year='*'
      / vspace=5 vaxis=by 1000;

   title 'High Values of the Dow Jones Industrial Average';
   title2 'from 1968 to 2008';
run;
```

Program Description

Set the FORMCHAR option. Setting FORMCHAR to this exact string renders better HTML output when it is viewed outside of the SAS environment where SAS Monospace fonts are not available.

```
options formchar="|----|+|---+=|-/\<>*";
```

Create the DJIA data set. DJIA contains the high and low closing marks for the Dow Jones Industrial Average from 1968 to 2008 . The DATA step creates this data set.

```
data djia;
      input Year HighDate date7. High LowDate date7. Low;
      format highdate lowdate date7.;
      datalines;
1968 03DEC68   985.21 21MAR68   825.13
1969 14MAY69   968.85 17DEC69   769.93
...more data lines...
2006 27DEC06 12510.57 20JAN06 10667.39
2007 09OCT07 14164.53 05MAR07 12050.41
2008 02MAY08 13058.20 10OCT08  8451.19
;
```

Create the plot. The plot request plots the values of High on the vertical axis and the values of Year on the horizontal axis. It also specifies an asterisk as the plotting symbol.

```
proc plot data=djia;
      plot high*year='*'
      / vspace=5 vaxis=by 1000;
```

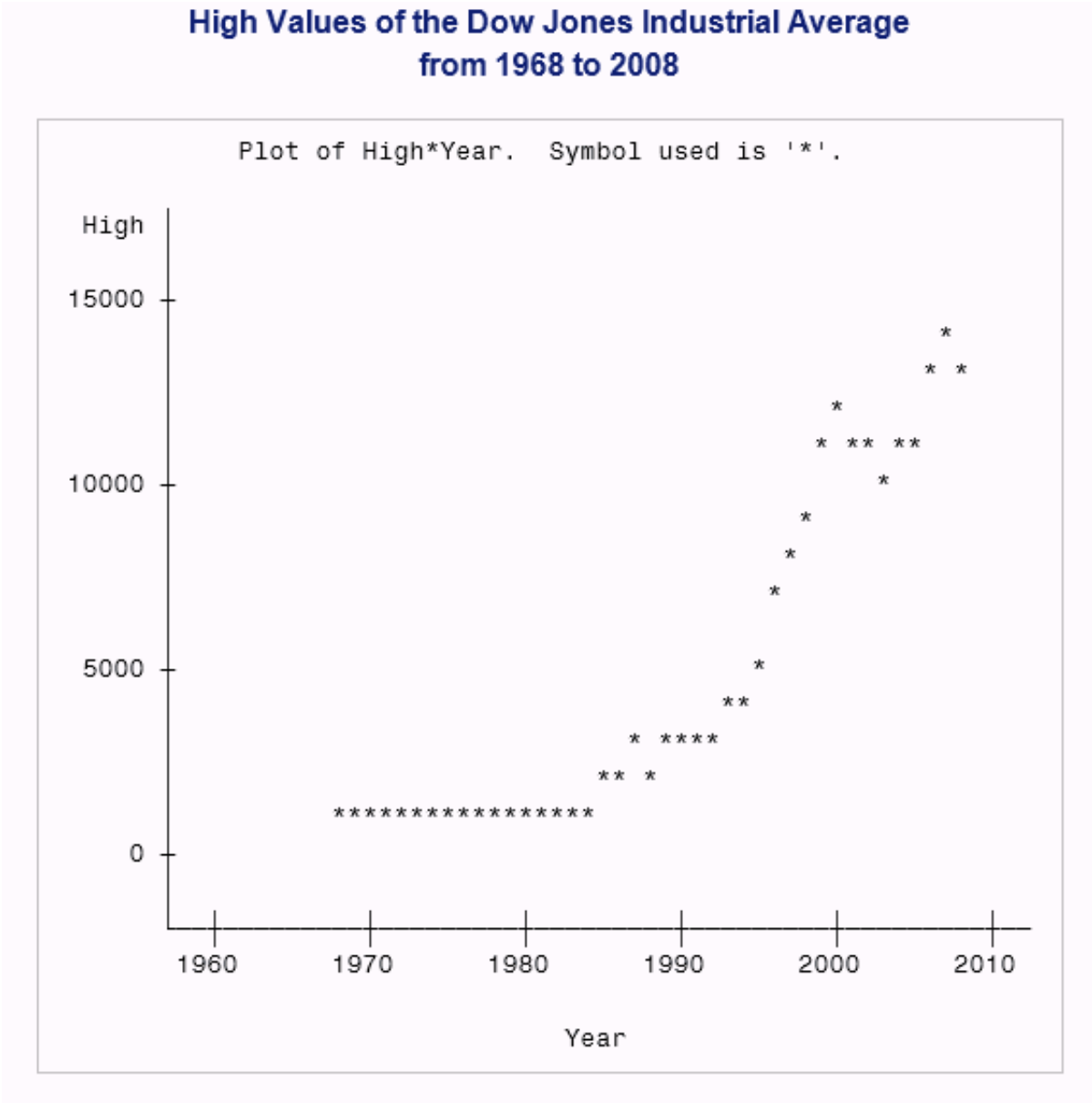
Specify the titles.

```
   title 'High Values of the Dow Jones Industrial Average';
   title2 'from 1968 to 2008';
run;
```

Output

PROC PLOT determines the tick marks and the scale of both axes.

Output 35.4 *Plot with Asterisk as the Plotting Symbol*



Example 2: Controlling the Horizontal Axis and Adding a Reference Line

Features: PLOT statement options:
HAXIS=
VREF=

Data set: DJIA

This example specifies values for the horizontal axis and draws a reference line from the vertical axis.

Program

```
options formchar="|----|+|---+=|-/\<>*";

data djia;
      input Year HighDate date7. High LowDate date7. Low;
      format highdate lowdate date7.;
      datalines;
1968 03DEC68   985.21 21MAR68   825.13
1969 14MAY69   968.85 17DEC69   769.93
...more data lines...
2006 27DEC06 12510.57 20JAN06 10667.39
2007 09OCT07 14164.53 05MAR07 12050.41
2008 02MAY08 13058.20 10OCT08  8451.19
;

proc plot data=djia;
   plot high*year='*'

     / haxis=1965 to 2020 by 10 vref=3000;


title 'High Values of Dow Jones Industrial Average';
title2 'from 1968 to 2008';
run;
```

Program Description

Set the FORMCHAR option. Setting FORMCHAR to this exact string renders better HTML output when it is viewed outside of the SAS environment where SAS Monospace fonts are not available.

```
options formchar="|----|+|---+=|-/\<>*";
```

Create the DJIA data set. DJIA contains the high and low closing marks for the Dow Jones Industrial Average from 1968 to 2008. The DATA step creates this data set.

```
data djia;
      input Year HighDate date7. High LowDate date7. Low;
      format highdate lowdate date7.;
      datalines;
1968 03DEC68   985.21 21MAR68   825.13
1969 14MAY69   968.85 17DEC69   769.93
```

```
...more data lines...
2006 27DEC06 12510.57 20JAN06 10667.39
2007 09OCT07 14164.53 05MAR07 12050.41
2008 02MAY08 13058.20 10OCT08  8451.19
;
```

Create the plot. The plot request plots the values of High on the vertical axis and the values of Year on the horizontal axis. It also specifies an asterisk as the plotting symbol.

```
proc plot data=djia;
   plot high*year='*'
```

Customize the horizontal axis and draw a reference line. HAXIS= specifies that the horizontal axis will show the values 1968 to 2008 in ten-year increments. VREF= draws a reference line that extends from the value 3000 on the vertical axis.

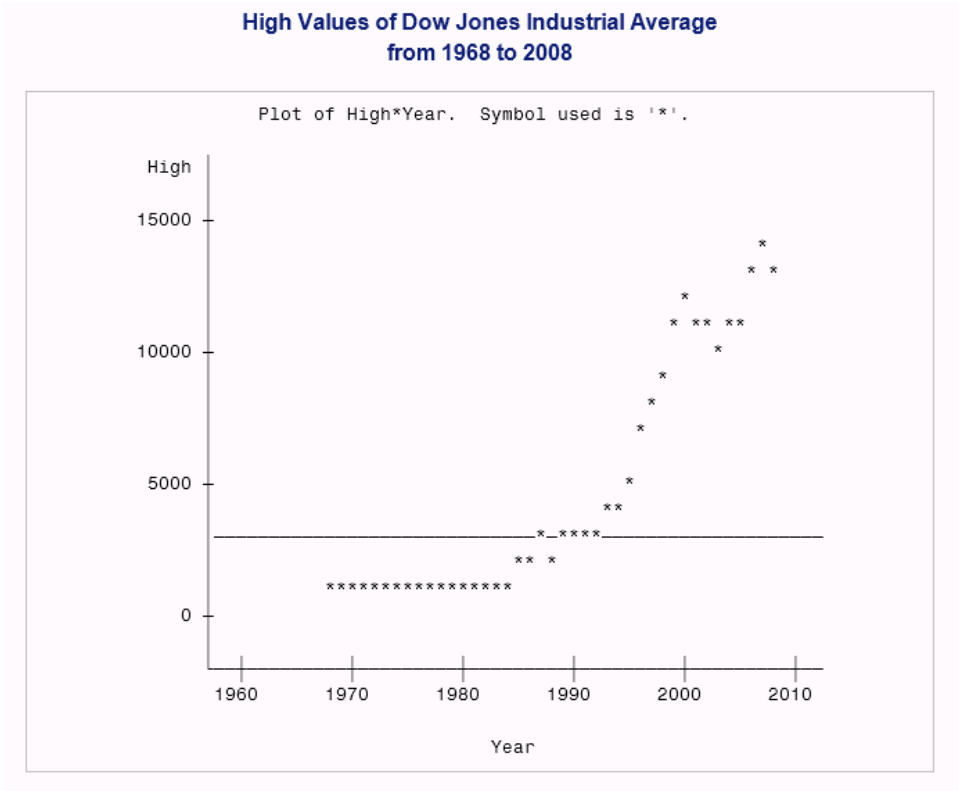
```
       / haxis=1965 to 2020 by 10 vref=3000;
```

Specify the titles.

```
title 'High Values of Dow Jones Industrial Average';
title2 'from 1968 to 2008';
run;
```

Output

Output 35.5 *Plot with Reference Line*



Example 3: Overlaying Two Plots

Features: PLOT statement options:
- BOX
- HAXIS
- OVERLAY
- VAXIS

Data set: DJIA

This example overlays two plots and puts a box around the plot.

Program

```
options formchar="|----|+|---+=|-/\<>*";

data djia;
      input Year HighDate date7. High LowDate date7. Low;
      format highdate lowdate date7.;
      datalines;
1968 03DEC68   985.21 21MAR68   825.13
1969 14MAY69   968.85 17DEC69   769.93
...more data lines...
2006 27DEC06 12510.57 20JAN06 10667.39
2007 09OCT07 14164.53 05MAR07 12050.41
2008 02MAY08 13058.20 10OCT08  8451.19
;

proc plot data=djia;
   plot high*year='*'
        low*year='o' / overlay box
        haxis=by 10
        vaxis=by 5000;

   title 'Plot of Highs and Lows';
   title2 'for the Dow Jones Industrial Average';
run;
```

Program Description

Set the FORMCHAR option. Setting FORMCHAR to this exact string renders better HTML output when it is viewed outside of the SAS environment where SAS Monospace fonts are not available.

```
options formchar="|----|+|---+=|-/\<>*";
```

Create the DJIA data set. DJIA contains the high and low closing marks for the Dow Jones Industrial Average from 1968 to 2008. The DATA step creates this data set.

```
data djia;
      input Year HighDate date7. High LowDate date7. Low;
      format highdate lowdate date7.;
      datalines;
1968 03DEC68   985.21 21MAR68   825.13
1969 14MAY69   968.85 17DEC69   769.93
```

```
...more data lines...
2006 27DEC06 12510.57 20JAN06 10667.39
2007 09OCT07 14164.53 05MAR07 12050.41
2008 02MAY08 13058.20 10OCT08  8451.19
;
```

Create the plot. The first plot request plots High on the vertical axis, plots Year on the horizontal axis, and specifies an asterisk as a plotting symbol. The second plot request plots Low on the vertical axis, plots Year on the horizontal axis, and specifies an '**o** ' as a plotting symbol. OVERLAY superimposes the second plot onto the first. BOX draws a box around the plot. OVERLAY and BOX apply to both plot requests. HAXIS= specifies that the horizontal axis will show the values 1968 to 2008 in ten-year increments. VAXIS= specifies that the vertical axis will show the values in increments of 5,000.

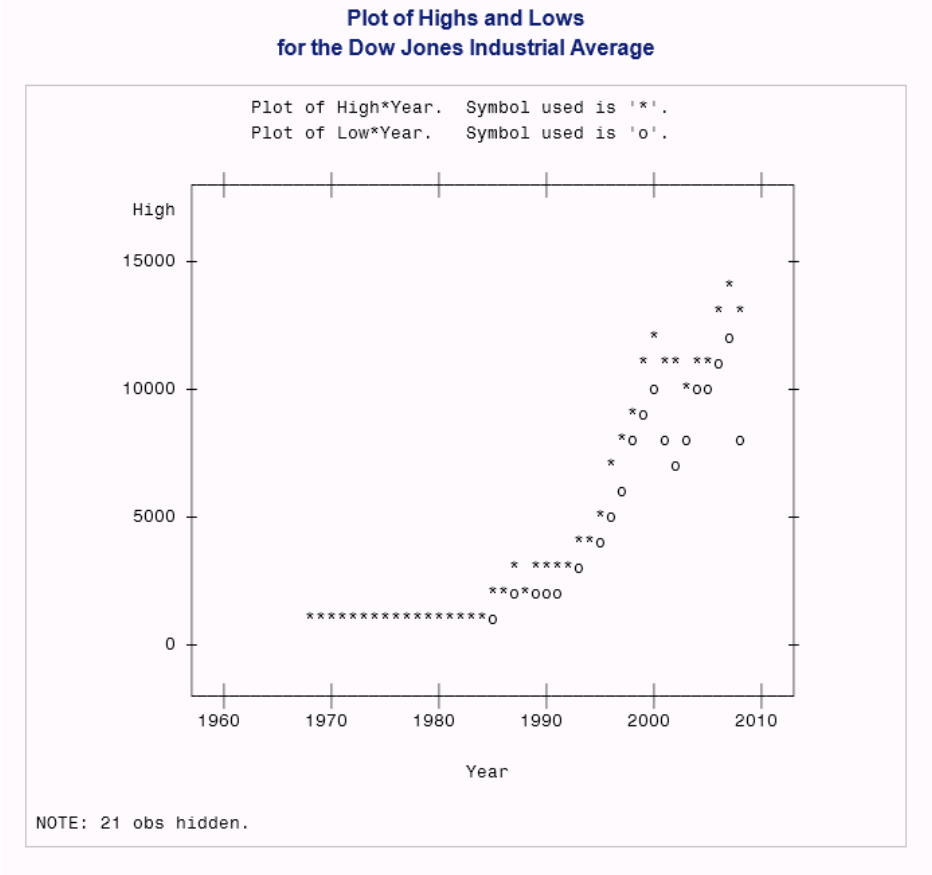
```
proc plot data=djia;
   plot high*year='*'
        low*year='o' / overlay box
        haxis=by 10
        vaxis=by 5000;
```

Specify the titles.

```
   title 'Plot of Highs and Lows';
   title2 'for the Dow Jones Industrial Average';
run;
```

Output

Output 35.6 *Two Plots Overlaid Using Different Plotting Symbols*



Example 4: Producing Multiple Plots per Page

Features: PROC PLOT statement options:
HPERCENT=
VPERCENT=

Data set: DJIA

This example puts three plots on one page of output.

Program

```
options formchar="|----|+|---+=|-/\<>*" pagesize=40 linesize=120;

data djia;
      input Year HighDate date7. High LowDate date7. Low;
      format highdate lowdate date7.;
      datalines;
1968 03DEC68   985.21 21MAR68   825.13
1969 14MAY69   968.85 17DEC69   769.93
...more data lines...
2006 27DEC06 12510.57 20JAN06 10667.39
```

```
2007 09OCT07 14164.53 05MAR07 12050.41
2008 02MAY08 13058.20 10OCT08  8451.19
;

proc plot data=djia vpercent=50 hpercent=50;

   plot high*year='*';

   plot low*year='o';

   plot high*year='*' low*year='o' / overlay box;

   title 'Plots of the Dow Jones Industrial Average';
   title2 'from 1968 to 2008';
run;
```

Program Description

Set the FORMCHAR option. Setting FORMCHAR to this exact string renders better HTML output when it is viewed outside of the SAS environment where SAS Monospace fonts are not available. The PAGESIZE= option sets the number of lines of output to 40, and the LINESIZE= option sets the line size in the output window to 120 characters.

```
options formchar="|----|+|---+=|-/\<>*" pagesize=40 linesize=120;
```

Create the DJIA data set. DJIA contains the high and low closing marks for the Dow Jones Industrial Average from 1968 to 2008. The DATA step creates this data set.

```
data djia;
      input Year HighDate date7. High LowDate date7. Low;
      format highdate lowdate date7.;
      datalines;
1968 03DEC68   985.21 21MAR68   825.13
1969 14MAY69   968.85 17DEC69   769.93
...more data lines...
2006 27DEC06 12510.57 20JAN06 10667.39
2007 09OCT07 14164.53 05MAR07 12050.41
2008 02MAY08 13058.20 10OCT08  8451.19
;
```

Specify the plot sizes. VPERCENT= specifies that 50% of the vertical space on the page of output is used for each plot. HPERCENT= specifies that 50% of the horizontal space is used for each plot.

```
proc plot data=djia vpercent=50 hpercent=50;
```

Create the first plot. This plot request plots the values of High on the vertical axis and the values of Year on the horizontal axis. It also specifies an asterisk as the plotting symbol.

```
   plot high*year='*';
```

Create the second plot. This plot request plots the values of Low on the vertical axis and the values of Year on the horizontal axis. It also specifies an asterisk as the plotting symbol.

```
   plot low*year='o';
```

Create the third plot. The first plot request plots High on the vertical axis, plots Year on the horizontal axis, and specifies an asterisk as a plotting symbol. The second plot

request plots Low on the vertical axis, plots Year on the horizontal axis, and specifies an 'o' as a plotting symbol. OVERLAY superimposes the second plot onto the first. BOX draws a box around the plot. OVERLAY and BOX apply to both plot requests.

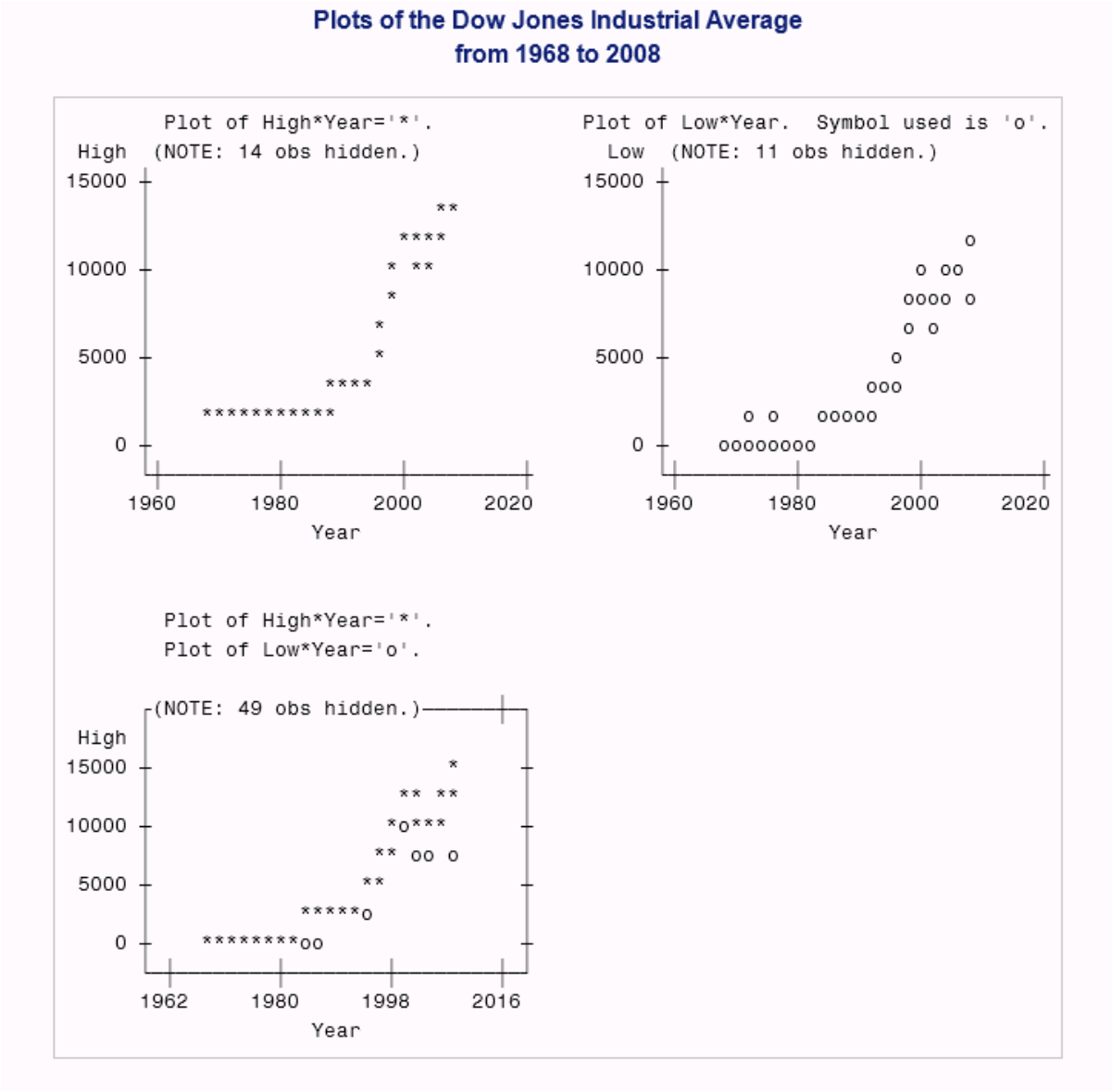
```
plot high*year='*' low*year='o' / overlay box;
```

Specify the titles.

```
title 'Plots of the Dow Jones Industrial Average';
title2 'from 1968 to 2008';
run;
```

Output

Output 35.7 *Three Plots on One Page*



Example 5: Plotting Data on a Logarithmic Scale

Features: PLOT statement option:
HAXIS=

This example uses a DATA step to generate data. The PROC PLOT step shows two plots of the same data: one plot without a horizontal axis specification and one plot with a logarithmic scale specified for the horizontal axis.

Program

```
data equa;
   do Y=1 to 3 by .1;
      X=10**y;
      output;
   end;
run;

proc plot data=equa hpercent=50;

    plot y*x / vspace=1;
    plot y*x / haxis=10 100 1000 vspace=1;

    title 'Two Plots with Different';
    title2 'Horizontal Axis Specifications';
run;
```

Program Description

Create the EQUA data set. EQUA contains values of X and Y. Each value of X is calculated as 10^{Y}.

```
data equa;
   do Y=1 to 3 by .1;
      X=10**y;
      output;
   end;
run;
```

Specify the plot sizes. HPERCENT= makes room for two plots side-by-side by specifying that 50% of the horizontal space is used for each plot.

```
proc plot data=equa hpercent=50;
```

Create the plots. The plot requests plot Y on the vertical axis and X on the horizontal axis. HAXIS= specifies a logarithmic scale for the horizontal axis for the second plot.

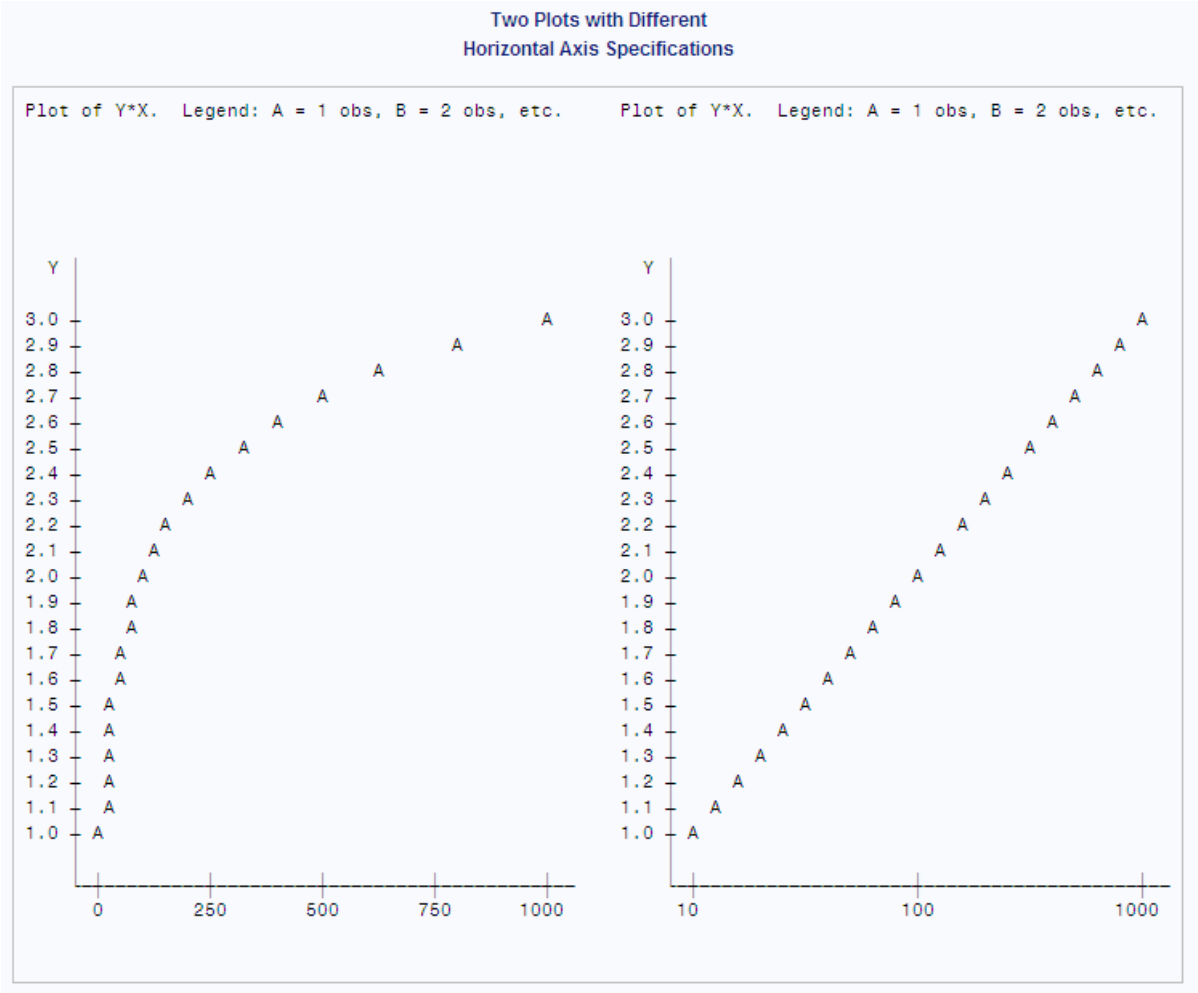
```
    plot y*x / vspace=1;
    plot y*x / haxis=10 100 1000 vspace=1;
```

Specify the titles.

```
    title 'Two Plots with Different';
    title2 'Horizontal Axis Specifications';
run;
```

Output

Output 35.8 *Two Plots with Different Horizontal Axis Specifications*



Example 6: Plotting Date Values on an Axis

Features: PLOT statement option:
HAXIS=

This example shows how you can specify date values on an axis.

Program

```
options formchar="|----|+|---+=|-/\<>*";

data emergency_calls;
   input Date : date7. Calls @@;
   label calls='Number of Calls';
   datalines;
1APR94 134   11APR94 384  13FEB94 488
2MAR94 289   21MAR94 201  14MAR94 460
3JUN94 184   13JUN94 152  30APR94 356
```

```
4JAN94 179   14JAN94 128  16JUN94 480
5APR94 360   15APR94 350  24JUL94 388
6MAY94 245   15DEC94 150  17NOV94 328
7JUL94 280   16MAY94 240  25AUG94 280
8AUG94 494   17JUL94 499  26SEP94 394
9SEP94 309   18AUG94 248  23NOV94 590
19SEP94 356  24FEB94 201  29JUL94 330
10OCT94 222  25MAR94 183  30AUG94 321
11NOV94 294  26APR94 412   2DEC94 511
27MAY94 294  22DEC94 413  28JUN94 309
;

proc plot data=emergency_calls;
   plot calls*date / haxis='1JAN94'd to '1JAN95'd by month vaxis=by 100 vspace=5;

   format date mmyyd5.;

   title 'Calls to City Emergency Services Number';
   title2 'Sample of Days for 1994';
run;
```

Program Description

Set the FORMCHAR option. Setting FORMCHAR to this exact string renders better HTML output when it is viewed outside of the SAS environment where SAS Monospace fonts are not available.

```
options formchar="|----|+|---+=|-/\<>*";
```

Create the EMERGENCY_CALLS data set. EMERGENCY_CALLS contains the number of telephone calls to an emergency help line for each date.

```
data emergency_calls;
   input Date : date7. Calls @@;
   label calls='Number of Calls';
   datalines;
1APR94 134   11APR94 384  13FEB94 488
2MAR94 289   21MAR94 201  14MAR94 460
3JUN94 184   13JUN94 152  30APR94 356
4JAN94 179   14JAN94 128  16JUN94 480
5APR94 360   15APR94 350  24JUL94 388
6MAY94 245   15DEC94 150  17NOV94 328
7JUL94 280   16MAY94 240  25AUG94 280
8AUG94 494   17JUL94 499  26SEP94 394
9SEP94 309   18AUG94 248  23NOV94 590
19SEP94 356  24FEB94 201  29JUL94 330
10OCT94 222  25MAR94 183  30AUG94 321
11NOV94 294  26APR94 412   2DEC94 511
27MAY94 294  22DEC94 413  28JUN94 309
;
```

Create the plot. The plot request plots Calls on the vertical axis and Date on the horizontal axis. HAXIS= uses a monthly time for the horizontal axis. The notation **'1JAN94'd** is a date constant. The value **'1JAN95'd** ensures that the axis will have enough room for observations from December.

```
proc plot data=emergency_calls;
   plot calls*date / haxis='1JAN94'd to '1JAN95'd by month vaxis=by 100 vspace=5;
```

Format the DATE values. The FORMAT statement assigns the MMYYD5. format to Date, which uses 2-digit month and 2-digit year values separated by a hyphen.

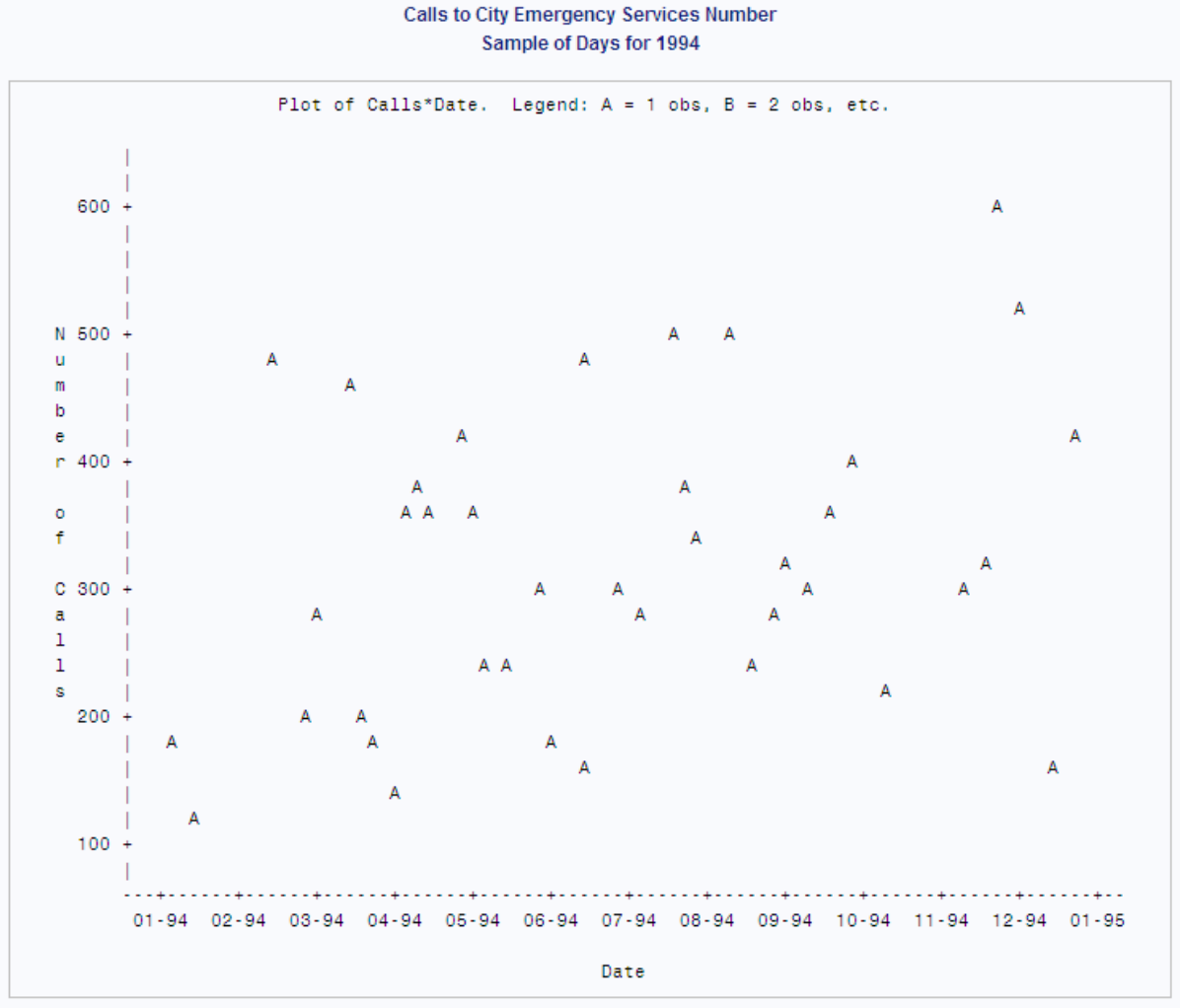
```
format date mmyyd5.;
```

Specify the titles.

```
title 'Calls to City Emergency Services Number';
title2 'Sample of Days for 1994';
run;
```

Output

Output 35.9 *Plot with Date Values along the Horizontal Axis*



Example 7: Producing a Contour Plot

Features: PLOT statement option:
CONTOUR=

This example shows how to represent the values of three variables with a two-dimensional plot by setting one of the variables as the CONTOUR variable. The variables X and Y appear on the axes, and Z is the contour variable. Program statements are used to generate the observations for the plot, and the following equation describes the contour surface: $z = 46.2 + .09x - .0005x^2 + .1y - .0005y^2 + .0004xy$

Program

```
options formchar="|----|+|---+=|-/\<>*";

data contours;
   format Z 5.1;
   do X=0 to 400 by 5;
      do Y=0 to 350 by 10;
         z=46.2+.09*x-.0005*x**2+.1*y-.0005*y**2+.0004*x*y;
         output;
      end;
   end;
run;

proc print data=contours(obs=5) noobs;
   title 'CONTOURS Data Set';
   title2 'First 5 Observations Only';
run;
```

```
                        CONTOURS Data Set                          1
                    First 5 Observations Only

                           Z    X     Y

                        46.2    0     0
                        47.2    0    10
                        48.0    0    20
                        48.8    0    30
                        49.4    0    40
```

```
options formchar="|----|+|---+=|-/\<>*";

proc plot data=contours;
plot y*x=z / contour=10;

    title 'A Contour Plot';
run;
```

Program Description

Set the FORMCHAR option. Setting FORMCHAR to this exact string renders better HTML output when it is viewed outside of the SAS environment where SAS Monospace fonts are not available.

```
options formchar="|----|+|---+=|-/\<>*";
```

Create the CONTOURS data set.

```
data contours;
   format Z 5.1;
   do X=0 to 400 by 5;
      do Y=0 to 350 by 10;
```

```
            z=46.2+.09*x-.0005*x**2+.1*y-.0005*y**2+.0004*x*y;
            output;
         end;
      end;
run;
```

Print the CONTOURS data set. The OBS= data set option limits the printing to only the first 5 observations. NOOBS suppresses printing of the observation numbers.

```
proc print data=contours(obs=5) noobs;
   title 'CONTOURS Data Set';
   title2 'First 5 Observations Only';
run;
```

CONTOURS data set. CONTOURS contains observations with values of X that range from 0 to 400 by 5 and with values of Y that range from 0 to 350 by 10.

```
                        CONTOURS Data Set                          1
                    First 5 Observations Only

                          Z    X     Y

                       46.2    0     0
                       47.2    0    10
                       48.0    0    20
                       48.8    0    30
                       49.4    0    40
```

Set the FORMCHAR option. Setting FORMCHAR to this exact string renders better HTML output when it is viewed outside of the SAS environment where SAS Monospace fonts are not available.

```
options formchar="|----|+|---+=|-/\<>*";
```

Create the plot. The plot request plots Y on the vertical axis, plots X on the horizontal axis, and specifies Z as the contour variable. CONTOUR=10 specifies that the plot will divide the values of Z into ten increments, and each increment will have a different plotting symbol.

```
proc plot data=contours;
plot y*x=z / contour=10;
```

Specify the title.

```
    title 'A Contour Plot';
run;
```

Output

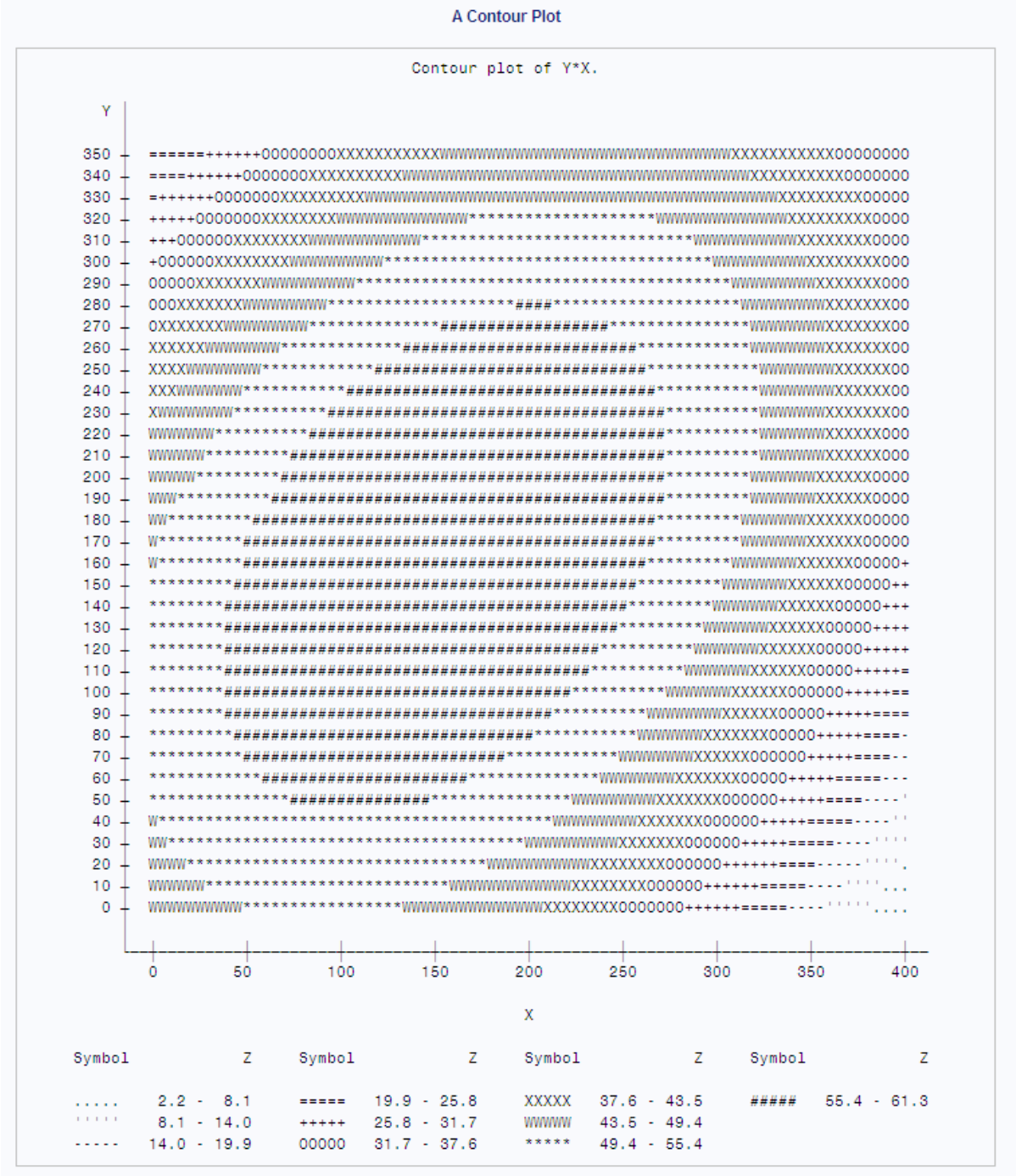
The shadings associated with the values of Z appear at the bottom of the plot. The plotting symbol # shows where high values of Z occur.

Output 35.10 *CONTOURS Data Set, First Five Observations*

CONTOURS Data Set
First 5 Observations Only

Z	X	Y
46.2	0	0
47.2	0	10
48.0	0	20
48.8	0	30
49.4	0	40

Output 35.11 *Contour Plot of Y*X*



Example 8: Plotting BY Groups

Features: PLOT statement option:
HREF=

Other features: BY statement

Data set: EDUCATION

This example shows BY-group processing in PROC PLOT.

Program

```
options formchar="|----|+|---+=|-/\<>*";

data education;
   input State $14. +1 Code $ DropoutRate Expenditures MathScore
         Region $;
   label dropout='Dropout Percentage - 1989'
          expend='Expenditure Per Pupil - 1989'
            math='8th Grade Math Exam - 1990';
   datalines;
Alabama        AL 22.3 3197 252 SE
Alaska         AK 35.8 7716 .   W
...more data lines...
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
;

proc sort data=education;
   by region;
run;

proc plot data=education;
   by region;

    plot expenditures*dropoutrate='*' / href=28.6
         vaxis=by 500 vspace=5
         haxis=by 5 hspace=12;

    title 'Plot of Dropout Rate and Expenditure Per Pupil';
 run;
```

Program Description

Set the FORMCHAR option. Setting FORMCHAR to this exact string renders better HTML output when it is viewed outside of the SAS environment where SAS Monospace fonts are not available.

```
options formchar="|----|+|---+=|-/\<>*";
```

Create the EDUCATION data set. EDUCATION contains educational data (Source: U.S. Department of Education) about some U.S. states. DropoutRate is the percentage of high school dropouts. Expenditures is the dollar amount the state spends on each pupil. MathScore is the score of eighth-grade students on a standardized math test. Not all states participated in the math test. The "EDUCATION" on page 1740 DATA step creates this data set.

```
data education;
   input State $14. +1 Code $ DropoutRate Expenditures MathScore
         Region $;
   label dropout='Dropout Percentage - 1989'
          expend='Expenditure Per Pupil - 1989'
            math='8th Grade Math Exam - 1990';
   datalines;
```

```
Alabama        AL 22.3 3197 252 SE
Alaska         AK 35.8 7716 .   W
...more data lines...
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
;
```

Sort the EDUCATION data set. PROC SORT sorts EDUCATION by Region so that Region can be used as the BY variable in PROC PLOT.

```
proc sort data=education;
   by region;
run;
```

Create a separate plot for each BY group. The BY statement creates a separate plot for each value of Region.

```
proc plot data=education;
   by region;
```

Create the plot with a reference line. The plot request plots Expenditures on the vertical axis, plots DropoutRate on the horizontal axis, and specifies an asterisk as the plotting symbol. HREF= draws a reference line that extends from 28.6 on the horizontal axis. The reference line represents the national average. VAXIS and HAXIS are used to set the tick marks along the vertical and horizontal axes.

```
    plot expenditures*dropoutrate='*' / href=28.6
         vaxis=by 500 vspace=5
         haxis=by 5 hspace=12;
```

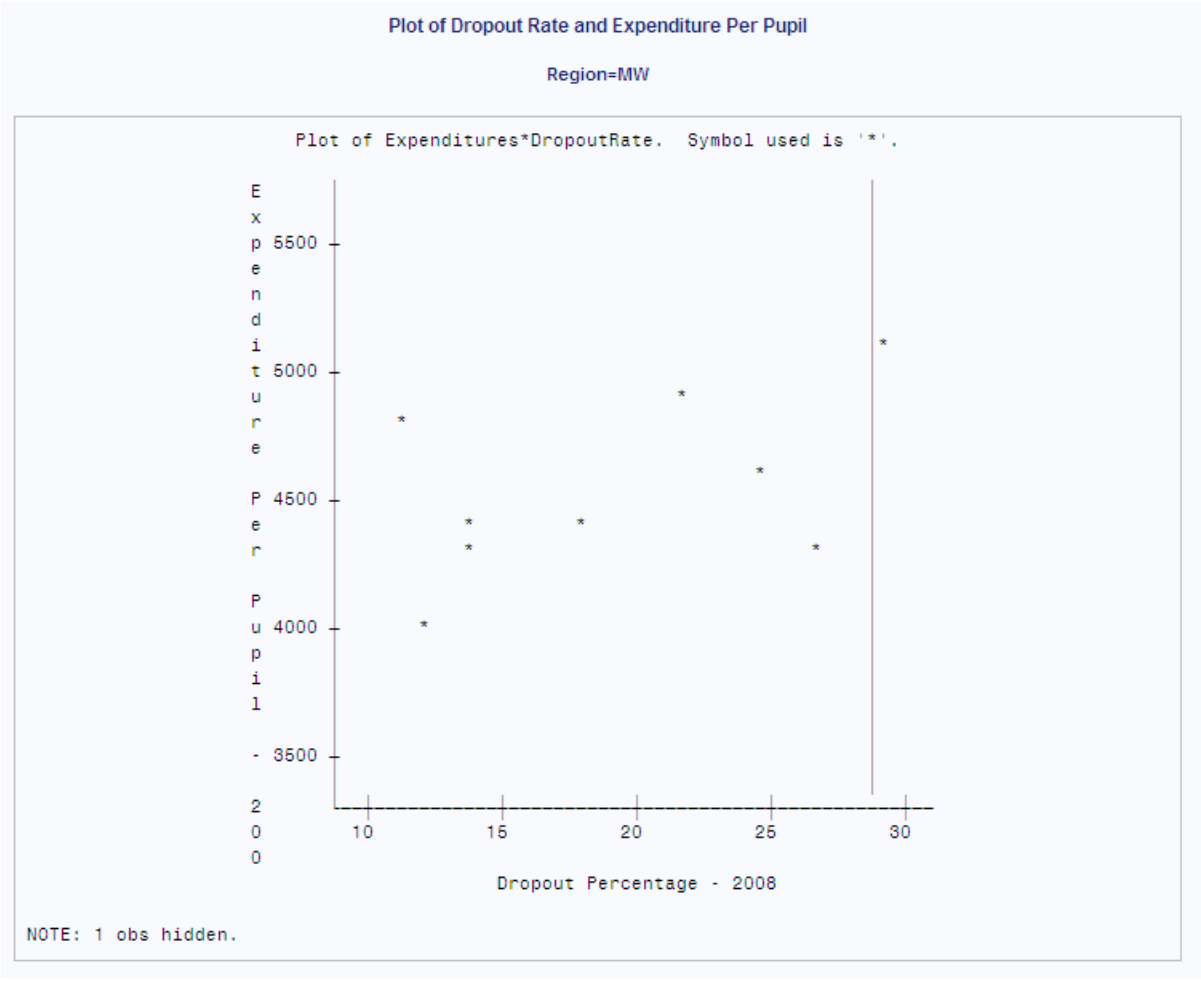
Specify the title.

```
    title 'Plot of Dropout Rate and Expenditure Per Pupil';
run;
```

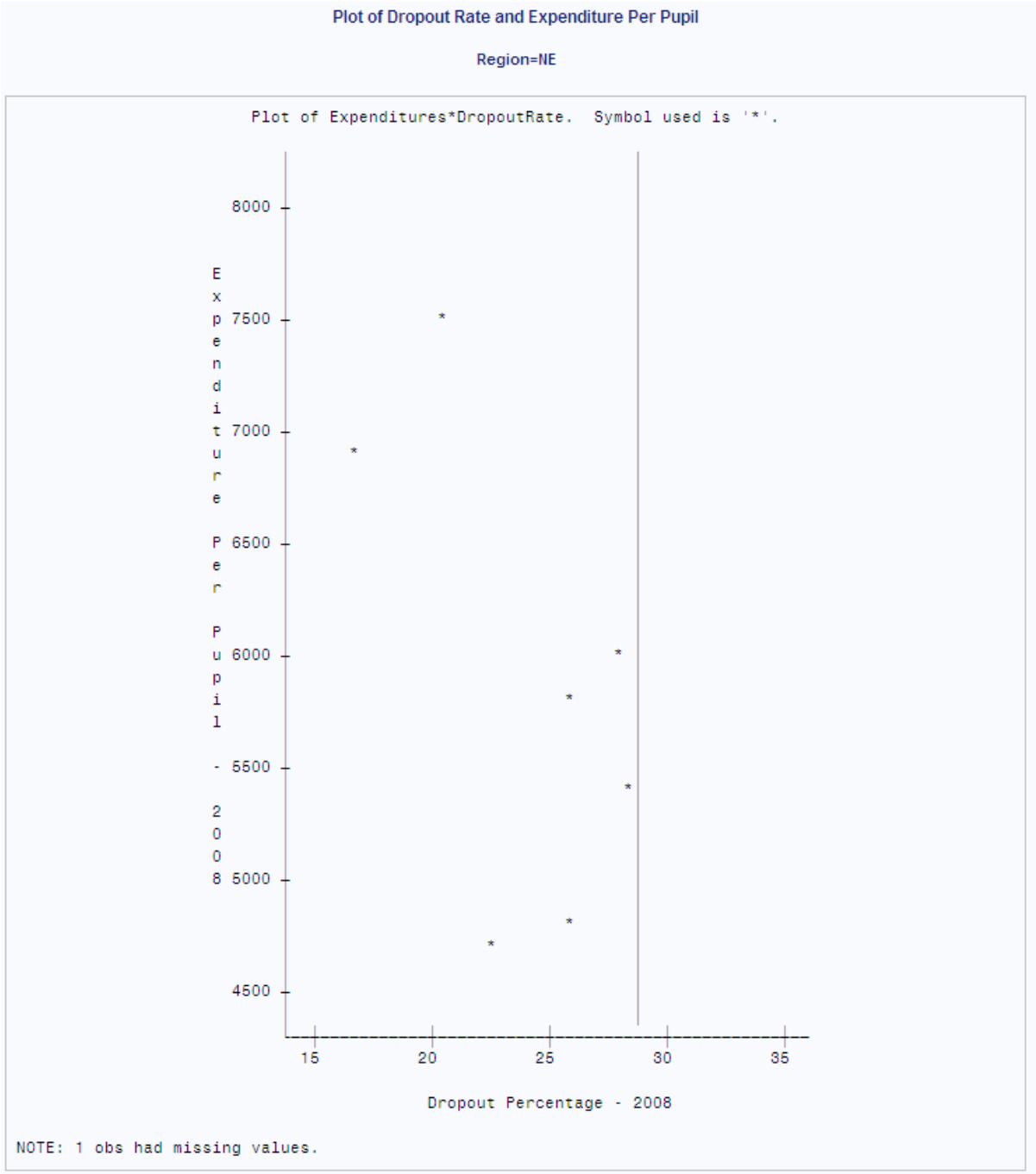
Output

PROC PLOT produces a plot for each BY group. Only the plots for Midwest and Northeast are shown.

Output 35.12 *Plot for Each BY Group, Midwest Region*



Output 35.13 *Plot for Each BY Group, Northeast Region*



Example 9: Adding Labels to a Plot

Features: PLOT statement
label variable in plot request

Data set: EDUCATION

This example shows how to modify the plot request to label points on the plot with the values of variables. This example adds labels to the plot shown in "Example 8: Plotting BY Groups" on page 936.

Program

```
options formchar="|----|+|---+=|-/\<>*";

proc sort data=education;
   by region;
run;

proc plot data=education;
   by region;

   plot expenditures*dropoutrate='*' $ state / href=28.6
        vaxis=by 500 vspace=5
        haxis=by 5 hspace=12;

   title 'Plot of Dropout Rate and Expenditure Per Pupil';
run;
```

Program Description

Set the FORMCHAR option. Setting FORMCHAR to this exact string renders better HTML output when it is viewed outside of the SAS environment where SAS Monospace fonts are not available.

```
options formchar="|----|+|---+=|-/\<>*";
```

Sort the EDUCATION data set. PROC SORT sorts EDUCATION by Region so that Region can be used as the BY variable in PROC PLOT.

```
proc sort data=education;
   by region;
run;
```

Create a separate plot for each BY group. The BY statement creates a separate plot for each value of Region.

```
proc plot data=education;
   by region;
```

Create the plot with a reference line and a label for each data point. The plot request plots Expenditures on the vertical axis, plots DropoutRate on the horizontal axis, and specifies an asterisk as the plotting symbol. The label variable specification (**`$ state`**) in the PLOT statement labels each point on the plot with the name of the corresponding state. HREF= draws a reference line that extends from 28.6 on the horizontal axis. The reference line represents the national average. VAXIS and HAXIS are used to set the tick marks along the vertical and horizontal axes.

```
   plot expenditures*dropoutrate='*' $ state / href=28.6
        vaxis=by 500 vspace=5
        haxis=by 5 hspace=12;
```

Specify the title.

```
   title 'Plot of Dropout Rate and Expenditure Per Pupil';
run;
```

Output

PROC PLOT produces a plot for each BY group. Only the plots for **`Midwest`** and **`Northeast`** are shown.

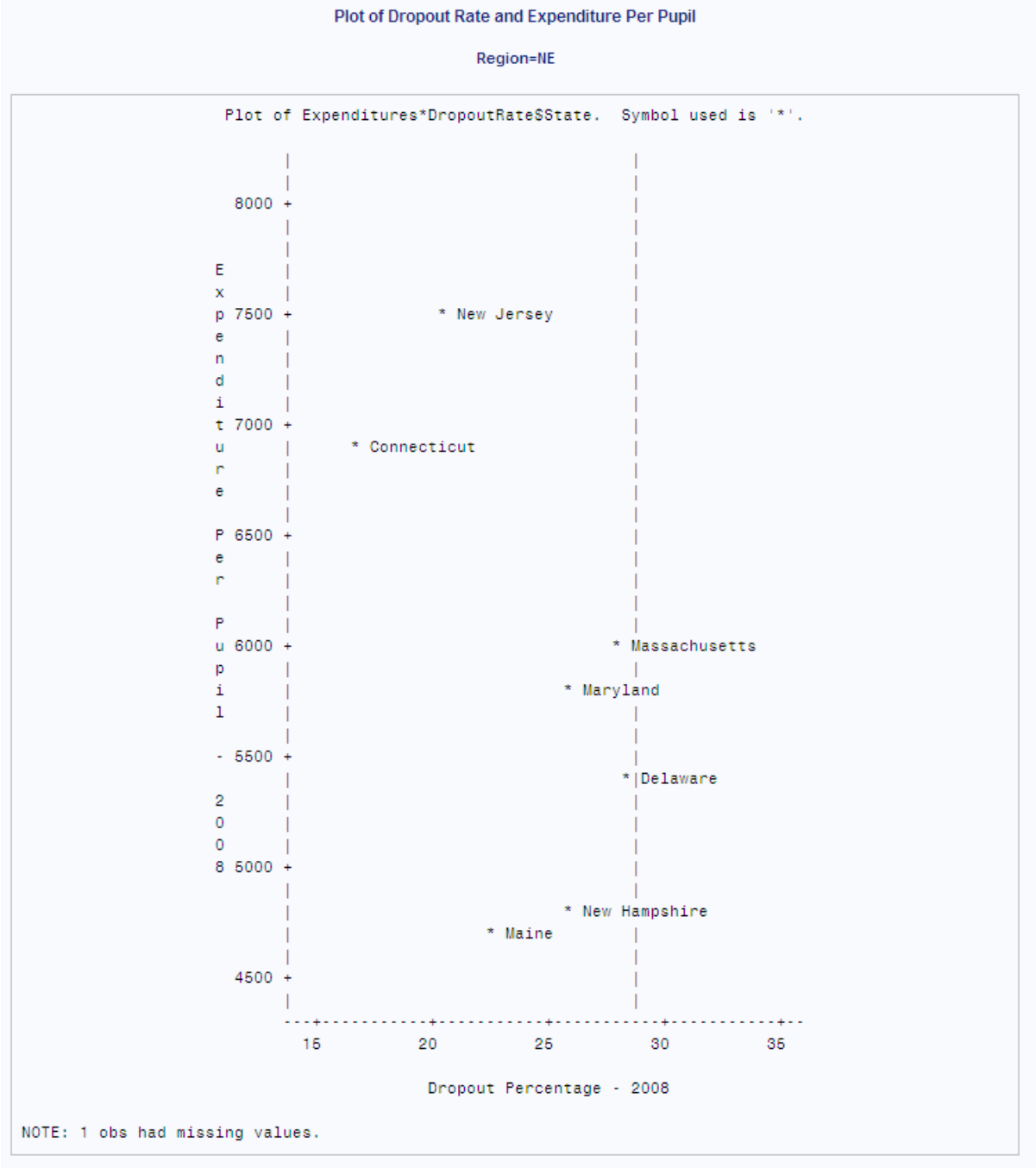
Output 35.14 *Plot with Labels, Midwest Region*

Plot of Dropout Rate and Expenditure Per Pupil

Region=MW

```
 Plot of Expenditures*DropoutRate$State.  Symbol used is '*'.

E       |                                                  |
x       |                                                  |
p 5500 +                                                  |
e       |                                                  |
n       |                                                  |
d       |                                                  |
i       |                                          Michigan|*
t 5000 +                                                  |
u       |                                 * Illinois       |
r       |     * Minnesota                                  |
e       |                                                  |
        |                                       * Ohio     |
P 4500 +                                                  |
e       |  Nebraska *          * Kansas                    |
r       |      Iowa * Indiana                        * Missouri
        |                                                  |
P       |                                                  |
u 4000 +       * North Dakota                             |
p       |                                                  |
i       |                                                  |
l       |                                                  |
        |                                                  |
- 3500 +                                                  |
        |                                                  |
2       ---+-----------+-----------+-----------+-----------+--
0         10          15          20          25          30
0
                        Dropout Percentage - 2008

NOTE: 1 obs hidden.
```

Output 35.15 *Plot with Labels, Northeast Region*



Example 10: Excluding Observations That Have Missing Values

Features: PROC PLOT statement option:
NOMISS

Data set: EDUCATION

This example shows how missing values affect the calculation of the axes.

Program

```
options formchar="|----|+|---+=|-/\<>*";

proc sort data=education;
   by region;
run;

proc plot data=education nomiss;

    by region;

    plot expenditures*dropoutrate='*' $ state / href=28.6
         vaxis=by 500 vspace=5
         haxis=by 5 hspace=12;

    title 'Plot of Dropout Rate and Expenditure Per Pupil';
run;
```

Program Description

Set the FORMCHAR option. Setting FORMCHAR to this exact string renders better HTML output when it is viewed outside of the SAS environment where SAS Monospace fonts are not available.

```
options formchar="|----|+|---+=|-/\<>*";
```

Sort the EDUCATION data set. PROC SORT sorts EDUCATION by Region so that Region can be used as the BY variable in PROC PLOT.

```
proc sort data=education;
   by region;
run;
```

Exclude data points with missing values. NOMISS excludes observations that have a missing value for either of the axis variables.

```
proc plot data=education nomiss;
```

Create a separate plot for each BY group. The BY statement creates a separate plot for each value of Region.

```
    by region;
```

Create the plot with a reference line and a label for each data point. The plot request plots Expenditures on the vertical axis, plots DropoutRate on the horizontal axis, and specifies an asterisk as the plotting symbol. The label variable specification (**`$ state`**) in the PLOT statement labels each point on the plot with the name of the corresponding state. HREF= draws a reference line extending from 28.6 on the horizontal axis. The reference line represents the national average. VAXIS and HAXIS are used to set the tick marks along the vertical and horizontal axes.

```
    plot expenditures*dropoutrate='*' $ state / href=28.6
         vaxis=by 500 vspace=5
         haxis=by 5 hspace=12;
```

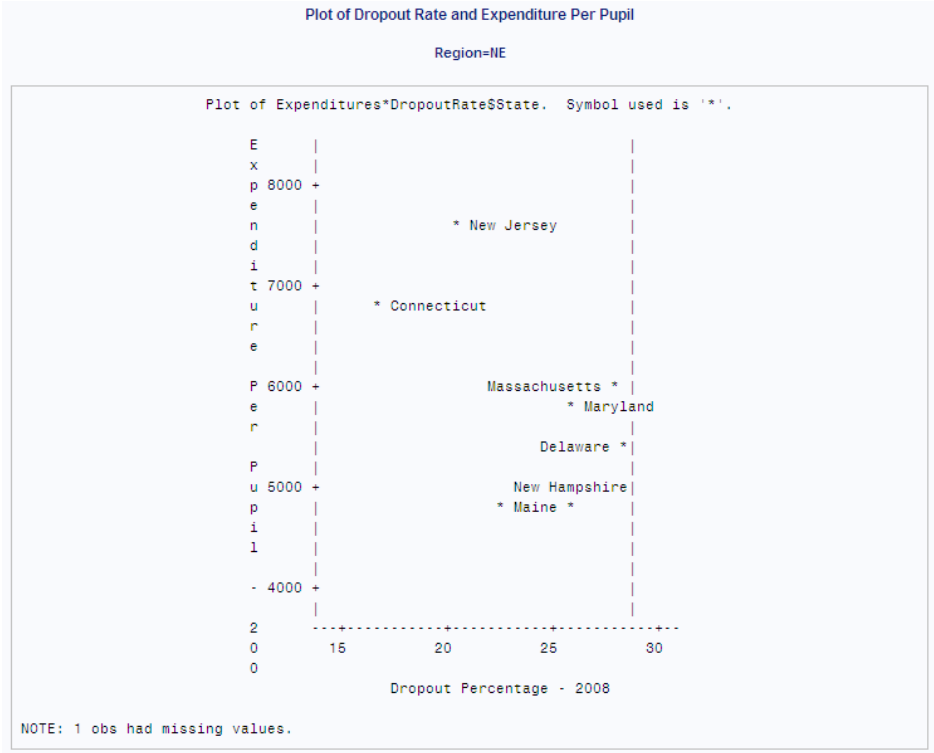
Specify the title.

```
    title 'Plot of Dropout Rate and Expenditure Per Pupil';
run;
```

Output

PROC PLOT produces a plot for each BY group. Only the plot for the **Northeast** is shown. Because **New York** has a missing value for Expenditures, the observation is excluded and PROC PLOT does not use the value 35 for DropoutRate to calculate the horizontal axis. Compare the horizontal axis in this output with the horizontal axis in the plot for **Northeast** in "Example 9: Adding Labels to a Plot" on page 940.

Output 35.16 *Plot with Missing Values Excluded*



Example 11: Adjusting Labels on a Plot with the PLACEMENT= Option

Features: PLOT statement options:
- label variable in plot request
- LIST=
- PLACEMENT=

Other features: RUN group processing

Data set: CENSUS

This example illustrates the default placement of labels and how to adjust the placement of labels on a crowded plot. The labels are values of variables in the data set.[1]

This example also shows RUN group processing in PROC PLOT.

Program

```
options formchar="|----|+|---+=|-/\<>*";
```

1 Source: U.S. Bureau of the Census and the 1987 Uniform Crime Reports, FBI.

```
data census;
   input Density CrimeRate State $ 14-27 PostalCode $ 29-30;
   datalines;
263.3 4575.3 Ohio           OH
62.1 7017.1  Washington     WA

...more data lines...

111.6 4665.6 Tennessee      TN
120.4 4649.9 North Carolina NC
;


proc plot data=census;
   plot density*crimerate=state $ state /

   box
   list=1
   haxis=by 1000
   vaxis=by 250
   vspace=10
   hspace=10;

    plot density*crimerate=state $ state /
              box
              list=1
              haxis=by 1000
              vaxis=by 250
              vspace=10

            placement=((v=2 1 : l=2 1)
                  ((l=2 2 1 : v=0 1 0) * (s=right left : h=2 -2))
                  (s=center right left * l=2 1 * v=0 1 -1 2 *
                   h=0 1 to 5 by alt));

   title 'A Plot of Population Density and Crime Rates';
run;
```

Program Description

Set the FORMCHAR option. Setting FORMCHAR to this exact string renders better HTML output when it is viewed outside of the SAS environment where SAS Monospace fonts are not available.

```
options formchar="|----|+|---+=|-/\<>*";
```

Create the CENSUS data set. CENSUS contains the variables CrimeRate and Density for selected states. CrimeRate is the number of crimes per 100,000 people. Density is the population density per square mile in the 1980 census. A DATA step, "CENSUS" on page 1710, creates this data set.

```
data census;
   input Density CrimeRate State $ 14-27 PostalCode $ 29-30;
   datalines;
263.3 4575.3 Ohio           OH
62.1 7017.1  Washington     WA

...more data lines...
```

```
111.6 4665.6 Tennessee      TN
120.4 4649.9 North Carolina NC
;
```

Create the plot with a label for each data point. The plot request plots Density on the vertical axis, CrimeRate on the horizontal axis, and uses the first letter of the value of State as the plotting symbol. This makes it easier to match the symbol with its label. The label variable specification (**`$ state`**) in the PLOT statement labels each point with the corresponding state name.

```
proc plot data=census;
   plot density*crimerate=state $ state /
```

Specify plot options. BOX draws a box around the plot. LIST= lists the labels that have penalties greater than or equal to 1. HAXIS= and VAXIS= specify increments only. PROC PLOT uses the data to determine the range for the axes.

```
   box
   list=1
   haxis=by 1000
   vaxis=by 250
   vspace=10
   hspace=10;
```

Request a second plot. Because PROC PLOT is interactive, the procedure is still running at this point in the program. It is not necessary to restart the procedure to submit another plot request. LIST=1 produces no output because there are no penalties of 1 or greater.

```
    plot density*crimerate=state $ state /
             box
             list=1
             haxis=by 1000
             vaxis=by 250
             vspace=10
```

Specify placement options. PLACEMENT= gives PROC PLOT more placement states to use to place the labels. PLACEMENT= contains three expressions. The first expression specifies the preferred positions for the label. The first expression resolves to placement states centered above the plotting symbol, with the label on one or two lines. The second and third expressions resolve to placement states that enable PROC PLOT to place the label in multiple positions around the plotting symbol.

```
          placement=((v=2 1 : l=2 1)
                ((l=2 2 1 : v=0 1 0) * (s=right left : h=2 -2))
                (s=center right left * l=2 1 * v=0 1 -1 2 *
                 h=0 1 to 5 by alt));
```

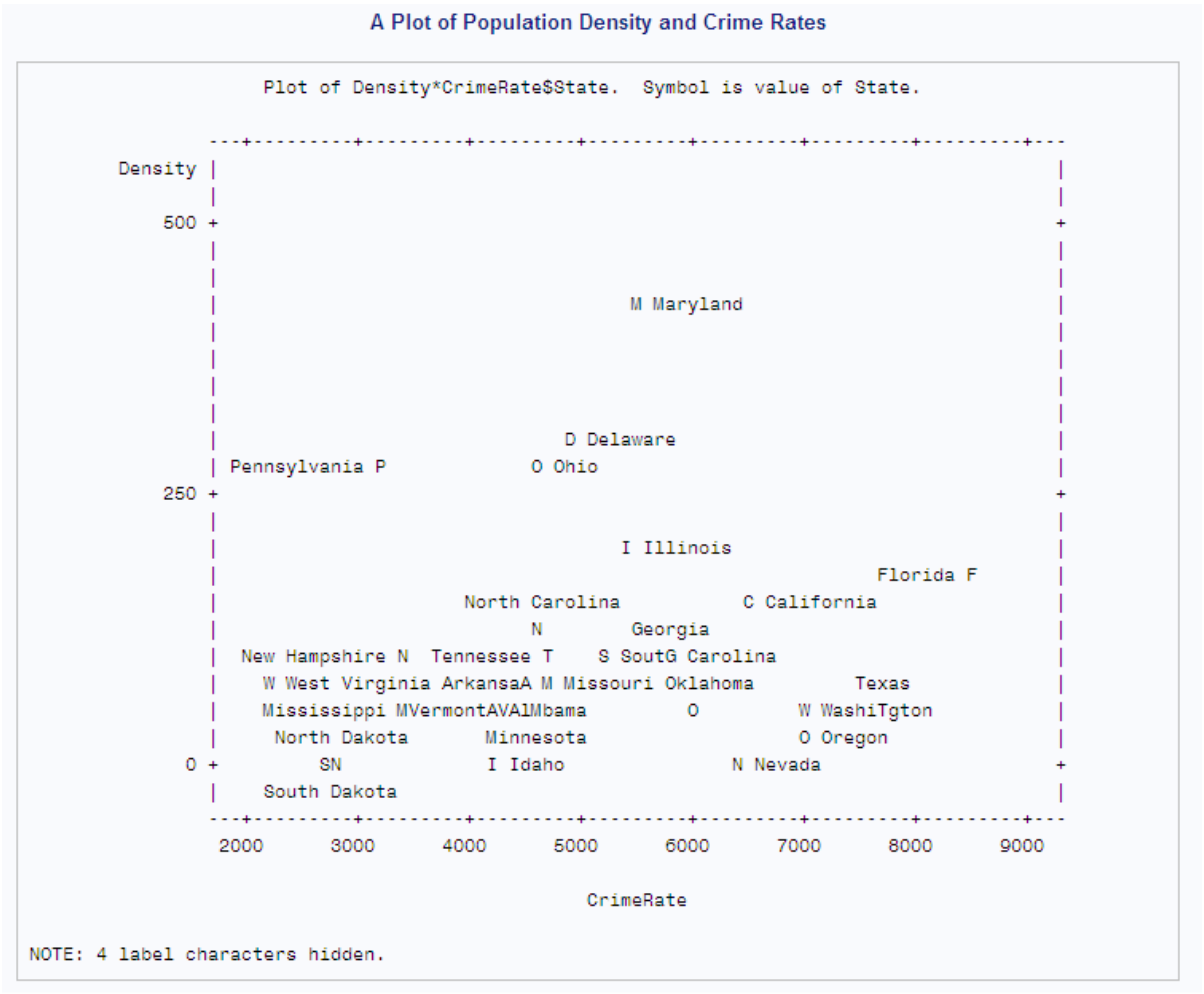
Specify the title.

```
   title 'A Plot of Population Density and Crime Rates';
run;
```

Output

The labels **Tennessee**, **South Carolina**, **Arkansas**, **Minnesota**, and **South Dakota** have penalties. The default placement states do not provide enough possibilities for PROC PLOT to avoid penalties given the proximity of the points. Four label characters are hidden.

Output 35.17 *Plot with Penalties*

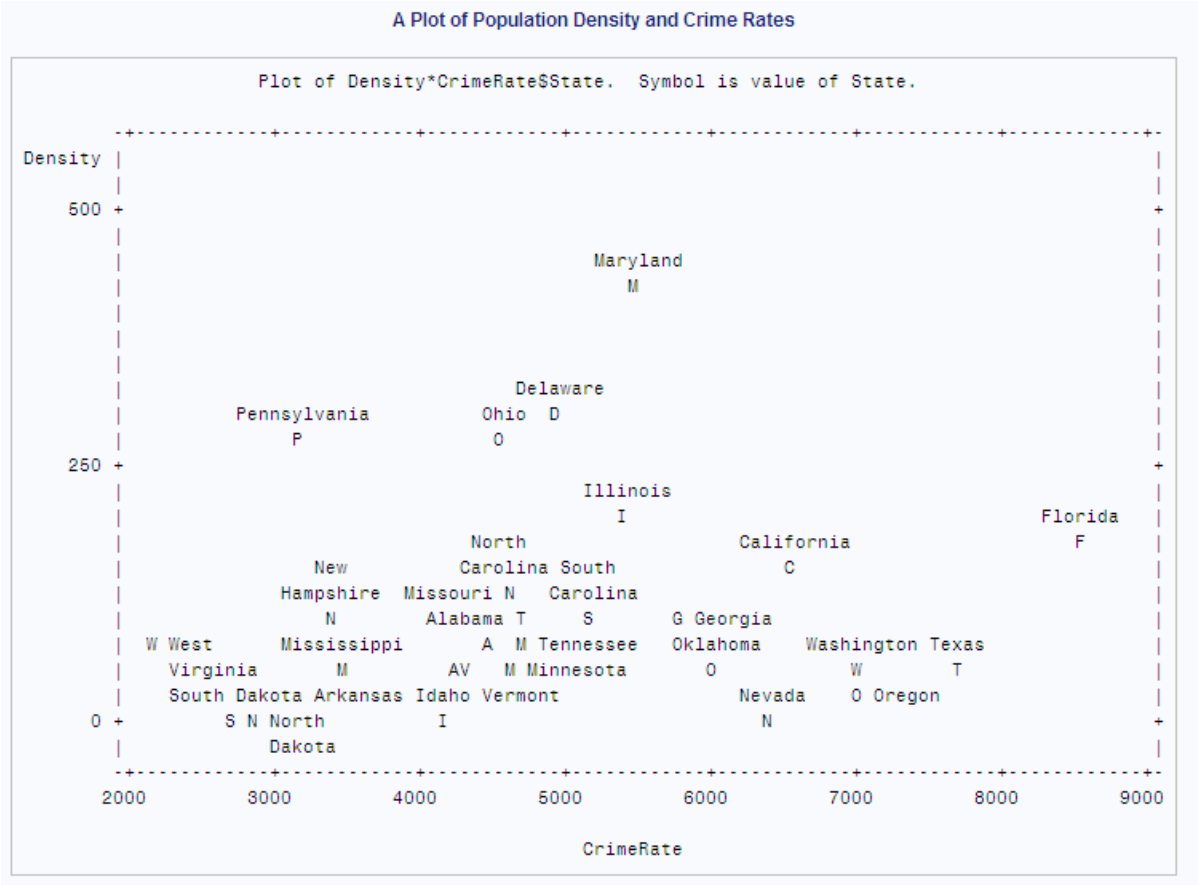


List of Point Locations, Penalties, and Placement States

Label	Vertical Axis	Horizontal Axis	Penalty	Starting Position	Lines	Vertical Shift	Horizontal Shift
South Carolina	103.40	5161.9	2	Right	1	0	2
Alabama	76.60	4451.4	9	Center	1	-1	2
Arkansas	43.90	4245.2	2	Center	1	1	-1
Vermont	55.20	4271.2	2	Left	1	0	-2
Washington	62.10	7017.1	2	Right	1	0	2

No collisions occur in the plot.

Output 35.18 *Plot with Labels Placed to Avoid Collisions*



Example 12: Adjusting Labeling on a Plot with a Macro

Features: PLOT statement options
label variable in plot request
PLACEMENT=

Data set: CENSUS

This example illustrates the default placement of labels and uses a macro to adjust the placement of labels. The labels are values of a variable in the data set.

Program

```
options formchar="|----|+|---+=|-/\<>*";

%macro place(n);
   %if &n > 13 %then %let n = 13;
       placement=(
   %if &n <= 0 %then (s=center); %else (h=2 -2 : s=right left);
   %if &n = 1 %then (v=1 * h=0 -1 to -2 by alt);
   %else %if &n = 2 %then (v=1 -1 * h=0 -1 to -5 by alt);
   %else %if &n > 2 %then (v=1 to 2 by alt * h=0 -1 to -10 by alt);
```

```
   %if &n > 3 %then
       (s=center right left * v=0 1 to %eval(&n - 2) by alt *
       h=0 -1 to %eval(-3 * (&n - 2)) by alt *
       l=1 to %eval(2 + (10 * &n - 35) / 30)); )
   %if &n > 4 %then penalty(7)=%eval((3 * &n) / 2);
%mend;

proc plot data=census;
   plot density*crimerate=state $ state /

           box
           list=1
           haxis=by 1000
           vaxis=by 250
           vspace=12
           %place(4);

   title 'A Plot of Population Density and Crime Rates';
run;
```

Program Description

Set the FORMCHAR option. Setting FORMCHAR to this exact string renders better HTML output when it is viewed outside of the SAS environment where SAS Monospace fonts are not available.

```
options formchar="|----|+|---+=|-/\<>*";
```

Use conditional logic to determine placement. The %PLACE macro provides an alternative to using the PLACEMENT= option. The higher the value of n , the more freedom PROC PLOT has to place labels.

```
%macro place(n);
   %if &n > 13 %then %let n = 13;
       placement=(
   %if &n <= 0 %then (s=center); %else (h=2 -2 : s=right left);
   %if &n = 1 %then (v=1 * h=0 -1 to -2 by alt);
   %else %if &n = 2 %then (v=1 -1 * h=0 -1 to -5 by alt);
   %else %if &n > 2 %then (v=1 to 2 by alt * h=0 -1 to -10 by alt);
   %if &n > 3 %then
       (s=center right left * v=0 1 to %eval(&n - 2) by alt *
       h=0 -1 to %eval(-3 * (&n - 2)) by alt *
       l=1 to %eval(2 + (10 * &n - 35) / 30)); )
   %if &n > 4 %then penalty(7)=%eval((3 * &n) / 2);
%mend;
```

Create the plot. The plot request plots Density on the vertical axis, CrimeRate on the horizontal axis, and uses the first letter of the value of State as the plotting symbol. The label variable specification ($ state) in the PLOT statement t labels each point with the corresponding state name.

```
proc plot data=census;
   plot density*crimerate=state $ state /
```

Specify plot options. BOX draws a box around the plot. LIST= lists the labels that have penalties greater than or equal to 1. HAXIS= and VAXIS= specify increments only. PROC PLOT uses the data to determine the range for the axes. The PLACE macro determines the placement of the labels.

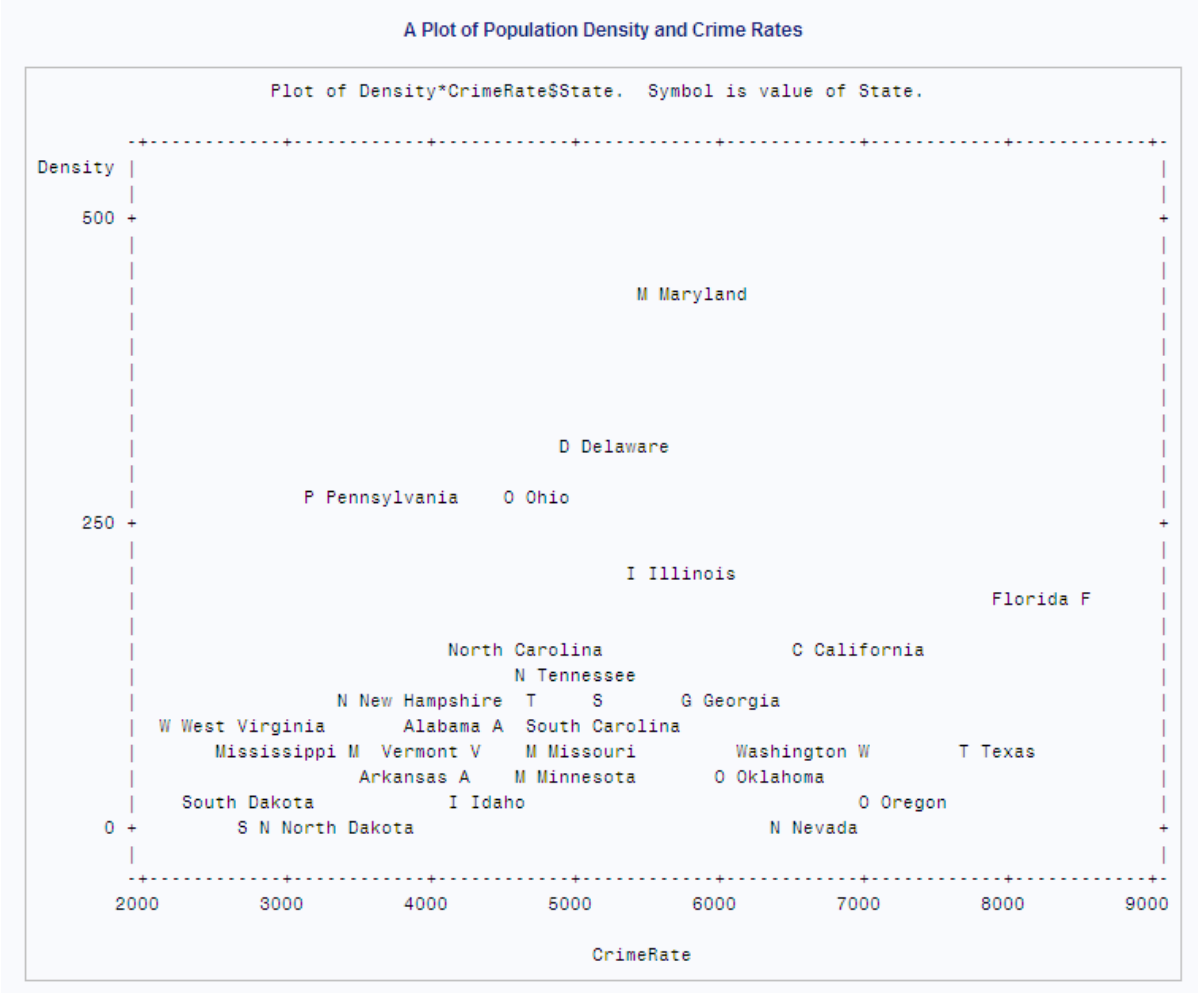
```
box
list=1
haxis=by 1000
vaxis=by 250
vspace=12
%place(4);
```

Specify the title.

```
title 'A Plot of Population Density and Crime Rates';
run;
```

Output

Output 35.19 *Plot with Labels Placed Using a Macro*



Example 13: Changing a Default Penalty

Features: PLOT statement option:
PENALTIES=

Data set: CENSUS

This example demonstrates how changing a default penalty affects the placement of labels. The goal is to produce a plot that has labels that do not detract from how the points are scattered.

Program

```
options formchar="|----|+|---+=|-/\<>*";

   proc plot data=census;
      plot density*crimerate=state $ state /

           placement=(h=100 to 10 by alt * s=left right)

           penalties(4)=500  list=0

           haxis=0 to 13000 by 1000
           vaxis=by 100
           vspace=5;

   title 'A Plot of Population Density and Crime Rates';
run;
```

Program Description

Set the FORMCHAR option. Setting FORMCHAR to this exact string renders better HTML output when it is viewed outside of the SAS environment where SAS Monospace fonts are not available.

```
options formchar="|----|+|---+=|-/\<>*";

   proc plot data=census;
      plot density*crimerate=state $ state /
```

Specify the placement. PLACEMENT= specifies that the preferred placement states are 100 columns to the left and the right of the point, on the same line with the point.

```
           placement=(h=100 to 10 by alt * s=left right)
```

Change the default penalty. PENALTIES(4)= changes the default penalty for a free horizontal shift to 500, which removes all penalties for a horizontal shift. LIST= shows how far PROC PLOT shifted the labels away from their respective points.

```
           penalties(4)=500  list=0
```

Customize the axes. HAXIS= creates a horizontal axis long enough to leave space for the labels on the sides of the plot. VAXIS= specifies that the values on the vertical axis be in increments of 100.

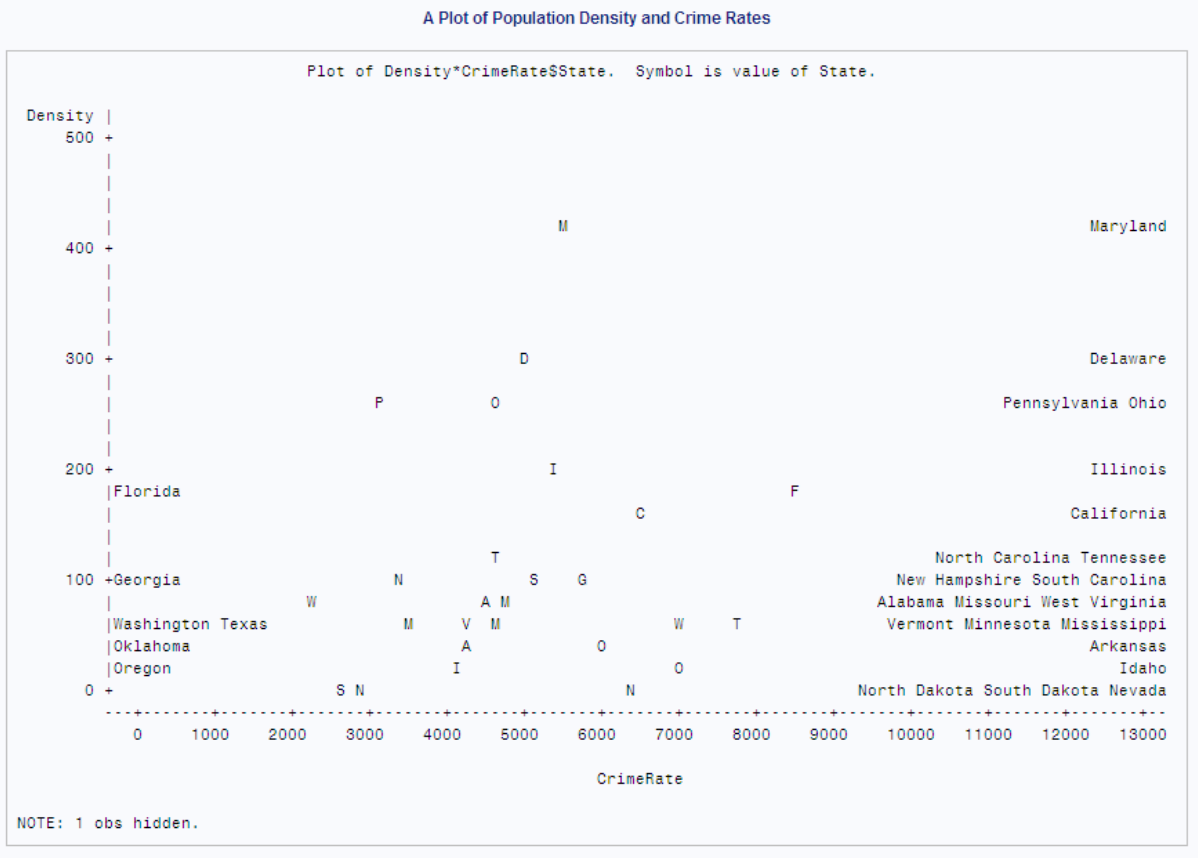
```
           haxis=0 to 13000 by 1000
           vaxis=by 100
           vspace=5;
```

Specify the title.

```
   title 'A Plot of Population Density and Crime Rates';
run;
```

Output

Output 35.20 *Plot with Default Penalties Adjusted*



Output 35.21 *List of Point Locations, Penalties, and Placement States*

List of Point Locations, Penalties, and Placement States

Label	Vertical Axis	Horizontal Axis	Penalty	Starting Position	Lines	Vertical Shift	Horizontal Shift
Maryland	428.70	5477.6	0	Right	1	0	55
Delaware	307.60	4938.8	0	Right	1	0	59
Pennsylvania	264.30	3163.2	0	Right	1	0	65
Ohio	263.30	4575.3	0	Right	1	0	66
Illinois	205.30	5416.5	0	Right	1	0	56
Florida	180.00	8503.2	0	Left	1	0	-64
California	151.40	6506.4	0	Right	1	0	45
Tennessee	111.60	4665.6	0	Right	1	0	61
North Carolina	120.40	4649.9	0	Right	1	0	46
New Hampshire	102.40	3371.7	0	Right	1	0	52
South Carolina	103.40	5161.9	0	Right	1	0	52
Georgia	94.10	5792.0	0	Left	1	0	-42
West Virginia	80.80	2190.7	0	Right	1	0	76
Alabama	76.60	4451.4	0	Right	1	0	41
Missouri	71.20	4707.5	0	Right	1	0	47
Mississippi	53.40	3438.6	0	Right	1	0	68
Vermont	55.20	4271.2	0	Right	1	0	44
Minnesota	51.20	4615.8	0	Right	1	0	49
Washington	62.10	7017.1	0	Left	1	0	-49
Texas	54.30	7722.4	0	Left	1	0	-49
Arkansas	43.90	4245.2	0	Right	1	0	65
Oklahoma	44.10	6025.6	0	Left	1	0	-43
Idaho	11.50	4156.3	0	Right	1	0	69
Oregon	27.40	6969.9	0	Left	1	0	-53
South Dakota	9.10	2678.0	0	Right	1	0	67
North Dakota	9.40	2833.0	0	Right	1	0	52
Nevada	7.30	6371.4	0	Right	1	0	50

Chapter 36

PMENU Procedure

Overview: PMENU Procedure

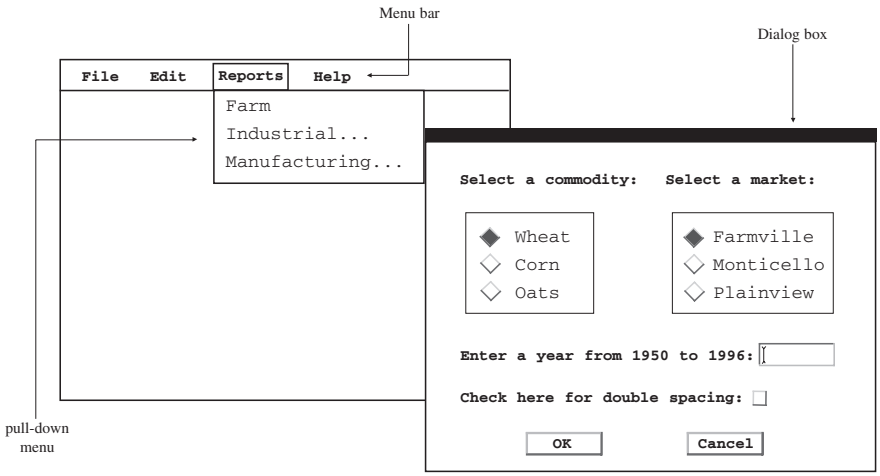
The PMENU procedure defines menus that can be used in DATA step windows, macro windows, both SAS/AF and SAS/FSP windows, or in any SAS application that enables you to specify customized menus.

Menus can replace the command line as a way to execute commands. To activate menus, issue the PMENU command from any command line. Menus must be activated in order for them to appear.

When menus are activated, each active window has a menu bar, which lists items that you can select. Depending on which item you select, SAS either processes a command, displays a menu or a submenu, or requests that you complete information in a dialog box. The dialog box is simply a box of questions or choices that require answers before

an action can be performed. The following figure illustrates features that you can create with PROC PMENU.

Figure 36.1 *Menu Bar, Menu, and Dialog Box*



Note: A menu bar in some operating environments might appear as a pop-up menu or might appear at the bottom of the window.

The PMENU procedure produces no immediately visible output. It simply builds a catalog entry of type PMENU that can be used later in an application.

Concepts: PMENU Procedure

Procedure Execution

Initiating the Procedure

You can define multiple menus by separating their definitions with RUN statements. A group of statements that ends with a RUN statement is called a RUN group. You must completely define a PMENU catalog entry before submitting a RUN statement. You do not have to restart the procedure after a RUN statement.

You must include an initial MENU statement that defines the menu bar, and you must include all ITEM statements and any SELECTION, MENU, SUBMENU, and DIALOG statements as well as statements that are associated with the DIALOG statement within the same RUN group. For example, the following statements define two separate PMENU catalog entries. Both are stored in the same catalog, but each PMENU catalog entry is independent of the other. In the example, both PMENU catalog entries create menu bars that simply list windowing environment commands the user can select and execute:

```
libname proclib 'SAS-data-library';

proc pmenu catalog=proclib.mycat;
   menu menu1;
   item end;
```

```
   item bye;
run;

   menu menu2;
   item end;
   item pgm;
   item log;
   item output;
run;
```

When you submit these statements, you receive a message that says that the PMENU entries have been created. To display one of these menu bars, you must associate the PMENU catalog entry with a window and then activate the window with the menus turned on, as described in "Steps for Building and Using PMENU Catalog Entries" on page 957.

Ending the Procedure

Submit a QUIT, DATA, or new PROC statement to execute any statements that have not executed and end the PMENU procedure. Submit a RUN CANCEL statement to cancel any statements that have not executed and end the PMENU procedure.

Steps for Building and Using PMENU Catalog Entries

In most cases, building and using PMENU entries requires the following steps:

1. Use PROC PMENU to define the menu bars, menus, and other features that you want. Store the output of PROC PMENU in a SAS catalog. For more information, see "Associating a Menu with a Window" on page 989.
2. Define a window using SAS/AF and SAS/FSP software, or the WINDOW or %WINDOW statement in Base SAS software.
3. Associate the PMENU catalog entry created in step 1 with a window by using one of the following:
   - the MENU= option in the WINDOW statement in Base SAS software. For more information, see "Associating a Menu with a Window" on page 989.
   - the MENU= option in the %WINDOW statement in the macro facility.
   - the **Command Menu** field in the GATTR window in PROGRAM entries in SAS/AF software.
   - the Keys, Pmenu, and **Commands** window in a FRAME entry in SAS/AF software. See "Example 5: Associating Menus with a FRAME Application" on page 992.
   - the PMENU function in SAS/AF and SAS/FSP software.
   - the SETPMENU command in SAS/FSP software. See "Example 1: Building a Menu Bar for an FSEDIT Application" on page 972.
4. Activate the window that you created. Make sure that the menus are turned on.

Templates for Coding PROC PMENU Steps

The following coding templates summarize how to use the statements in the PMENU procedure. Refer to descriptions of the statements for more information:

- Build a simple menu bar. All items on the menu bar are windowing environment commands:

```
proc pmenu;
   menu menu-bar;
   item command;
   ...more-ITEM-statements...
run;
```

- Create a menu bar with an item that produces a menu:

```
proc pmenu;
    menu menu-bar;
    item 'menu-item' menu=pull-down-menu;
    ...more-ITEM-statements...
    menu pull-down-menu;
    ...ITEM-statements-for-pull-down-menu...
run;
```

- Create a menu bar with an item that submits a command other than the one that appears on the menu bar:

```
proc pmenu;
   menu menu-bar;
   item 'menu-item' selection=selection;
   ...more-ITEM-statements...
   selection selection 'command-string';
run;
```

- Create a menu bar with an item that opens a dialog box, which displays information and requests text input:

```
proc pmenu;
   menu menu-bar;
   item 'menu-item' menu=pull-down-menu;
   ...more-ITEM-statements...
   menu pull-down-menu;
      item 'menu-item' dialog=dialog-box;
      dialog dialog-box 'command @1';
         text #line @column 'text';
         text #line @column LEN=field-length;
run;
```

- Create a menu bar with an item that opens a dialog box, which permits one choice from a list of possible values:

```
proc pmenu;
   menu menu-bar;
   item 'menu-item' menu=pull-down-menu;
   ...more-ITEM-statements...
   menu pull-down-menu;
      item 'menu-item' dialog=dialog-box;
      dialog dialog-box 'command %1';
         text #line @column 'text';
         radiobox default=button-number;
         rbutton #line @column
                 'text-for-selection';
         ...more-RBUTTON-statements...
run;
```

- Create a menu bar with an item that opens a dialog box, which permits several independent choices:

```
proc pmenu;
   menu menu-bar;
   item 'menu-item' menu=pull-down-menu;
   ...more-ITEM-statements...
   menu pull-down-menu;
      item 'menu-item' dialog=dialog-box;
      dialog dialog-box 'command &1';
         text #line @column 'text';
         checkbox #line @column 'text';
         ...more-CHECKBOX-statements...
run;
```

Syntax: PMENU Procedure

Restriction: You must use at least one MENU statement followed by at least one ITEM statement.

Tips: Supports RUN group processing

You can also use appropriate global statements with this procedure. See "Fundamental Concepts for Using Base SAS Procedures" on page 17 for a list.

See: "PMENU Procedure: Windows" in *SAS Companion for Windows*

"PMENU Procedure: UNIX" in *SAS Companion for UNIX Environments*

"PMENU Procedure: z/OS" in *SAS Companion for z/OS*

PROC PMENU <CATALOG=<*libref.*>*catalog*>
<DESC '*entry-description*'>;
MENU *menu-bar*;
ITEM *command* <*option(s)*>;
ITEM *'menu-item'* <*option(s)*>;
DIALOG *dialog-box* '*command-string field-number-specification*';
CHECKBOX <ON> *#line @column 'text-for-selection'*
<COLOR=*color*> <SUBSTITUTE=*'text-for-substitution'*>;
RADIOBOX DEFAULT=*button-number*;
RBUTTON <NONE> *#line @column 'text-for-selection'*
<COLOR=*color*> <SUBSTITUTE=*'text-for-substitution'*>;
TEXT *#line @column field-description*
<ATTR=*attribute*> <COLOR=*color*>;
MENU *pull-down-menu*;
SELECTION *selection 'command-string'*;
SEPARATOR;
SUBMENU *submenu-name SAS-file*;

Statement	Task	Example
"PROC PMENU Statement"	Define customized menus	Ex. 1
"CHECKBOX Statement"	Define choices a user can make in a dialog box	
"DIALOG Statement"	Describe a dialog box that is associated with an item in a menu	Ex. 2, Ex. 3, Ex. 4
"ITEM Statement"	Identify an item to be listed in a menu bar or in a menu	Ex. 1, Ex. 3, Ex. 5
"MENU Statement"	Name the catalog entry or define a menu	Ex. 1, Ex. 5
"RADIOBOX Statement"	List and define mutually exclusive choices within a dialog box	Ex. 3
"RBUTTON Statement"	List and define mutually exclusive choices within a dialog box	Ex. 3
"SELECTION Statement"	Define a command that is submitted when an item is selected	Ex. 1, Ex. 4
"SEPARATOR Statement"	Draw a line between items in a menu	
"SUBMENU Statement"	Define a common submenu associated with an item	Ex. 1
"TEXT Statement"	Specify text and the input fields for a dialog box	Ex. 2

PROC PMENU Statement

Invokes the PMENU procedure and specifies where to store all PMENU catalog entries that are created in the PROC PMENU step.

Example: "Example 1: Building a Menu Bar for an FSEDIT Application" on page 972

Syntax

PROC PMENU <CATALOG=<*libref.*>*catalog*>
<DESC '*entry-description*'>;

Optional Arguments

CATALOG=<*libref.*>*catalog*
specifies the catalog in which you want to store PMENU entries.

Default: If you omit *libref*, then the PMENU entries are stored in a catalog in the SASUSER library. If you omit CATALOG=, then the entries are stored in the SASUSER.PROFILE catalog.

Example: "Example 1: Building a Menu Bar for an FSEDIT Application" on page 972

DESC ***'entry-description'***
provides a description for the PMENU catalog entries created in the step.

Default: Menu description

Note: These descriptions are displayed when you use the CATALOG window in the windowing environment or the CONTENTS statement in the CATALOG procedure.

CHECKBOX Statement

Defines choices that a user can make within a dialog box.

Restriction: Must be used after a DIALOG statement.

Syntax

CHECKBOX <ON> #*line* @*column* '*text-for-selection*'
<COLOR=*color*> <SUBSTITUTE='*text-for-substitution*'>;

Required Arguments

column
specifies the column in the dialog box where the check box and text are placed.

line
specifies the line in the dialog box where the check box and text are placed.

text-for-selection
defines the text that describes this check box. This text appears in the window and, if the SUBSTITUTE= option is not used, is also inserted into the command in the preceding DIALOG statement when the user selects the check box.

Optional Arguments

COLOR=*color*
defines the color of the check box and the text that describes it.

ON
indicates that by default this check box is active. If you use this option, then you must specify it immediately after the CHECKBOX keyword.

SUBSTITUTE=*'text-for-substitution'*
specifies the text that is to be inserted into the command in the DIALOG statement.

Details

Check Boxes in a Dialog Box

Each CHECKBOX statement defines a single item that the user can select independent of other selections. That is, if you define five choices with five CHECKBOX statements, then the user can select any combination of these choices. When the user selects choices, the text-for-selection values that are associated with the selections are inserted into the command string of the previous DIALOG statement at field locations prefixed by an ampersand (&).

DIALOG Statement

Describes a dialog box that is associated with an item on a menu.

Restriction: Must be followed by at least one TEXT statement.

Examples: "Example 2: Collecting User Input in a Dialog Box" on page 975
"Example 3: Creating a Dialog Box to Search Multiple Variables" on page 978
"Example 4: Creating Menus for a DATA Step Window Application" on page 986

Syntax

DIALOG *dialog-box* '*command-string field-number-specification*';

Required Arguments

command-string
is the command or partial command that is executed when the item is selected. The limit of the *command-string* that results after the substitutions are made is the command-line limit for your operating environment. Typically, the command-line limit is approximately 80 characters.

The limit for '*command-string field-number-specification*' is 200 characters.

Note: If you are using PROC PMENU to submit any command that is valid only in the PROGRAM EDITOR window (such as the INCLUDE command), then you must have the windowing environment running, and you must return control to the PROGRAM EDITOR window.

dialog-box
is the same name specified for the DIALOG= option in a previous ITEM statement.

field-number-specification
can be one or more of the following:

@1…@*n* %1…%*n* &1…&*n*

You can embed the field numbers, for example @1, %1, or &1, in the command string and mix different types of field numbers within a command string. The numeric portion of the field number corresponds to the relative position of TEXT, RADIOBOX, and CHECKBOX statements, not to any actual number in these statements.

@1…@*n*
are optional TEXT statement numbers that can add information to the command before it is submitted. Numbers preceded by an at sign (@) correspond to TEXT statements that use the LEN= option to define input fields.

%1…%*n*
are optional RADIOBOX statement numbers that can add information to the command before it is submitted. Numbers preceded by a percent sign (%) correspond to RADIOBOX statements following the DIALOG statement.

Note: Keep in mind that the numbers correspond to RADIOBOX statements, not to RBUTTON statements.

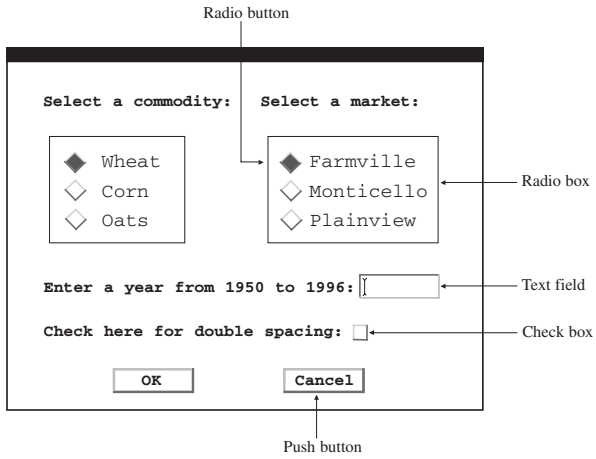
&1…&*n*
are optional CHECKBOX statement numbers that can add information to the command before it is submitted. Numbers preceded by an ampersand (&) correspond to CHECKBOX statements following the DIALOG statement.

Note: To specify a literal @ (at sign), % (percent sign), or & (ampersand) in the *command-string*, use a double character: @@ (at signs), %% (percent signs), or && (ampersands).

Details

- You cannot control the placement of the dialog box. The dialog box is not scrollable. The size and placement of the dialog box are determined by your windowing environment.
- To use the DIALOG statement, specify an ITEM statement with the DIALOG= option in the ITEM statement.
- The ITEM statement creates an entry in a menu bar or in a menu, and the DIALOG= option specifies which DIALOG statement describes the dialog box.
- You can use CHECKBOX, RADIOBOX, and RBUTTON statements to define the contents of the dialog box.
- The following figure shows a typical dialog box. A dialog box can request information in three ways:
  - Fill in a field. Fields that accept text from a user are called text fields.
  - Choose from a list of mutually exclusive choices. A group of selections of this type is called a radio button, and each individual selection is called a radio button.
  - Indicate whether you want to select other independent choices. For example, you could choose to use various options by selecting any or all of the listed selections. A selection of this type is called a check box.

Figure 36.2 *A Typical Dialog Box*



Dialog boxes have two or more buttons, such as OK and Cancel, automatically built into the box.[1] A button causes an action to occur.

1 The actual names of the buttons vary in different windowing environments.

ITEM Statement

Identifies an item to be listed in a menu bar or in a menu.

Examples: "Example 1: Building a Menu Bar for an FSEDIT Application" on page 972
"Example 3: Creating a Dialog Box to Search Multiple Variables" on page 978
"Example 5: Associating Menus with a FRAME Application" on page 992

Syntax

ITEM *command* <*option(s)*> <*action-options*>;
ITEM '*menu-item*' <*option(s)*> <*action-options*>;

Summary of Optional Arguments

ACCELERATE=*name-of-key*
: defines a key sequence that can be used instead of selecting an item.

action-option
: specifies the action for the item.

GRAY
: indicates that the item is not an active choice in this window.

HELP='*help-text*'
: specifies text that is displayed when the user displays the menu item.

ID=*integer*
: specifies a value that is used as an identifier for an item in a menu.

MNEMONIC=*character*
: defines a single character that can select the item.

STATE=CHECK|RADIO
: places a check box or a radio button next to an item.

Required Arguments

command
: a single word that is a valid SAS command for the window in which the menu appears. Commands that are more than one word, such as WHERE CLEAR, must be enclosed in single quotation marks. The *command* appears in uppercase letters on the menu bar.

  If you want to control the case of a SAS command on the menu, then enclose the command in single quotation marks. The case that you use then appears on the menu.

'menu-item'
: a word or text string, enclosed in quotation marks, that describes the action that occurs when the user selects this item. A menu item should not begin with a percent sign (%).

Optional Arguments

ACCELERATE=*name-of-key*
defines a key sequence that can be used instead of selecting an item. When the user presses the key sequence, it has the same effect as selecting the item from the menu bar or menu.

Restrictions:
The functionality of this option is limited to only a few characters. For details, see the SAS documentation for your operating environment.

This option is not available in all operating environments. If you include this option and it is not available in your operating environment, then the option is ignored.

action-option
is one of the following:

DIALOG=*dialog-box*
specifies the name of an associated DIALOG statement, which displays a dialog box when the user selects this item.

Example: "Example 3: Creating a Dialog Box to Search Multiple Variables" on page 978

MENU=*pull-down-menu*
specifies the name of an associated MENU statement, which displays a menu when the user selects this item.

See: "Example 1: Building a Menu Bar for an FSEDIT Application" on page 972

SELECTION=*selection*
specifies the name of an associated SELECTION statement, which submits a command when the user selects this item.

See: "Example 1: Building a Menu Bar for an FSEDIT Application" on page 972

SUBMENU=*submenu*
associates the item with a common submenu.

specifies the name of an associated SUBMENU statement, which displays a pmenu entry when the user selects this item.

See: "Example 1: Building a Menu Bar for an FSEDIT Application" on page 972

If no DIALOG=, MENU=, SELECTION=, or SUBMENU= option is specified, then the *command* or *menu-item* text string is submitted as a command-line command when the user selects the item.

GRAY
indicates that the item is not an active choice in this window. This option is useful when you want to define standard lists of items for many windows, but not all items are valid in all windows. When this option is set and the user selects the item, no action occurs.

HELP='*help-text*'
specifies text that is displayed when the user displays the menu item. For example, if you use a mouse to pull down a menu, then position the mouse pointer over the item and the text is displayed.

Restriction: This option is not available in all operating environments. If you include this option and it is not available in your operating environment, then the option is ignored.

Tip: The place where the text is displayed is operating environment-specific.

ID=*integer*
specifies a value that is used as an identifier for an item in a menu. This identifier is used within a SAS/AF application to selectively activate or deactivate items in a menu or to set the state of an item as a check box or a radio button.

Restrictions:
Integers from 0 to 3000 are reserved for operating environment and SAS use.

This option is not available in all operating environments. If you include this option and it is not available in your operating environment, then the option is ignored.

Note: The minimum value allowed is 3001.

Tips:
ID= is useful with the WINFO function in SAS Component Language.

You can use the same ID for more than one item.

See: "STATE=CHECK|RADIO" on page 966

MNEMONIC=*character*
underlines the first occurrence of *character* in the text string that appears on the menu. The *character* must be in the text string.

The *character* is typically used in combination with another key, such as ALT. When you use the key sequence, it has the same effect as putting your cursor on the item. But it *does not* invoke the action that the item controls.

Restriction: This option is not available in all operating environments. If you include this option and it is not available in your operating environment, then the option is ignored.

STATE=CHECK|RADIO
provides the ability to place a check box or a radio button next to an item that has been selected.

Restriction: This option is not available in all operating environments. If you include this option and it is not available in your operating environment, then the option is ignored.

Tip: STATE= is used with the ID= option and the WINFO function in SAS Component Language.

Details

Defining Items on the Menu Bar

You must use ITEM statements to name all the items that appear in a menu bar. You also use the ITEM statement to name the items that appear in any menus. The items that you specify in the ITEM statement can be commands that are issued when the user selects the item, or they can be descriptions of other actions that are performed by associated DIALOG, MENU, SELECTION, or SUBMENU statements.

All ITEM statements for a menu must be placed immediately after the MENU statement and before any DIALOG, SELECTION, SUBMENU, or other MENU statements. In some operating environments, you can insert SEPARATOR statements between ITEM statements to produce lines separating groups of items in a menu. See "SEPARATOR Statement" on page 970 for more information.

Note: If you specify a menu bar that is too long for the window, then it might be truncated or wrapped to multiple lines.

MENU Statement

Names the catalog entry that stores the menus or defines a menu.

Examples: "Example 1: Building a Menu Bar for an FSEDIT Application" on page 972
"Example 5: Associating Menus with a FRAME Application" on page 992

Syntax

MENU *menu-bar*;

MENU *pull-down-menu*;

Required Arguments

One of the following arguments is required:

menu-bar
names the catalog entry that stores the menus.

pull-down-menu
names the menu that appears when the user selects an item in the menu bar. The value of *pull-down-menu* must match the *pull-down-menu* name that is specified in the MENU= option in a previous ITEM statement.

Details

Defining Menus

When used to define a menu, the MENU statement must follow an ITEM statement that specifies the MENU= option. Both the ITEM statement and the MENU statement for the menu must be in the same RUN group as the MENU statement that defines the menu bar for the PMENU catalog entry.

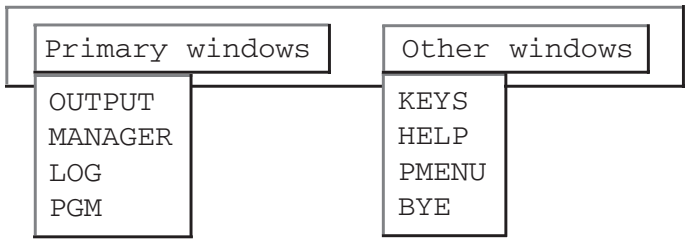
For both menu bars and menus, follow the MENU statement with ITEM statements that define each of the items that appear on the menu. Group all ITEM statements for a menu together. For example, the following PROC PMENU step creates one catalog entry, WINDOWS, which produces a menu bar with two items, **Primary windows** and **Other windows**. When you select one of these items, a menu is displayed.

```
libname proclib 'SAS-data-library';

proc pmenu cat=proclib.mycat;

      /* create catalog entry */
   menu windows;
   item 'Primary windows' menu=prime;
   item 'Other windows' menu=other;

      /* create first menu */
   menu prime;
   item output;
   item manager;
   item log;
   item pgm;
```

```
      /* create second menu */
   menu other;
   item keys;
   item help;
   item pmenu;
   item bye;

   /* end of run group */
run;
```

The following figure shows the resulting menu selections.

Figure 36.3 *Menu*



RADIOBOX Statement

Defines a box that contains mutually exclusive choices within a dialog box.

Restrictions: Must be used after a DIALOG statement.

Must be followed by one or more RBUTTON statements.

Example: "Example 3: Creating a Dialog Box to Search Multiple Variables" on page 978

Syntax

RADIOBOX DEFAULT=*button-number*;

Required Argument

DEFAULT=*button-number*
indicates which radio button is the default.

Default: 1

Details

The RADIOBOX statement indicates the beginning of a list of selections. Immediately after the RADIOBOX statement, you must list an RBUTTON statement for each of the selections the user can make. When the user makes a choice, the text value that is associated with the selection is inserted into the command string of the previous DIALOG statement at field locations prefixed by a percent sign (%).

RBUTTON Statement

Lists mutually exclusive choices within a dialog box.

Restriction: Must be used after a RADIOBOX statement.

Example: "Example 3: Creating a Dialog Box to Search Multiple Variables" on page 978

Syntax

RBUTTON <NONE> #*line* @*column*
'*text-for-selection*' <COLOR=*color*> <SUBSTITUTE='*text-for-substitution*'>;

Required Arguments

column
specifies the column in the dialog box where the radio button and text are placed.

line
specifies the line in the dialog box where the radio button and text are placed.

text-for-selection
defines the text that appears in the dialog box and, if the SUBSTITUTE= option is not used, defines the text that is inserted into the command in the preceding DIALOG statement.

Note: Be careful not to overlap columns and lines when placing text and radio buttons. If you overlap text and buttons, Then you will get an error message. Also, specify space between other text and a radio button.

Optional Arguments

COLOR=*color*
defines the color of the radio button and the text that describes the button.

Restriction: This option is not available in all operating environments. If you include this option and it is not available in your operating environment, then the option is ignored.

NONE
defines a button that indicates none of the other choices. Defining this button enables the user to ignore any of the other choices. No characters, including blanks, are inserted into the DIALOG statement.

Restriction: If you use this option, then it must appear immediately after the RBUTTON keyword.

SUBSTITUTE=*'text-for-substitution'*
specifies the text that is to be inserted into the command in the DIALOG statement.

See: "Example 3: Creating a Dialog Box to Search Multiple Variables" on page 978

SELECTION Statement

Defines a command that is submitted when an item is selected.

Restriction: Must be used after an ITEM statement

Examples: "Example 1: Building a Menu Bar for an FSEDIT Application" on page 972
"Example 4: Creating Menus for a DATA Step Window Application" on page 986

Syntax

SELECTION *selection* '*command-string*';

Required Arguments

selection
is the same name specified for the SELECTION= option in a previous ITEM statement.

command-string
is a text string, enclosed in quotation marks, that is submitted as a command-line command when the user selects this item. There is a limit of 200 characters for *command-string*. However, the command-line limit of approximately 80 characters cannot be exceeded. The command-line limit differs slightly for various operating environments.

Note: SAS uses only the first eight characters of an item that is specified with a SELECTION statement. When a user selects an item from a menu list, the first eight characters of each item name in the list must be unique so that SAS can select the correct item in the list. If the first eight characters are not unique, SAS selects the last item in the list.

Details

You define the name of the item in the ITEM statement and specify the SELECTION= option to associate the item with a subsequent SELECTION statement. The SELECTION statement then defines the actual command that is submitted when the user chooses the item in the menu bar or menu.

You are likely to use the SELECTION statement to define a command string. You create a simple alias by using the ITEM statement, which invokes a longer command string that is defined in the SELECTION statement. For example, you could include an item in the menu bar that invokes a WINDOW statement to enable data entry. The actual commands that are processed when the user selects this item are the commands to include and submit the application.

Note: If you are using PROC PMENU to issue any command that is valid only in the PROGRAM EDITOR window (such as the INCLUDE command), then you must have the windowing environment running. Also, you must return control to the PROGRAM EDITOR window.

SEPARATOR Statement

Draws a line between items on a menu.

Restrictions: Must be used after an ITEM statement.

Not available in all operating environments.

Syntax

SEPARATOR;

SUBMENU Statement

Specifies the SAS file that contains a common submenu associated with an item.

Example: "Example 1: Building a Menu Bar for an FSEDIT Application" on page 972

Syntax

SUBMENU *submenu-name SAS-file*;

Required Arguments

submenu-name
specifies a name for the submenu statement. To associate a submenu with a menu item, *submenu-name* must match the submenu name specified in the SUBMENU= action-option in the ITEM statement.

SAS-file
specifies the name of the SAS file that contains the common submenu.

TEXT Statement

Specifies text and the input fields for a dialog box.

Restriction: Can be used only after a DIALOG statement.

Example: "Example 2: Collecting User Input in a Dialog Box" on page 975

Syntax

TEXT #*line* @*column*
field-description <ATTR=*attribute*> <COLOR=*color*>;

Required Arguments

column
specifies the starting column for the text or input field.

field-description
defines how the TEXT statement is used. The *field-description* can be one of the following:

LEN=*field-length*
is the length of an input field in which the user can enter information. If the LEN= argument is used, then the information entered in the field is inserted into the command string of the previous DIALOG statement at field locations that are prefixed by an at sign (@).

See: "Example 2: Collecting User Input in a Dialog Box" on page 975

'*text*'
is the text string that appears inside the dialog box at the location defined by *line* and *column*.

line
specifies the line number for the text or input field.

Optional Arguments

ATTR=*attribute*
defines the attribute for the text or input field. These are valid attribute values:

- BLINK
- HIGHLIGH
- REV_VIDE
- UNDERLIN

Restrictions:
This option is not available in all operating environments. If you include this option and it is not available in your operating environment, then the option is ignored.

Your hardware might not support all of these attributes.

COLOR=*color*
defines the color for the text or input field characters. Here are the color values that you can use:

BLACK	BROWN
GRAY	MAGENTA
PINK	WHITE
BLUE	CYAN
GREEN	ORANGE
RED	YELLOW

Restrictions:
This option is not available in all operating environments. If you include this option and it is not available in your operating environment, then the option is ignored.

Your hardware might not support all of these colors.

Examples: PMENU Procedure

Example 1: Building a Menu Bar for an FSEDIT Application

Features: PROC PMENU statement option
CATALOG=

ITEM statement options

MENU=
SELECTION=
SUBMENU=

MENU statement

SELECTION statement

SUBMENU statement

Details

This example creates a menu bar that can be used in an FSEDIT application to replace the default menu bar. The selections available on these menus do not enable end users to delete or duplicate observations.

Note:

- The windows in the PROC PMENU examples were produced in the UNIX environment and might appear slightly different from the same windows in other operating environments.
- You should know the operating environment-specific system options that can affect how menus are displayed and merged with existing SAS menus. For details, see the SAS documentation for your operating environment.

Program

```
libname proclib
'SAS-data-library';

proc pmenu catalog=proclib.menucat;

   menu project;

      item 'File' menu=f;
      item 'Edit' submenu=editmnu;
      item 'Scroll' menu=s;
      item 'Help' menu=h;

      menu f;
         item 'Goback' selection=g;
         item 'Save';
         selection g 'end';

      submenu editmnu sashelp.core.edit;

      menu s;
         item 'Next Obs' selection=n;
         item 'Prev Obs' selection=p;
         item 'Top';
         item 'Bottom';
         selection n 'forward';
         selection p 'backward';

      menu h;
         item 'Keys';
         item 'About this application' selection=hlp;
         selection hlp 'sethelp user.menucat.staffhlp.help;help';
quit;
```

Program Description

Declare the PROCLIB library. The PROCLIB library is used to store menu definitions.

```
libname proclib
'SAS-data-library';
```

Specify the catalog for storing menu definitions. Menu definitions will be stored in the PROCLIB.MENUCAT catalog.

```
proc pmenu catalog=proclib.menucat;
```

Specify the name of the catalog entry. The MENU statement specifies PROJECT as the name of the catalog entry. The menus are stored in the catalog entry PROCLIB.MENUCAT.PROJECT.PMENU.

```
menu project;
```

Design the menu bar. The ITEM statements specify the items for the menu bar. The value of the MENU= option is used in a subsequent MENU statement. The Edit item uses a common predefined submenu; the menus for the other items are defined in this PROC step.

```
item 'File' menu=f;
item 'Edit' submenu=editmnu;
item 'Scroll' menu=s;
item 'Help' menu=h;
```

Design the File menu. This group of statements defines the selections available under File on the menu bar. The first ITEM statement specifies **Goback** as the first selection under File. The value of the SELECTION= option corresponds to the subsequent SELECTION statement, which specifies END as the command that is issued for that selection. The second ITEM statement specifies that the SAVE command is issued for that selection.

```
menu f;
   item 'Goback' selection=g;
   item 'Save';
   selection g 'end';
```

Add the EDITMNU submenu. The SUBMENU statement associates a predefined submenu that is located in the SAS file SASHELP.CORE.EDIT with the Edit item on the menu bar. The name of this SUBMENU statement is EDITMNU, which corresponds with the name in the SUBMENU= action-option in the ITEM statement for the Edit item.

```
submenu editmnu sashelp.core.edit;
```

Design the Scroll menu. This group of statements defines the selections available under Scroll on the menu bar.

```
menu s;
   item 'Next Obs' selection=n;
   item 'Prev Obs' selection=p;
   item 'Top';
   item 'Bottom';
   selection n 'forward';
   selection p 'backward';
```

Design the Help menu. This group of statements defines the selections available under Help on the menu bar. The SETHELP command specifies a HELP entry that contains user-written information for this FSEDIT application. The semicolon that appears after the HELP entry name enables the HELP command to be included in the string. The HELP command invokes the HELP entry.

```
        menu h;
           item 'Keys';
           item 'About this application' selection=hlp;
           selection hlp 'sethelp user.menucat.staffhlp.help;help';
quit;
```

Associating a Menu Bar with an FSEDIT Session

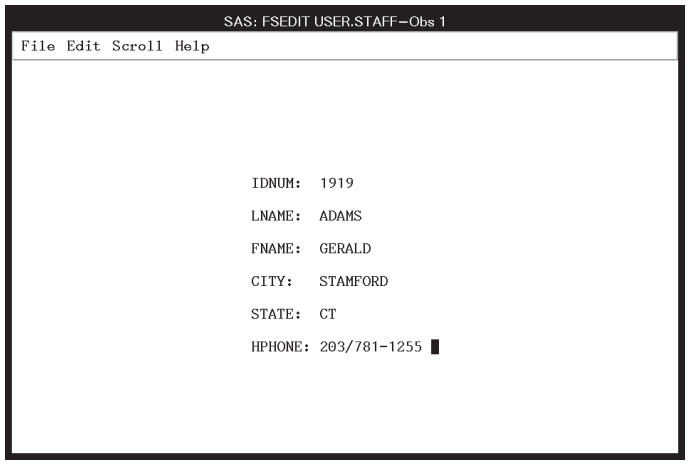
The following SETPMENU command associates the customized menu bar with the FSEDIT window.

```
setpmenu proclib.menucat.project.pmenu;pmenu on
```

You can also specify the menu bar on the command line in the FSEDIT session or by issuing a CALL EXECCMD command in SAS Component Language (SCL).

For other methods of associating the customized menu bar with the FSEDIT window, see "Associating a Menu Bar with an FSEDIT Session" on page 983.

The following FSEDIT window shows the menu bar:



Example 2: Collecting User Input in a Dialog Box

Features: DIALOG statement

TEXT statement option
LEN=

Details

This example adds a dialog box to the menus created in "Example 1: Building a Menu Bar for an FSEDIT Application" on page 972. The dialog box enables the user to use a WHERE clause to subset the SAS data set.

Tasks include these:

- collecting user input in a dialog box
- creating customized menus for an FSEDIT application

Program

```
libname proclib
'SAS-data-library';

proc pmenu catalog=proclib.menucat;

   menu project;

      item 'File' menu=f;
      item 'Edit' menu=e;
      item 'Scroll' menu=s;
      item 'Subset' menu=sub;
      item 'Help' menu=h;

      menu f;
         item 'Goback' selection=g;
         item 'Save';
         selection g 'end';

      menu e;
         item 'Cancel';
         item 'Add';

      menu s;
         item 'Next Obs' selection=n;
         item 'Prev Obs' selection=p;
         item 'Top';
         item 'Bottom';
         selection n 'forward';
         selection p 'backward';

      menu sub;
         item 'Where' dialog=d1;
         item 'Where Clear';

      menu h;
         item 'Keys';
         item 'About this application' selection=hlp;
         selection hlp 'sethelp proclib.menucat.staffhlp.help;help';

      dialog d1 'where @1';
         text #2 @3 'Enter a valid WHERE clause or UNDO';
         text #4 @3 'WHERE ';
         text #4 @10 len=40;
quit;
```

Program Description

Declare the PROCLIB library. The PROCLIB library is used to store menu definitions.

```
libname proclib
'SAS-data-library';
```

Specify the catalog for storing menu definitions. Menu definitions will be stored in the PROCLIB.MENUCAT catalog.

```
proc pmenu catalog=proclib.menucat;
```

Specify the name of the catalog entry. The MENU statement specifies PROJECT as the name of the catalog entry. The menus are stored in the catalog entry PROCLIB.MENUCAT.PROJECT.PMENU.

```
menu project;
```

Design the menu bar. The ITEM statements specify the items for the menu bar. The value of the MENU= option is used in a subsequent MENU statement.

```
item 'File' menu=f;
item 'Edit' menu=e;
item 'Scroll' menu=s;
item 'Subset' menu=sub;
item 'Help' menu=h;
```

Design the File menu. This group of statements defines the selections under File on the menu bar. The first ITEM statement specifies **`Goback`** as the first selection under File. The value of the SELECTION= option corresponds to the subsequent SELECTION statement, which specifies END as the command that is issued for that selection. The second ITEM statement specifies that the SAVE command is issued for that selection.

```
menu f;
   item 'Goback' selection=g;
   item 'Save';
   selection g 'end';
```

Design the Edit menu. This group of statements defines the selections available under Edit on the menu bar.

```
menu e;
   item 'Cancel';
   item 'Add';
```

Design the Scroll menu. This group of statements defines the selections available under Scroll on the menu bar.

```
menu s;
   item 'Next Obs' selection=n;
   item 'Prev Obs' selection=p;
   item 'Top';
   item 'Bottom';
   selection n 'forward';
   selection p 'backward';
```

Design the Subset menu. This group of statements defines the selections available under Subset on the menu bar. The value d1 in the DIALOG= option is used in the subsequent DIALOG statement.

```
menu sub;
   item 'Where' dialog=d1;
   item 'Where Clear';
```

Design the Help menu. This group of statements defines the selections available under Help on the menu bar. The SETHELP command specifies a HELP entry that contains user-written information for this FSEDIT application. The semicolon enables the HELP command to be included in the string. The HELP command invokes the HELP entry.

```
menu h;
   item 'Keys';
   item 'About this application' selection=hlp;
   selection hlp 'sethelp proclib.menucat.staffhlp.help;help';
```

Design the dialog box. The DIALOG statement builds a WHERE command. The arguments for the WHERE command are provided by user input into the text entry fields described by the three TEXT statements. The @1 notation is a placeholder for user input in the text field. The TEXT statements specify the text in the dialog box and the length of the input field.

```
dialog d1 'where @1';
   text #2 @3 'Enter a valid WHERE clause or UNDO';
   text #4 @3 'WHERE ';
   text #4 @10 len=40;
quit;
```

Associating a Menu Bar with an FSEDIT Window

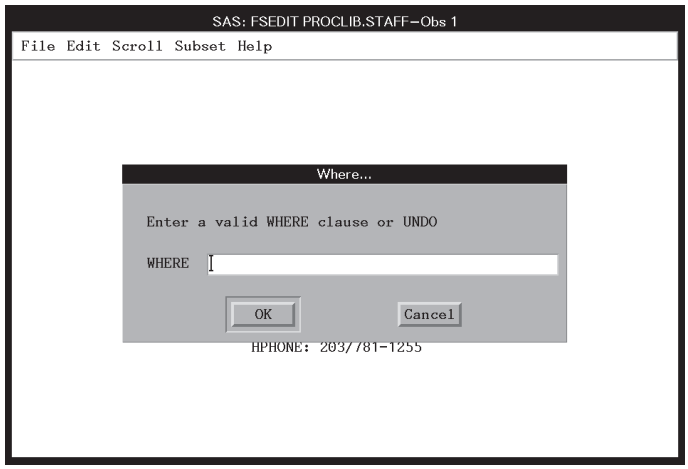
The following SETPMENU command associates the customized menu bar with the FSEDIT window.

```
setpmenu proclib.menucat.project.pmenu;pmenu on
```

You can also specify the menu bar on the command line in the FSEDIT session or by issuing a CALL EXECCMD command in SAS Component Language (SCL). Refer to *SAS(R) Component Language 9.3: Reference* for complete documentation on SCL.

For other methods of associating the customized menu bar with the FSEDIT window, see "Associating a Menu Bar with an FSEDIT Session" on page 983.

The following dialog box appears when the user chooses **Subset** and then **Where**.



Example 3: Creating a Dialog Box to Search Multiple Variables

Features: DIALOG statement
SAS macro invocation
ITEM statement
DIALOG= option
RADIOBOX statement option
DEFAULT=

RBUTTON statement option
SUBSTITUTE=

Other features: SAS macro invocation

Details

This example shows how to modify the menu bar in an FSEDIT session to enable a search for one value across multiple variables. The example creates customized menus to use in an FSEDIT session. The menu structure is the same as in the preceding example, except for the WHERE dialog box.

When selected, the menu item invokes a macro. The user input becomes values for macro parameters. The macro generates a WHERE command that expands to include all the variables needed for the search.

Tasks include these:

- associating customized menus with an FSEDIT session
- searching multiple variables with a WHERE clause
- extending PROC PMENU functionality with a SAS macro

Program

```
libname proclib
'SAS-data-library';

proc pmenu catalog=proclib.menucat;

   menu project;

      item 'File' menu=f;
      item 'Edit' menu=e;
      item 'Scroll' menu=s;
      item 'Subset' menu=sub;
      item 'Help' menu=h;

      menu f;
         item 'Goback' selection=g;
         item 'Save';
         selection g 'end';

      menu e;
         item 'Cancel';
         item 'Add';

      menu s;
         item 'Next Obs' selection=n;
         item 'Prev Obs' selection=p;
         item 'Top';
         item 'Bottom';
         selection n 'forward';
         selection p 'backward';

      menu sub;
         item 'Where' dialog=d1;
         item 'Where Clear';

      menu h;
         item 'Keys';
```

```
            item 'About this application' selection=hlp;
            selection hlp 'sethelp proclib.menucat.staffhlp.help;help';

          dialog d1 '%%wbuild(%1,%2,@1,%3)';

             text #1 @1 'Choose a region:';
             radiobox default=1;
                rbutton #3 @5 'Northeast' substitute='NE';
                rbutton #4 @5 'Northwest' substitute='NW';
                rbutton #5 @5 'Southeast' substitute='SE';
                rbutton #6 @5 'Southwest' substitute='SW';

             text #8 @1 'Choose a contaminant:';
             radiobox default=1;
                rbutton #10 @5  'Pollutant A' substitute='pol_a,2';
                rbutton #11 @5  'Pollutant B' substitute='pol_b,4';

             text #13 @1 'Enter Value for Search:';
             text #13 @25 len=6;

             text #15 @1 'Choose a comparison criterion:';
             radiobox default=1;
                rbutton #16 @5 'Greater Than or Equal To'
                               substitute='GE';
                rbutton #17 @5 'Less Than or Equal To'
                               substitute='LE';
                rbutton #18 @5 'Equal To' substitute='EQ';
quit;
```

Program Description

Declare the PROCLIB library. The PROCLIB library is used to store menu definitions.

```
libname proclib
'SAS-data-library';
```

Specify the catalog for storing menu definitions. Menu definitions will be stored in the PROCLIB.MENUCAT catalog.

```
proc pmenu catalog=proclib.menucat;
```

Specify the name of the catalog entry. The MENU statement specifies STAFF as the name of the catalog entry. The menus are stored in the catalog entry PROCLIB.MENUCAT.PROJECT.PMENU.

```
    menu project;
```

Design the menu bar. The ITEM statements specify the items for the menu bar. The value of the MENU= option is used in a subsequent MENU statement.

```
      item 'File' menu=f;
      item 'Edit' menu=e;
      item 'Scroll' menu=s;
      item 'Subset' menu=sub;
      item 'Help' menu=h;
```

Design the File menu. This group of statements defines the selections under File on the menu bar. The first ITEM statement specifies **Goback** as the first selection under File. The value of the SELECTION= option corresponds to the subsequent SELECTION

statement, which specifies END as the command that is issued for that selection. The second ITEM statement specifies that the SAVE command is issued for that selection.

```
menu f;
   item 'Goback' selection=g;
   item 'Save';
   selection g 'end';
```

Design the Edit menu. The ITEM statements define the selections under Edit on the menu bar.

```
menu e;
   item 'Cancel';
   item 'Add';
```

Design the Scroll menu. This group of statements defines the selections under Scroll on the menu bar. If the quoted string in the ITEM statement is not a valid command, then the SELECTION= option corresponds to a subsequent SELECTION statement, which specifies a valid command.

```
menu s;
   item 'Next Obs' selection=n;
   item 'Prev Obs' selection=p;
   item 'Top';
   item 'Bottom';
   selection n 'forward';
   selection p 'backward';
```

Design the Subset menu. This group of statements defines the selections under Subset on the menu bar. The DIALOG= option names a dialog box that is defined in a subsequent DIALOG statement.

```
menu sub;
   item 'Where' dialog=d1;
   item 'Where Clear';
```

Design the Help menu. This group of statements defines the selections under Help on the menu bar. The SETHELP command specifies a HELP entry that contains user-written information for this FSEDIT application. The semicolon that appears after the HELP entry name enables the HELP command to be included in the string. The HELP command invokes the HELP entry.

```
menu h;
   item 'Keys';
   item 'About this application' selection=hlp;
   selection hlp 'sethelp proclib.menucat.staffhlp.help;help';
```

Design the dialog box. WBUILD is a SAS macro. The double percent sign that precedes WBUILD is necessary to prevent PROC PMENU from expecting a field number to follow. The field numbers %1, %2, and %3 equate to the values that the user specified with the radio buttons. The field number @1 equates to the search value that the user enters.

```
 dialog d1 '%%wbuild(%1,%2,@1,%3)';
```

Add a radio button for region selection. The TEXT statement specifies text for the dialog box that appears on line 1 and begins in column 1. The RADIOBOX statement specifies that a radio button will appear in the dialog box. DEFAULT= specifies that the first radio button (Northeast) will be selected by default. The RBUTTON statements

specify the mutually exclusive choices for the radio buttons: Northeast, Northwest, Southeast, or Southwest. SUBSTITUTE= gives the value that is substituted for the %1 in the DIALOG statement above if that radio button is selected.

```
text #1 @1 'Choose a region:';
radiobox default=1;
   rbutton #3 @5 'Northeast' substitute='NE';
   rbutton #4 @5 'Northwest' substitute='NW';
   rbutton #5 @5 'Southeast' substitute='SE';
   rbutton #6 @5 'Southwest' substitute='SW';
```

Add a radio button for pollutant selection. The TEXT statement specifies text for the dialog box that appears on line 8 (#8) and begins in column 1 (@1). The RADIOBOX statement specifies that a radio button will appear in the dialog box. DEFAULT= specifies that the first radio button (Pollutant A) will be selected by default. The RBUTTON statements specify the mutually exclusive choices for the radio buttons: Pollutant A or Pollutant B. SUBSTITUTE= gives the value that is substituted for the %2 in the preceding DIALOG statement if that radio button is selected.

```
text #8 @1 'Choose a contaminant:';
radiobox default=1;
   rbutton #10 @5  'Pollutant A' substitute='pol_a,2';
   rbutton #11 @5  'Pollutant B' substitute='pol_b,4';
```

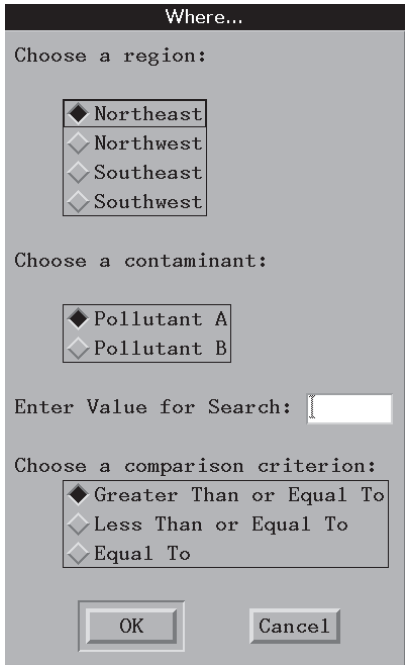
Add an input field. The first TEXT statement specifies text for the dialog box that appears on line 13 and begins in column 1. The second TEXT statement specifies an input field that is 6 bytes long that appears on line 13 and begins in column 25. The value that the user enters in the field is substituted for the @1 in the preceding DIALOG statement.

```
text #13 @1 'Enter Value for Search:';
text #13 @25 len=6;
```

Add a radio button for comparison operator selection. The TEXT statement specifies text for the dialog box that appears on line 15 and begins in column 1. The RADIOBOX statement specifies that a radio button will appear in the dialog box. DEFAULT= specifies that the first radio button (Greater Than or Equal To) will be selected by default. The RBUTTON statements specify the mutually exclusive choices for the radio buttons. SUBSTITUTE= gives the value that is substituted for the %3 in the preceding DIALOG statement if that radio button is selected.

```
text #15 @1 'Choose a comparison criterion:';
radiobox default=1;
   rbutton #16 @5 'Greater Than or Equal To'
                  substitute='GE';
   rbutton #17 @5 'Less Than or Equal To'
                  substitute='LE';
   rbutton #18 @5 'Equal To' substitute='EQ';
quit;
```

The following dialog box appears when the user selects Subset and then Where.



Details

Associating a Menu Bar with an FSEDIT Session

The SAS data set PROCLIB.LAKES has data about several lakes. Two pollutants, pollutant A and pollutant B, were tested at each lake. Tests were conducted for pollutant A twice at each lake, and the results are recorded in the variables POL_A1 and POL_A2. Tests were conducted for pollutant B four times at each lake, and the results are recorded in the variables POL_B1 - POL_B4. Each lake is located in one of four regions. The following output lists the contents of PROCLIB.LAKES:

PROCLIB.LAKES 1

region	lake	pol_a1	pol_a2	pol_b1	pol_b2	pol_b3	pol_b4
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

The "PROCLIB.LAKES" on page 1750 DATA step creates PROCLIB.LAKES.

The following statements initiate a PROC FSEDIT session for PROCLIB.LAKES:

```
proc fsedit data=proclib.lakes screen=proclib.lakes;
run;
```

To associate the customized menu bar menu with the FSEDIT session, do any one of the following:

- enter a SETPMENU command on the command line. The command for this example is

  ```
  setpmenu proclib.menucat.project.pmenu
  ```

  Turn on the menus by entering PMENU ON on the command line.
- enter the SETPMENU command in a Command window.
- include an SCL program with the FSEDIT session that uses the customized menus and turns on the menus, for example:

```
fseinit:
  call execcmd('setpmenu proclib.menucat.project.pmenu;
                pmenu on;');
return;
init:
return;
main:
return;
term:
return;
```

How the WBUILD Macro Works

Consider how you would learn whether any of the lakes in the Southwest region tested for a value of .50 or greater for pollutant A. Without the customized menu item, you would issue the following WHERE command in the FSEDIT window:

```
where region="SW" and (pol_a1 ge .50 or pol_a2 ge .50);
```

Using the custom menu item, you would select **Southwest**, **Pollutant A**, enter .50 as the value, and choose **Greater Than or Equal To** as the comparison criterion. Two lakes, **New Dam** and **Border**, meet the criteria.

The WBUILD macro uses the four pieces of information from the dialog box to generate a WHERE command:

- One of the values for region, either **NE**, **NW**, **SE**, or **SW**, becomes the value of the macro parameter REGION.
- Either **pol_a,2** or **pol_b,4** become the values of the PREFIX and NUMVAR macro parameters. The comma is part of the value that is passed to the WBUILD macro and serves to delimit the two parameters, PREFIX and NUMVAR.
- The value that the user enters for the search becomes the value of the macro parameter VALUE.
- The operator that the user chooses becomes the value of the macro parameter OPERATOR.

To see how the macro works, again consider the following example, in which you want to know whether any of the lakes in the southwest tested for a value of .50 or greater for pollutant A. The values of the macro parameters would be

REGION	**SW**
PREFIX	**pol_a**
NUMVAR	2
VALUE	.50
OPERATOR	GE

The first %IF statement checks to make sure that the user entered a value. If a value has been entered, then the macro begins to generate the WHERE command. First, the macro creates the beginning of the WHERE command:

```
where region="SW" and (
```

Next, the %DO loop executes. For pollutant A, it executes twice because NUMVAR=2. In the macro definition, the period in **&prefix.&i** concatenates **pol_a** with **1** and with **2**. At each iteration of the loop, the macro resolves PREFIX, OPERATOR, and VALUE, and it generates a part of the WHERE command. On the first iteration, it generates **pol_a1 GE .50**

The %IF statement in the loop checks to determine whether the loop is working on its last iteration. If it is not working, then the macro makes a compound WHERE command by putting an **OR** between the individual clauses. The next part of the WHERE command becomes **OR pol_a2 GE .50**

The loop ends after two executions for pollutant A, and the macro generates the end of the WHERE command:

```
)
```

Results from the macro are placed on the command line. The following code is the definition of the WBUILD macro. The underlined code shows the parts of the WHERE command that are text strings that the macro does not resolve:

```
%macro wbuild(region,prefix,numvar,value,operator);
       /* check to see if value is present */
    %if &value ne %then %do;
       where region="&region" AND (
               /* If the values are character,     */
               /* enclose &value in double quotation marks. */
         %do i=1 %to &numvar;
             &prefix.&i &operator &value
                /* if not on last variable,  */
                /* generate 'OR'             */
            %if &i ne &numvar %then %do;
                 OR
            %end;
         %end;
          )
    %end;

%mend wbuild;
```

Example 4: Creating Menus for a DATA Step Window Application

Features: DIALOG statement
SELECTION statement

Other features: FILENAME statement

Details

This example defines an application that enables the user to enter human resources data for various departments, and to request reports from the data sets that are created by the data entry.

The first part of the example describes the PROC PMENU step that creates the menus. The subsequent sections describe how to use the menus in a DATA step window application.

Tasks include these:

- associating customized menus with a DATA step window
- creating menus for a DATA step window
- submitting SAS code from a menu selection
- creating a menu selection that calls a dialog box

Program

```
libname proclib
'SAS-data-library';

filename de     'external-file';
filename prt    'external-file';

 proc pmenu catalog=proclib.menus;

    menu select;

   item 'File' menu=f;
   item 'Data_Entry' menu=deptsde;
   item 'Print_Report' menu=deptsprt;

   menu f;
      item 'End this window' selection=endwdw;
      item 'End this SAS session' selection=endsas;
      selection endwdw 'end';
      selection endsas 'bye';

    menu deptsde;
       item 'For Dept01' selection=de1;
       item 'For Dept02' selection=de2;
       item 'Other Departments' dialog=deother;

       selection de1 'end;pgm;include de;change xx 01;submit';
       selection de2 'end;pgm;include de;change xx 02;submit';

       dialog deother 'end;pgm;include de;c deptxx @1;submit';
          text #1 @1 'Enter department name';
```

```
          text #2 @3 'in the form DEPT99:';
          text #2 @25 len=7;

       menu deptsprt;
          item 'For Dept01' selection=prt1;
          item 'For Dept02' selection=prt2;
          item 'Other Departments' dialog=prother;

          selection prt1
                'end;pgm;include prt;change xx 01 all;submit';
          selection prt2
                'end;pgm;include prt;change xx 02 all;submit';

       dialog prother 'end;pgm;include prt;c deptxx @1 all;submit';
          text #1 @1 'Enter department name';
          text #2 @3 'in the form DEPT99:';
          text #2 @25 len=7;

   run;

     menu entrdata;
        item 'File' menu=f;
        menu f;
           item 'End this window' selection=endwdw;
           item 'End this SAS session' selection=endsas;
           selection endwdw 'end';
           selection endsas 'bye';
   run;
quit;
```

Program Description

Declare the PROCLIB library. The PROCLIB library is used to store menu definitions.

```
libname proclib
'SAS-data-library';
```

Declare the DE and PRT filenames. The FILENAME statements define the external files in which the programs to create the windows are stored.

```
filename de     'external-file';
filename prt    'external-file';
```

Specify the catalog for storing menu definitions. Menu definitions will be stored in the PROCLIB.MENUCAT catalog.

```
 proc pmenu catalog=proclib.menus;
```

Specify the name of the catalog entry. The MENU statement specifies SELECT as the name of the catalog entry. The menus are stored in the catalog entry PROCLIB.MENUS.SELECT.PMENU.

```
    menu select;
```

Design the menu bar. The ITEM statements specify the three items on the menu bar. The value of the MENU= option is used in a subsequent MENU statement.

```
   item 'File' menu=f;
   item 'Data_Entry' menu=deptsde;
   item 'Print_Report' menu=deptsprt;
```

Design the File menu. This group of statements defines the selections under File. The value of the SELECTION= option is used in a subsequent SELECTION statement.

```
menu f;
   item 'End this window' selection=endwdw;
   item 'End this SAS session' selection=endsas;
   selection endwdw 'end';
   selection endsas 'bye';
```

Design the Data_Entry menu. This group of statements defines the selections under Data_Entry on the menu bar. The ITEM statements specify that For Dept01 and For Dept02 appear under Data_Entry. The value of the SELECTION= option equates to a subsequent SELECTION statement, which contains the string of commands that are actually submitted. The value of the DIALOG= option equates to a subsequent DIALOG statement, which describes the dialog box that appears when this item is selected.

```
menu deptsde;
   item 'For Dept01' selection=de1;
   item 'For Dept02' selection=de2;
   item 'Other Departments' dialog=deother;
```

Specify commands under the Data_Entry menu. The commands in single quotation marks are submitted when the user selects For Dept01 or For Dept02. The END command ends the current window and returns to the PROGRAM EDITOR window so that further commands can be submitted. The INCLUDE command includes the SAS statements that create the data entry window. The CHANGE command modifies the DATA statement in the included program so that it creates the correct data set. The SUBMIT command submits the DATA step program.

```
selection de1 'end;pgm;include de;change xx 01;submit';
selection de2 'end;pgm;include de;change xx 02;submit';
```

Design the DEOTHER dialog box. The DIALOG statement defines the dialog box that appears when the user selects Other Departments. The DIALOG statement modifies the command string so that the name of the department that is entered by the user is used to change **deptxx** in the SAS program that is included. The first two TEXT statements specify text that appears in the dialog box. The third TEXT statement specifies an input field. The name that is entered in this field is substituted for the @1 in the DIALOG statement.

```
dialog deother 'end;pgm;include de;c deptxx @1;submit';
   text #1 @1 'Enter department name';
   text #2 @3 'in the form DEPT99:';
   text #2 @25 len=7;
```

Design the Print_Report menu. This group of statements defines the choices under the Print_Report item. These ITEM statements specify that For Dept01 and For Dept02 appear in the menu. The value of the SELECTION= option equates to a subsequent SELECTION statement, which contains the string of commands that are actually submitted.

```
menu deptsprt;
   item 'For Dept01' selection=prt1;
   item 'For Dept02' selection=prt2;
   item 'Other Departments' dialog=prother;
```

Specify commands for the Print_Report menu. The commands in single quotation marks are submitted when the user selects For Dept01 or For Dept02. The END command ends the current window and returns to the PROGRAM EDITOR window so that further commands can be submitted. The INCLUDE command includes the SAS statements that print the report. (For more information, see "Printing a Program" on page 991.) The CHANGE command modifies the PROC PRINT step in the included program so that it prints the correct data set. The SUBMIT command submits the PROC PRINT program.

```
selection prt1
      'end;pgm;include prt;change xx 01 all;submit';
selection prt2
      'end;pgm;include prt;change xx 02 all;submit';
```

Design the PROTHER dialog box. The DIALOG statement defines the dialog box that appears when the user selects Other Departments. The DIALOG statement modifies the command string so that the name of the department that is entered by the user is used to change **deptxx** in the SAS program that is included. The first two TEXT statements specify text that appears in the dialog box. The third TEXT statement specifies an input field. The name entered in this field is substituted for the @1 in the DIALOG statement.

```
dialog prother 'end;pgm;include prt;c deptxx @1 all;submit';
   text #1 @1 'Enter department name';
   text #2 @3 'in the form DEPT99:';
   text #2 @25 len=7;
```

End this RUN group.

```
run;
```

Specify a second catalog entry and menu bar. The MENU statement specifies ENTRDATA as the name of the catalog entry that this RUN group is creating. File is the only item on the menu bar. The selections available are End this window and End this SAS session.

```
  menu entrdata;
     item 'File' menu=f;
     menu f;
        item 'End this window' selection=endwdw;
        item 'End this SAS session' selection=endsas;
        selection endwdw 'end';
        selection endsas 'bye';
run;
quit;
```

Other Examples

Associating a Menu with a Window

The first group of statements defines the primary window for the application. These statements are stored in the file that is referenced by the HRWDW fileref:

The WINDOW statement creates the HRSELECT window. MENU= associates the PROCLIB.MENUS.SELECT.PMENU entry with this window.

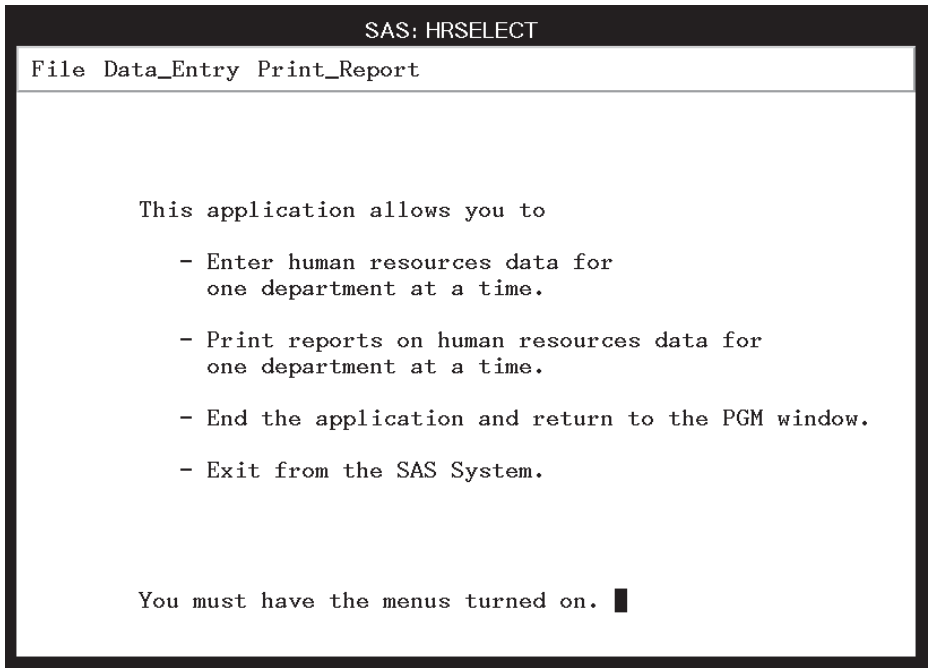
```
data _null_;
   window hrselect menu=proclib.menus.select
```

```
    #4  @10 'This application allows you to'
    #6  @13 '- Enter human resources data for'
    #7  @15 'one department at a time.'
    #9  @13 '- Print reports on human resources data for'
    #10 @15 'one department at a time.'
    #12 @13 '- End the application and return to the PGM window.'
    #14 @13 '- Exit from the SAS System.'
    #19 @10 'You must have the menus turned on.';
```

The DISPLAY statement displays the HRSELECT window.

```
    display hrselect;
run;
```

The HRSELECT window that is displayed by the DISPLAY statement:



Using a Data Entry Program

When the user selects **Data_Entry** from the menu bar in the HRSELECT window, a menu is displayed. When the user selects one of the listed departments or chooses to enter a different department, the following statements are invoked. These statements are stored in the file that is referenced by the DE fileref.

The WINDOW statement creates the HRDATA window. MENU= associates the PROCLIB.MENUS.ENTRDATA.PMENU entry with the window.

```
data proclib.deptxx;
    window hrdata menu=proclib.menus.entrdata
    #5  @10 'Employee Number'
    #8  @10 'Salary'
    #11 @10 'Employee Name'
    #5  @31 empno $4.
    #8  @31 salary 10.
    #11 @31 name $30.
    #19 @10 'Press ENTER to add the observation to the data set.';
```

The DISPLAY statement displays the HRDATA window.

```
    display hrdata;
run;
```

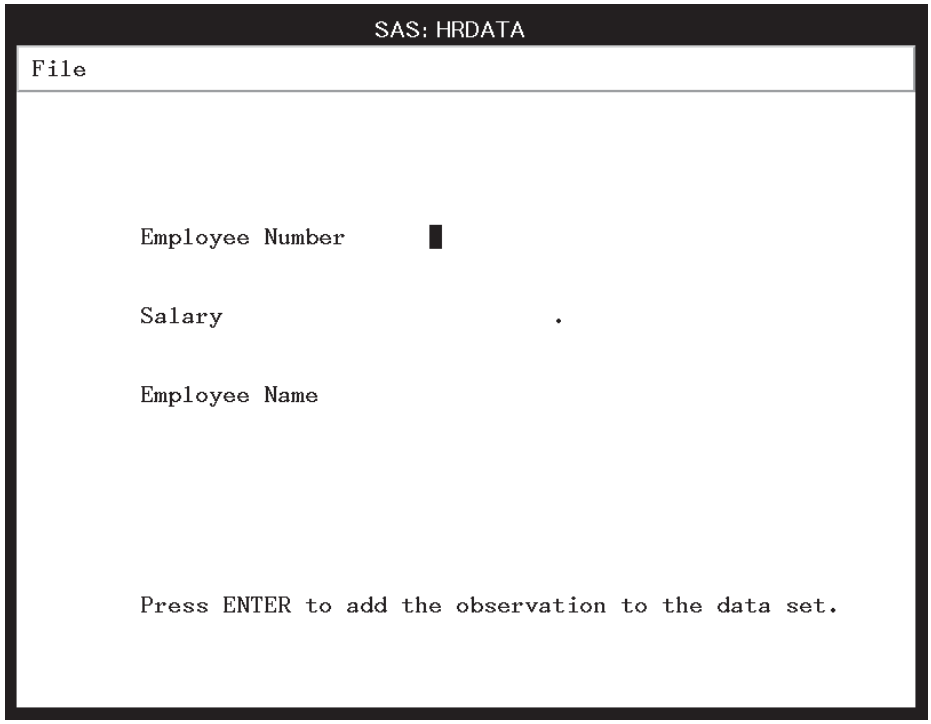
The %INCLUDE statement recalls the statements in the file HRWDW. The statements in HRWDW redisplay the primary window. See the HRSELECT window on page 990

```
filename hrwdw 'external-file';
%include hrwdw;
run;
```

The SELECTION and DIALOG statements in the PROC PMENU step modify the DATA statement in this program so that the correct department name is used when the data set is created. That is, if the user selects **Other Departments** and enters **DEPT05**, then the DATA statement is changed by the command string in the DIALOG statement to

```
data proclib.dept05;
```

The following figure displays the data entry window, HRDATA.



Printing a Program

When the user selects **Print_Report** from the menu bar, a menu is displayed. When the user selects one of the listed departments or chooses to enter a different department, the following statements are invoked. These statements are stored in the external file referenced by the PRT fileref.

PROC PRINTTO routes the output to an external file.

```
   proc printto
file='external-file' new;
   run;

   libname proclib
'SAS-data-library';
```

```
proc print data=proclib.deptxx;
   title 'Information for deptxx';
run;
```

This PROC PRINTTO step restores the default output destination. See Chapter 38, "PRINTTO Procedure," on page 1073

```
proc printto;
run;
```

The %INCLUDE statement recalls the statements in the file HRWDW. The statements in HRWDW redisplay the primary window.

```
filename hrwdw 'external-file';
%include hrwdw;
run;
```

Example 5: Associating Menus with a FRAME Application

Features: ITEM statement
MENU statement

Other features: SAS/AF software

Details

This example creates menus for a FRAME entry and gives the steps necessary to associate the menus with a FRAME entry from SAS/AF software.

Program

```
libname proclib
'SAS-data-library';

proc pmenu catalog=proclib.menucat;

   menu frame;

      item 'File' menu=f;
      item 'Help' menu=h;

      menu f;
         item 'Cancel';
         item 'End';

      menu h;
         item 'About the application' selection=a;
         item 'About the keys'   selection=k;

            selection a 'sethelp proclib.menucat.app.help;help';
            selection k 'sethelp proclib.menucat.keys.help;help';
run;
quit;
```

Program Description

Declare the PROCLIB library. The PROCLIB library is used to store menu definitions.

```
libname proclib
'SAS-data-library';
```

Specify the catalog for storing menu definitions. Menu definitions will be stored in the PROCLIB.MENUCAT catalog.

```
proc pmenu catalog=proclib.menucat;
```

Specify the name of the catalog entry. The MENU statement specifies FRAME as the name of the catalog entry. The menus are stored in the catalog entry PROCLIB.MENUS.FRAME.PMENU.

```
    menu frame;
```

Design the menu bar. The ITEM statements specify the items in the menu bar. The value of MENU= corresponds to a subsequent MENU statement.

```
        item 'File' menu=f;
        item 'Help' menu=h;
```

Design the File menu. The MENU statement equates to the MENU= option in a preceding ITEM statement. The ITEM statements specify the selections that are available under File on the menu bar.

```
        menu f;
           item 'Cancel';
           item 'End';
```

Design the Help menu. The MENU statement equates to the MENU= option in a preceding ITEM statement. The ITEM statements specify the selections that are available under Help on the menu bar. The value of the SELECTION= option equates to a subsequent SELECTION statement.

```
        menu h;
           item 'About the application' selection=a;
           item 'About the keys'   selection=k;
```

Specify commands for the Help menu. The SETHELP command specifies a HELP entry that contains user-written information for this application. The semicolon that appears after the HELP entry name enables the HELP command to be included in the string. The HELP command invokes the HELP entry.

```
            selection a 'sethelp proclib.menucat.app.help;help';
            selection k 'sethelp proclib.menucat.keys.help;help';
run;
quit;
```

Steps to Associate Menus with a FRAME

1. In the BUILD environment for the FRAME entry, from the menu bar, select **View** ⇨ **Properties Window**.
2. In the Properties window, select the **Value** field for the pmenuEntry Attribute Name. The Select An Entry window appears.
3. In the Select An Entry window, enter the name of the catalog entry that is specified in the PROC PMENU step that creates the menus.
4. Test the FRAME as follows from the menu bar of the FRAME:**Build** ⇨ **Test** Notice in the following display that the menus are now associated with the FRAME.

For more information about programming with FRAME entries, see *Getting Started with SAS/AF(R) 9.3 and Frames* .

Chapter 37

PRINT Procedure

Overview: PRINT Procedure

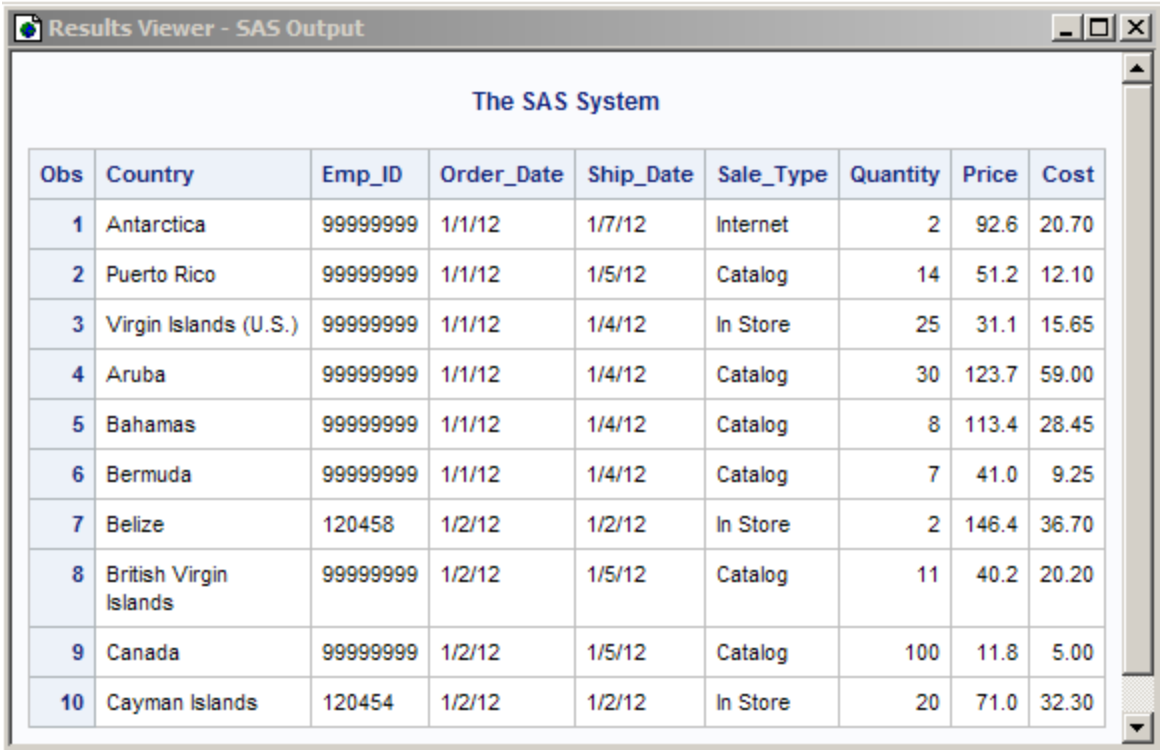
What Does the PRINT Procedure Do?

The PRINT procedure prints the observations in a SAS data set, using all or some of the variables. You can create a variety of reports ranging from a simple listing to a highly customized report that groups the data and calculates totals and subtotals for numeric variables.

Simple Listing Report

The following output illustrates the simplest type of report that you can produce. The statements that produce the output follow. "Example 1: Selecting Variables to Print" on page 1016 creates the data set EXPREV.

```
options obs=10;

proc print data=exprev;
run;
```



Results Viewer - SAS Output

The SAS System

Obs	Country	Emp_ID	Order_Date	Ship_Date	Sale_Type	Quantity	Price	Cost
1	Antarctica	99999999	1/1/12	1/7/12	Internet	2	92.6	20.70
2	Puerto Rico	99999999	1/1/12	1/5/12	Catalog	14	51.2	12.10
3	Virgin Islands (U.S.)	99999999	1/1/12	1/4/12	In Store	25	31.1	15.65
4	Aruba	99999999	1/1/12	1/4/12	Catalog	30	123.7	59.00
5	Bahamas	99999999	1/1/12	1/4/12	Catalog	8	113.4	28.45
6	Bermuda	99999999	1/1/12	1/4/12	Catalog	7	41.0	9.25
7	Belize	120458	1/2/12	1/2/12	In Store	2	146.4	36.70
8	British Virgin Islands	99999999	1/2/12	1/5/12	Catalog	11	40.2	20.20
9	Canada	99999999	1/2/12	1/5/12	Catalog	100	11.8	5.00
10	Cayman Islands	120454	1/2/12	1/2/12	In Store	20	71.0	32.30

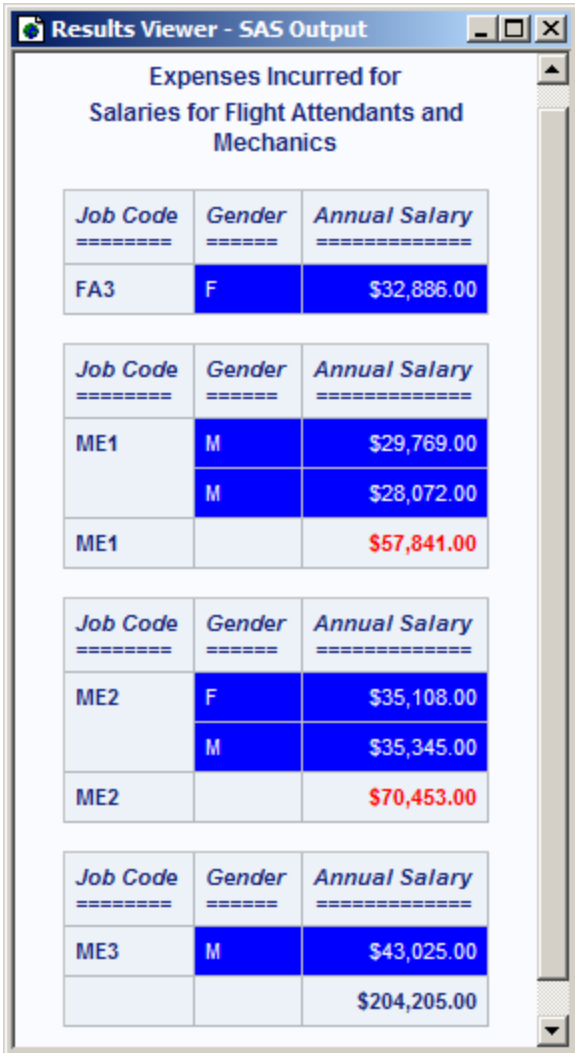
Customized Report

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:

- create HTML output
- customize the appearance of the report
- customize the title and the column headings
- 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

For an explanation of the program that produces this report, see "Program: Creating an HTML Report with the STYLE Option" on page 1066.

Display 37.1 *Customized Report Produced by PROC PRINT Using ODS*



Concepts: PRINT Procedure

Procedure Output

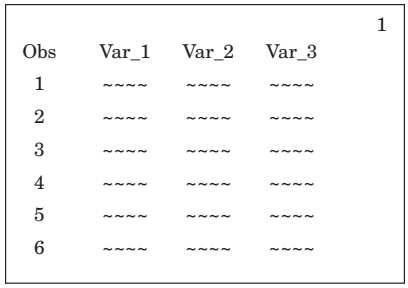
PROC PRINT always produces a printed report. You control the appearance of the report with statements and options. See the PRINT procedure examples starting on page 1016 for a sampling of the types of reports that the procedure produces.

Page Layout

Observations

By default, PROC PRINT uses an identical layout for all observations on a page of output. First, it attempts to print observations on a single line, as shown in the following figure.

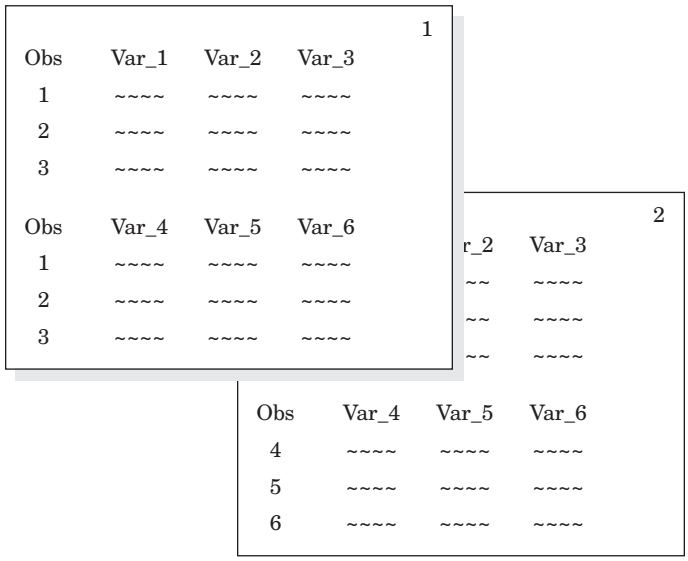
Figure 37.1 *Printing Observations on a Single Line*



				1
Obs	Var_1	Var_2	Var_3	
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 37.2 *Splitting Observations into Multiple Sections on One Page*

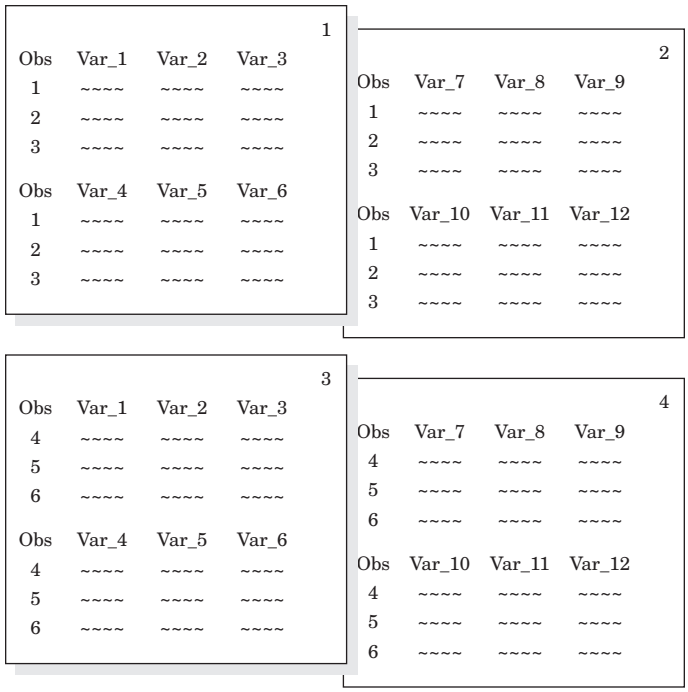


				1
Obs	Var_1	Var_2	Var_3	
1	~~~~	~~~~	~~~~	
2	~~~~	~~~~	~~~~	
3	~~~~	~~~~	~~~~	
Obs	Var_4	Var_5	Var_6	
1	~~~~	~~~~	~~~~	
2	~~~~	~~~~	~~~~	
3	~~~~	~~~~	~~~~	

				2
		r_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 37.3 *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	~~~~	~~~~	~~~~

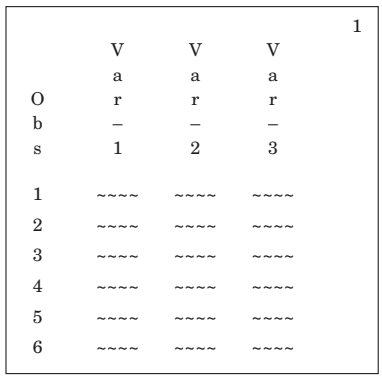
Note: You can alter the page layout with the ROWS= option in the PROC PRINT statement. (See the discussion of ROWS= option on page 1005.)

Note: PROC PRINT might produce slightly different output if the data set is not RADIX addressable. Version 6 compressed files are not RADIX addressable, while, beginning with Version 7, compressed files are RADIX addressable. (The integrity of the data is not compromised; the procedure simply numbers the observations differently.)

Column Headings

By default, spacing specifies whether PROC PRINT prints column headings horizontally or vertically. Figure 37.1 on page 998 , Figure 37.2 on page 998, and Figure 37.3 on page 999 all illustrate horizontal headings. The following figure illustrates vertical headings.

Figure 37.4 *Using Vertical Headings*



1

Obs	Var_1	Var_2	Var_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. (The WIDTH= option overrides this default behavior.) 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.

WIDTH= controls the column width.

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=.

Syntax: PRINT Procedure

Tips: Supports the Output Delivery System. For details, see Chapter 3, “Output Delivery System: Basic Concepts,” in *SAS Output Delivery System: User's Guide*.

You can use the ATTRIB, FORMAT, LABEL, and WHERE statements. See *SAS Statements: Reference* For more information, see “Statements with the Same Function in Multiple Procedures” on page 35. You can also use any global statement. For a list, see “Global Statements” in Chapter 1 of *SAS Statements: Reference*.

PROC PRINT <*option(s)*>;
BY <DESCENDING> *variable-1* <…<DESCENDING>*variable-n*> <NOTSORTED>;
PAGEBY *BY-variable*;
SUMBY *BY-variable*;
ID *variable(s)* <*option*>;
SUM *variable(s)* <*option*>;
VAR *variable(s)* <*option*>;

Statement	Task	Example
“PROC PRINT Statement”	Print observations in a data set	Ex. 1, Ex. 2, Ex. 3, Ex. 5
“BY Statement”	Produce a separate section of the report for each BY group	Ex. 3, Ex. 4, Ex. 5, Ex. 6, Ex. 8

Statement	Task	Example
"ID Statement"	Identify observations by the formatted values of the variables that you list instead of by observation numbers	Ex. 7
"PAGEBY Statement"	Control page ejects that occur before a page is full	Ex. 3
"SUMBY Statement"	Limit the number of sums that appear in the report	Ex. 4, Ex. 5, Ex. 6, Ex. 8
"SUM Statement"	Total values of numeric variables	Ex. 6
"VAR Statement"	Select variables that appear in the report and determine their order	Ex. 1, Ex. 2, Ex. 8

PROC PRINT Statement

Prints observations in a SAS data set using some or all of the variables.

Syntax

PROC PRINT <*option(s)*>;

Summary of Optional Arguments

CONTENTS=*link-text*
: specifies text for the links in the HTML contents file.

DATA=*SAS-data-set*
: specifies the SAS data set to print.

Control column format

HEADING=*direction*
: controls the orientation of the column headings.

LABEL
: specifies to use the variables' labels as column headings.

SPLIT='*split-character*'
: specifies the split character, which controls line breaks in column headings.

STYLE <(*locations(s)*)>=<*style-element-name*> <[*style-attribute-specification(s)*]>
: specify one or more style elements for the Output Delivery System to use for different parts of the report.

SUMLABEL
: displays the BY variable label on the summary line in place of the BY variable name.

Control general format

BLANKLINE=*n*

BLANKLINE=(COUNT=*n* <STYLE=[*style-attibute-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
suppress 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=*column-width*
determines the column width for each variable.

Optional Arguments

BLANKLINE=*n*
BLANKLINE=(COUNT=*n* <STYLE=[*style-attibute-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 to use for the blank line.

Default: DATA

Tip: You can use the BACKGROUNDCOLOR style element to make a visual distinction between observations using color.

See: The STYLE= option on page 1006 for valid style attributes.

Example: "Example 1: Selecting Variables to Print" on page 1016

CONTENTS=*link-text*
specifies the text for the links in the HTML contents file to the output produced by the PROC PRINT statement.

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 HTML 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 20

DOUBLE
writes a blank line between observations.

Alias: D

Restriction: DOUBLE is valid only for the ODS LISTING destination.

Example: "Example 1: Selecting Variables to Print" on page 1016

HEADING=*direction*
controls the orientation of the column headings, where *direction* is one of the following:

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 circumstances:

if you omit the LABEL option in the PROC PRINT statement, even if the PROC PRINT step contains a LABEL statement

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 "Statements with the Same Function in Multiple Procedures" on page 35.

For information about using the LABEL statement in a DATA step to create permanent labels, see "LABEL Statement" in *SAS Statements: Reference*.

Example: "Example 3: Creating Separate Sections of a Report for Groups of Observations" on page 1026

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 *string-1*.
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 *string-1*.
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 *string-1*; the number for the entire data set is labeled with *string-2*.

Examples:

"Example 2: Customizing Text in Column Headings" on page 1021

"Example 3: Creating Separate Sections of a Report for Groups of Observations" on page 1026

"Example 4: Summing Numeric Variables with One BY Group" on page 1036

NOOBS

suppress the column in the output that identifies each observation by number.

Example: "Example 3: Creating Separate Sections of a Report for Groups of Observations" on page 1026

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 1006.)

Example: "Example 2: Customizing Text in Column Headings" on page 1021

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 37.5 *Three Methods of Summing Variables*

Actual Values	PROC PRINT without the ROUND option		PROC PRINT with the ROUND 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	0.0160000
.009	3	0.01	3	0.01	
.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 " on page 998 for discussion of the default layout.

Example: "Example 7: Controlling the Layout of a Report with Many Variables" on page 1056

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 1004.)

Note: PROC PRINT does not split labels of BY variables in the heading preceding each BY group even if you specify SPLIT=. Instead, PROC PRINT replaces the split character with a blank.

Example: "Example 2: Customizing Text in Column Headings" on page 1021

STYLE <(*locations(s)*)>=<*style-element-name*> <[*style-attribute-specification(s)*]>
specify one or more style elements for the Output Delivery System to use for different parts of the report. You can use braces ({ and }) instead of square brackets ([and]).

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 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 37.1 *Specifying Locations in the STYLE Option*

Location	Affected Report Part	Can Also Be Used in These Statements
BYLABEL	Label for the BY variable on the line containing the SUM totals	None
DATA	Cells of all columns	VAR ID SUM
GRANDTOTAL	SUM line containing the grand totals for the whole report	SUM
HEADER	All column headings	VAR ID SUM
N	N= table and contents	None
OBS	Data in the OBS column	None

Location	Affected Report Part	Can Also Be Used in These Statements
OBSHEADER	Header of the OBS column	None
TABLE	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	SUM line containing totals for each BY group	SUM

For your convenience and for consistency with other procedures, the following table shows aliases for the different locations.

Table 37.2 *Aliases for Locations*

Location	Aliases
BYLABEL	BYSUMLABEL BYLBL BYSUMLBL
DATA	COLUMN COL
GRANDTOTAL	GRANDTOT GRAND GTOTAL GTOT
HEADER	HEAD HDR
N	none
OBS	OBSDATA OBSCOLUMN OBSCOL
OBSHEADER	OBSHEAD OBSHDR
TABLE	REPORT

Location	Aliases
TOTAL	TOT BYSUMLINE BYLINE BYSUM

Note: Style specifications in a statement other than the PROC PRINT statement override the same style specification in the PROC PRINT statement. However, style attributes that you specify in the PROC PRINT statement are inherited, provided that you do not override the style with style specifications in another statement. For example, if you specify a blue background and a white foreground for all column headings in the PROC PRINT statement, and you specify a gray background for the column headings of a variable in the VAR statement, the background for that particular column heading is gray, and the foreground is white (as specified in the PROC PRINT statement).

style-element-name
is the name of a style element that is part of a style definition that is registered with the Output Delivery System. SAS provides some style definitions. Users can create their own style definitions with the TEMPLATE procedure. See *SAS Output Delivery System: User's Guide*.

When style elements are processed, more specific style elements override less specific style elements. The following table shows the default style element for each location.

Table 37.3 *The Default Style Element for Each Location in PROC PRINT*

Location	Default Style Element
BYLABEL	Header
DATA	Data (for all but ID statement) RowHeader (for ID statement)
GRANDTOTAL	Header
HEADER	Header
N	NoteContent
OBS	RowHeader
OBSHEADER	Header
TABLE	Table
TOTAL	Header

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= *
BACKGROUNDIMAGE=	COLOR= *
BORDERCOLOR=	FRAME=
BORDERCOLORDARK=	HTMLCLASS=
BORDERCOLORLIGHT=	TEXTALIGN=
BORDERWIDTH=	OUTPUTWIDTH=
CELLPADDING=	POSTHTML=
CELLSPACING=	POSTIMAGE=
FONT= *	POSTTEXT=
FONTFAMILY= *	PREHTML=
FONTSIZE= *	PREIMAGE=
FONTSTYLE= *	PRETEXT=
FONTWEIGHT= *	RULES=

* 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=

Restrictions:
This option affects all destinations except Listing and Output.

STYLE= is not valid for the ODS LISTING destination

See: For information about style attributes, see DEFINE Style Statement in *SAS Output Delivery System: User's Guide*.

SUMLABEL
displays the BY variable label on the summary line in place of the BY variable name.

Default: If you omit SUMLABEL, PROC PRINT uses the BY variable names in the summary line.

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*.

Examples:
"Example 4: Summing Numeric Variables with One BY Group" on page 1036

"Example 5: Summing Numeric Variables with Multiple BY Variables" on page 1041

UNIFORM
See WIDTH=UNIFORM on page 1010 .

WIDTH=*column-width*
determines the column width for each variable. The value of *column-width* must be one of the following:

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 1000.

BY Statement

Produces a separate section of the report for each BY group.

See: Chapter 3, "Statements with the Same Function in Multiple Procedures," on page 35

Examples: "Example 3: Creating Separate Sections of a Report for Groups of Observations" on page 1026

"Example 4: Summing Numeric Variables with One BY Group" on page 1036

"Example 5: Summing Numeric Variables with Multiple BY Variables" on page 1041

"Example 6: Limiting the Number of Sums in a Report" on page 1051

"Example 8: Creating a Customized Layout with BY Groups and ID Variables" on page 1061

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, the observations in the data set must either 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 data set is 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 data is 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.

Details

Using 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 8: Creating a Customized Layout with BY Groups and ID Variables" on page 1061.)

Using 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.

Using 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.

ID Statement

Identifies observations by using the formatted values of the variables that you list instead of by using observation numbers.

Examples: "Example 7: Controlling the Layout of a Report with Many Variables" on page 1056

"Example 8: Creating a Customized Layout with BY Groups and ID Variables" on page 1061

Syntax

ID *variable(s)*
</ STYLE <(*location(s)*)>=<*style-element-name*> <[*style-attribute-specification(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-element-name*> <[*style-attribute-specification(s)*]>
specifies the style element to use for ID columns created with the ID statement.

Tip: To specify different style elements 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 1002 option in the PROC PRINT statement.

Details

Using 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 8: Creating a Customized Layout with BY Groups and ID Variables" on page 1061.)

PAGEBY Statement

Controls page ejects that occur before a page is full.

Requirement: BY statement

Example: "Example 3: Creating Separate Sections of a Report for Groups of Observations" on page 1026

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 21.)

SUM Statement

Totals values of numeric variables.

Examples: "Example 4: Summing Numeric Variables with One BY Group" on page 1036
"Example 5: Summing Numeric Variables with Multiple BY Variables" on page 1041
"Example 6: Limiting the Number of Sums in a Report" on page 1051
"Example 8: Creating a Customized Layout with BY Groups and ID Variables" on page 1061

Syntax

SUM *variable(s)*
</ STYLE <(*location(s)*)>=<*style-element-name*> <[*style-attribute-specification(s)*]>>;

Required Argument

variable(s)
identifies the numeric variables to total in the report.

Optional Argument

STYLE <(*location(s)*)>=<*style-element-name*> <[*style-attribute-specification(s)*]>
specifies the style element to use for cells containing sums that are created with the SUM statement.

Tips:
To specify different style elements 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 will be used.

See: For information about the arguments of this option and how it is used, see the option STYLE= on page 1002 in the PROC PRINT statement.

Details

Using 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 4: Summing Numeric Variables with One BY Group" on page 1036.)

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 5: Summing Numeric Variables with Multiple BY Variables" on page 1041.)

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 6: Limiting the Number of Sums in a Report" on page 1051

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.

Tip: If you omit the VAR statement, PROC PRINT prints all variables in the data set.

Examples: "Example 1: Selecting Variables to Print" on page 1016

"Example 8: Creating a Customized Layout with BY Groups and ID Variables" on page 1061

Syntax

VAR *variable(s)*
</ STYLE <(*location(s)*=<*style-element-name*> <[*style-attribute-specification(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-element-name*> <[*style-attribute-specification(s)*]>
specifies the style element to use for all columns that are created by a VAR statement.

Tip: To specify different style elements 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 1002 in the PROC PRINT statement.

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 up until the error. SAS writes a message to the SAS log and ends the PRINT procedure.

For LISTING output, if the page size is set too small, SAS cannot print both the data and any titles or footnotes on the same page. If this happens, only the data is printed to the LISTING destination and SAS writes a warning message to the log. To write both the data and titles or footnotes on the same page, make sure that the page size is adequate.

Examples: PRINT Procedure

Example 1: Selecting Variables to Print

Features: PROC PRINT statement options:
BLANKLINE
DOUBLE
STYLE

VAR statement

Other features: DATA step
FOOTNOTE statement
ODS HTML statement
OPTIONS statement
TITLE statement

Data set: EXPREV

Details

This example demonstrates the following tasks:

- selects three variables for the reports
- uses variable labels as column headings
- double spaces between rows of the report in the Listing output
- creates a default HTML report
- creates a stylized HTML report

Program: Creating an HTML 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

HTML is the default destination when SAS opens.

Set the OBS= system option to process 10 observations.

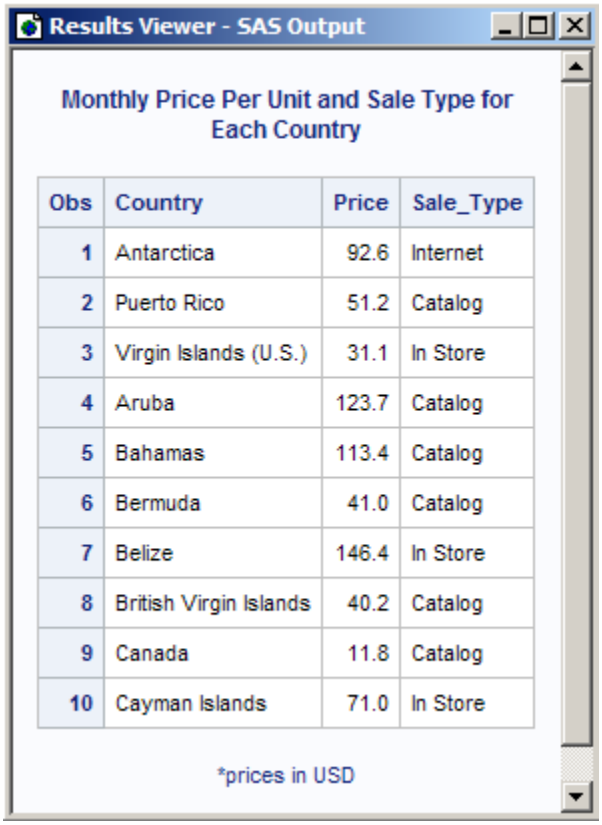
```
options obs=10;
```

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: HTML

Output 37.1 *Selecting Variables: Default HTML Output*



Results Viewer - SAS 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: Creating an HTML Report with the STYLE and BLANKLINE Options

```
options nodate obs=5;

ods html file='your_file_styles.html';


proc print data=exprev
   style(header) = {fontstyle=italic color= green}
   style(obs) = {backgroundcolor=#5959ab color=white}
   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 HTML output. The following example uses the STYLE option to add shading and spacing to your HTML report.

```
options nodate obs=5;

ods html file='your_file_styles.html';
```

Create stylized HTML output. The first STYLE option specifies that the column headings be written in green italic font. The second STYLE option specifies that ODS change the color of the background of the observations column to the RGB color code 5959AB. The BLANKLINE option specifies to add a blank line between each observation and use a background color of the RGB color code 456789. Because a style has not been defined for the obsheader style element, 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=#5959ab color=white}
   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: HTML Output with Styles

Output 37.2 *Selecting Variables: HTML Output Using Styles*

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

Program: Creating a Listing Report

```
options nodate pageno=1 linesize=80 pagesize=30 obs=10;
```

```
ods html close;
ods listing;

proc print data=exprev double;

   var country price sale_type;

   title 'Monthly Price Per Unit and Sale Type for Each Country';
   footnote '*prices in USD';
run;

ods listing close;
ods html;
```

Program Description

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 display.

```
options nodate pageno=1 linesize=80 pagesize=30 obs=10;
```

Close the HTML destination and open the Listing destination. HTML is the default destination when you start SAS. To create only a listing report, you can close the HTML destination and open the Listing destination.

```
ods html close;
ods listing;
```

Print the data set EXPREV. EXPREV contains information about a company's product order type and price per unit for two months. DOUBLE inserts a blank line between observations. The DOUBLE option has no effect on the HTML output.

```
proc print data=exprev double;
```

Select the variables to include in the report. The VAR statement creates columns for Country, Price, and Sale_Type, in that order.

```
   var country price sale_type;
```

Specify a title and a footnote. The TITLE statement specifies the title for the report. The FOOTNOTE statement specifies a footnote for the report.

```
   title 'Monthly Price Per Unit and Sale Type for Each Country';
   footnote '*prices in USD';
run;
```

Close the Listing destination and reopen the HTML destination. When you close and reopen the HTML destination, SAS saves HTML output to the current directory and not the Work library.

```
ods listing close;
ods html;
```

Output: Listing

By default, PROC PRINT identifies each observation by number under the column heading **Obs.**

Output 37.3 *Selecting Variables: Listing Output*

Monthly Price Per Unit and Sale Type for Each Country 1

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

Example 2: Customizing Text in Column Headings

Features: PROC PRINT statement options:
- N
- OBS=
- SPLIT=
- STYLE

VAR statement option:
- STYLE

Other features: LABEL statement
ODS PDF statement
FORMAT statement
TITLE statement

Data set: EXPREV

Details

This example demonstrates the following tasks:

- customizes and underlines the text in column headings for variables

- 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.
- creates a Listing report
- creates a default PDF report
- creates a stylized PDF report

Program: Creating a Listing Report

```
options nodate pageno=1 linesize=80 pagesize=30 obs=10;

ods html close;
ods listing;

proc print data=exprev split='*' 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 listing close;
ods html;
```

Program Description

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;
```

Close the HTML destination and open the Listing destination. By default, the HTML destination is open.

```
ods html close;
ods listing;
```

Print the report and define the column headings. SPLIT= identifies the asterisk as the character that starts a new line in column headings. 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. The split character (*) starts a new line in the column heading. The equal signs (=) in the value of OBS= underlines the column heading.

```
proc print data=exprev split='*' n
obs='Observation*Number*===========';
```

Select the variables to include in the report. The VAR statement creates columns for Country, Sale_Type, and Price, in that order.

```
var country sale_type 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. When you use the SPLIT= option in the PROC PRINT statement, the procedure uses labels for column headings. The split character (*) starts a new line in the column heading. The equal signs (=) in the labels underlines the column headings.

```
label country='Country Name**============'
      sale_type='Order Type**==========='
      price='Price Per Unit*in USD*==============';
```

Specify a title for the report, and format any variable containing numbers. The FORMAT statement assigns the DOLLAR10.2 format to the variable Price in the report. The TITLE statement specifies a title.

```
   format price dollar10.2;
   title 'Order Type and Price Per Unit in Each Country';
run;
```

Close the Listing destination and re-open the HTML destination.

```
ods listing close;
ods html;
```

Output: Listing

Output 37.4 *Customizing Column Headings: Listing Output*

Order Type and Price Per Unit in Each Country 1

Observation Number	Country Name	Order Type	Price Per Unit in USD
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

```
options nodate pageno=1 linesize=80 pagesize=40 obs=10;

ods pdf file='your_file.pdf';
```

```
proc print data=exprev split='*' 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. In the following example, ODS statements were added to produce PDF output.

```
options nodate pageno=1 linesize=80 pagesize=40 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.

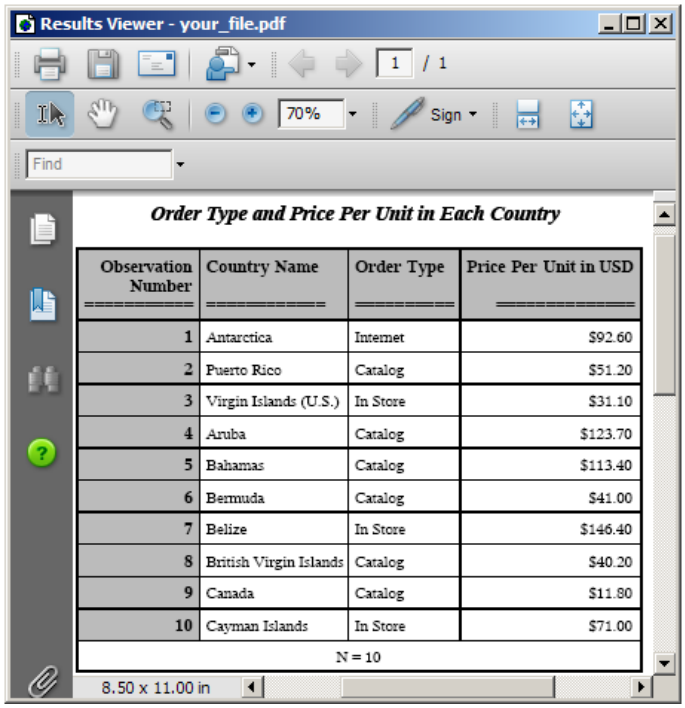
```
ods pdf file='your_file.pdf';

proc print data=exprev split='*' 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;
```

Close the PDF destination. The ODS PDF CLOSE statement closes the PDF destination.

```
ods pdf close;
```

Output: PDF

Output 37.5 *Customizing Column Heading: Default PDF Output*



Observation Number	Country Name	Order Type	Price Per Unit in USD
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 nodate pageno=1 linesize=80 pagesize=40 obs=10;

ods pdf file='your_file.pdf';

proc print data=exprev split='*' n obs='Observation*Number*==========='
     style(n) = {fontstyle=italic backgrouncolor= blue}
     style(header obs obsheader) = {backgrouncolor=yellow color=blue};

   var country sale_type price / style (data)= [ background = gray ];
   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

```
options nodate pageno=1 linesize=80 pagesize=40 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 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 gray.

```
proc print data=exprev split='*' n obs='Observation*Number*==========='
     style(n) = {fontstyle=italic backgrouncolor= blue}
     style(header obs obsheader) = {backgrouncolor=yellow color=blue};
```

Create stylized PDF output. The STYLE option changes the color of the cells containing data to gray.

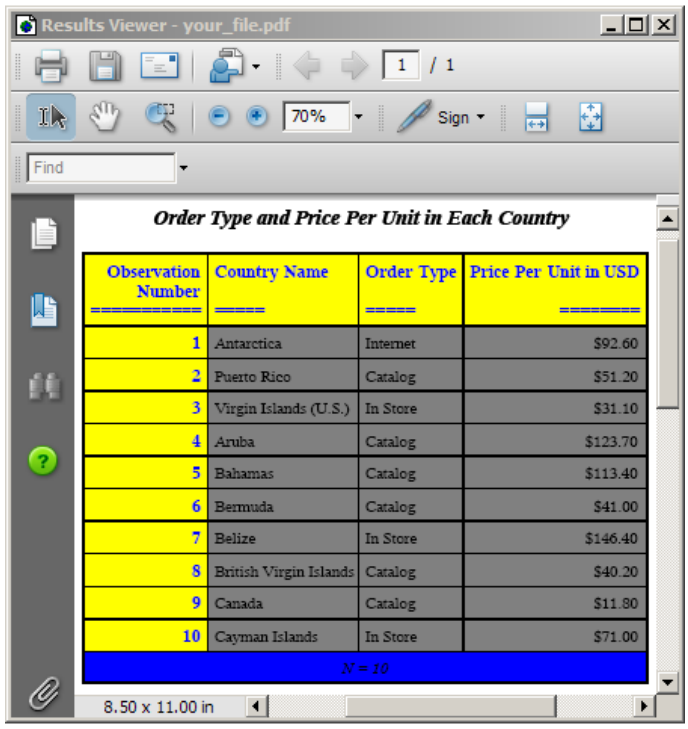
```
   var country sale_type price / style (data)= [ background = gray ];
   label country='Country Name**======'
         sale_type='Order Type**====='
         price='Price Per Unit in USD**========';
   format price dollar10.2;
run;

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 37.6 *Customizing Column Headings: PDF Using Styles*



Observation Number	Country Name	Order Type	Price Per Unit in USD
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 3: Creating Separate Sections of a Report for Groups of Observations

Features: PROC PRINT statement options:
LABEL
N=

NOOBS
STYLE

BY statement

PAGEBY statement

Other features: SORT procedure

FORMAT statement

LABEL statement

ODS RTF statement

TITLE statement

Data set: EXPREV

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 HTML report
- creates default and stylized RTF reports

Program: Creating an HTML Report

```
options nodate pageno=1 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 HTML destination is open by default. No ODS HTML statement is needed.

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 OBS= option specifies the number of observations to be displayed.

```
options nodate pageno=1 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 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.

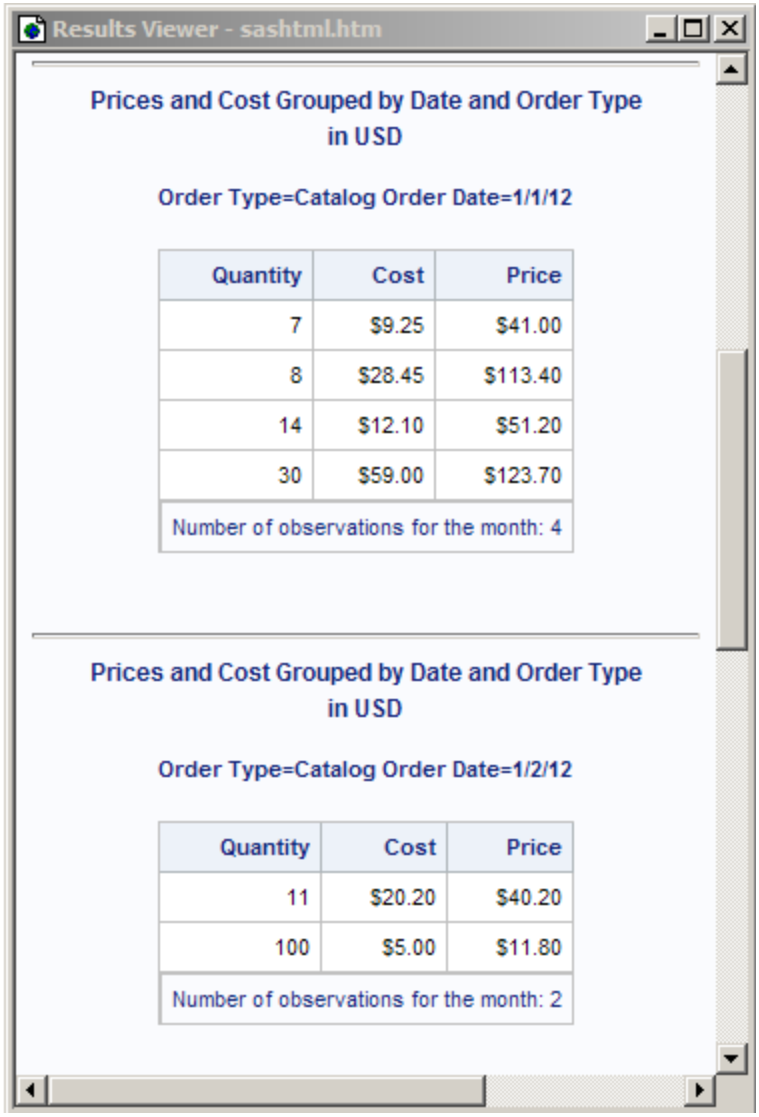
```
   label sale_type='Order Type' order_date='Order Date';
```

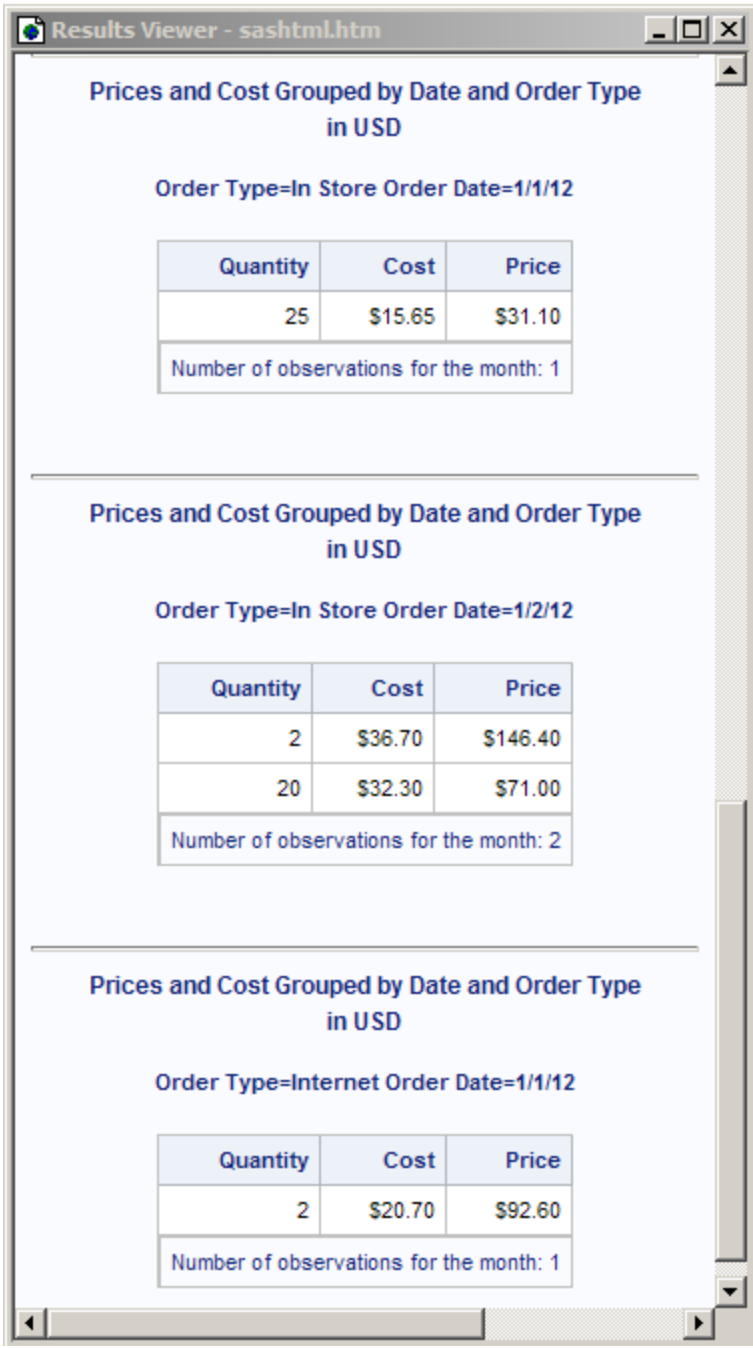
Format the columns that contain numbers and specify a title and footnote. The FORMAT statement assigns a format to Price and Cost for this report. The TITLE statement specifies a title. The TITLE2 statement specifies a second title.

```
   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: HTML

Output 37.7 *Creating Separate Sections of a Report for Groups of Observations: HTML Output*





Program: Creating an RTF Report

```
options nodate pageno=1 linesize=80 pagesize=40 obs=10;

ods rtf file='your_file.rtf' 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;

ods rtf close;
```

Program Description

```
options nodate pageno=1 linesize=80 pagesize=40 obs=10;
```

Create output for Microsoft Word and specify the file to store the output in. The ODS RTF statement opens the RTF destination and creates output formatted for Microsoft Word. The FILE= option specifies the external file that contains the RTF output. The STARTPAGE=NO option specifies that no new pages be inserted explicitly at the start of each by group.

```
ods rtf file='your_file.rtf' 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 RTF destination. The ODS RTF CLOSE statement closes the RTF destination.

```
ods rtf close;
```

Output: RTF

Output 37.8 *Creating Separate Sections of a Report for Groups of Observations: Default RTF Output*

Price and Cost Grouped by Date and Order Type in USD

Order Type=Catalog Order Date=1/1/12

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=1/2/12

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=1/1/12

Quantity	Cost	Price
25	$15.65	$31.10
Number of observations for each order type:1		

Order Type=In Store Order Date=1/2/12

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=1/1/12

Quantity	Cost	Price
2	$20.70	$92.60
Number of observations for each order type:1		

Program: Creating an RTF Report with the STYLE Option

```
options nodate pageno=1 linesize=80 pagesize=40 obs=10;

ods rtf file='your_file.rtf' startpage=no;

proc sort data=exprev;
   by sale_type order_date quantity;
run;

proc print data=exprev n='Number of observations for the month: '
           noobs label style(N) = {background = gray};
   var quantity / style(header) = [background = white];
   var cost / style(header) = [background = blue foreground =
white];
   var price / style(header) = [background = gray];
   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 rtf close;
```

Program Description

```
options nodate pageno=1 linesize=80 pagesize=40 obs=10;

ods rtf file='your_file.rtf' startpage=no;

proc sort data=exprev;
   by sale_type order_date quantity;
run;
```

Create a stylized RTF report. The first STYLE option specifies that the background color of the cell containing the number of observations be changed to gray. The second STYLE option specifies that the background color of the column heading for the variable Quantity be changed to white. The third STYLE option specifies that the background color of the column heading for the variable Cost be changed to 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 Sale_Type be changed to gray.

```
proc print data=exprev n='Number of observations for the month: '
           noobs label style(N) = {background = gray};
   var quantity / style(header) = [background = white];
   var cost / style(header) = [background = blue foreground =
white];
   var price / style(header) = [background = gray];
   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 rtf close;
```

Output: RTF with Styles

Output 37.9 *Creating Separate Sections of a Report for Groups of Observations: RTF Output Using Styles*

Prices and Cost Grouped by Date and Order Type
**prices in USD*

Order Type=Catalog Order Date=1/1/12

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=1/2/12

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=1/1/12

Quantity	Cost	Price
25	$15.65	$31.10
Number of observations for the month: 1		

Order Type=In Store Order Date=1/2/12

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=1/1/12

Quantity	Cost	Price
2	$20.70	$92.60
Number of observations for the month: 1		

Example 4: Summing Numeric Variables with One BY Group

Features: PROC PRINT statement options:
- N=
- SUMLABEL

BY statement

SUM statement

Other features: ODS CSVALL statement

SORT procedure

TITLE statement
- #BYVAL specification

SAS system options:
- BYLINE
- NOBYLINE

Data set: EXPREV

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 HTML file.
- creates a CSV file.

Program: Creating an HTML Report

```
options nodate pageno=1 linesize=80 pagesize=40 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 HTML destination is open by default. This program uses the default filename for the HTML output. No ODS HTML 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 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 nodate pageno=1 linesize=80 pagesize=40 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 DOLLAR10.2. 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.

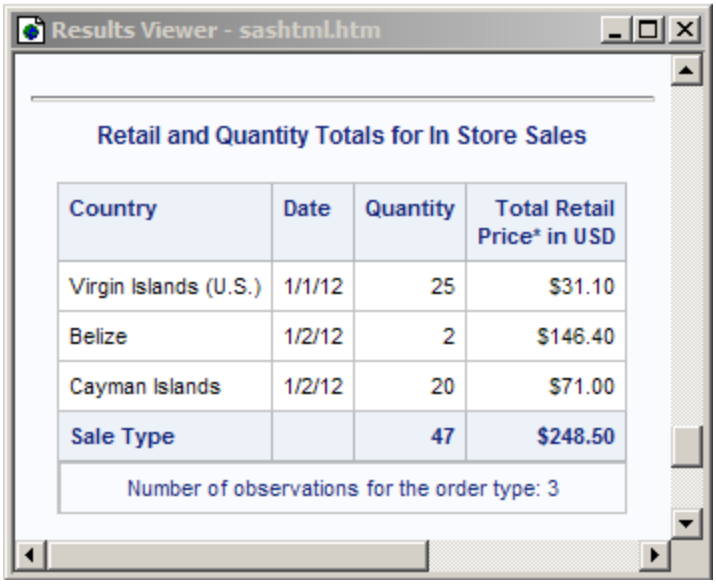
```
    title 'Retail and Quantity Totals for #byval(sale_type) Sales';
run;
```

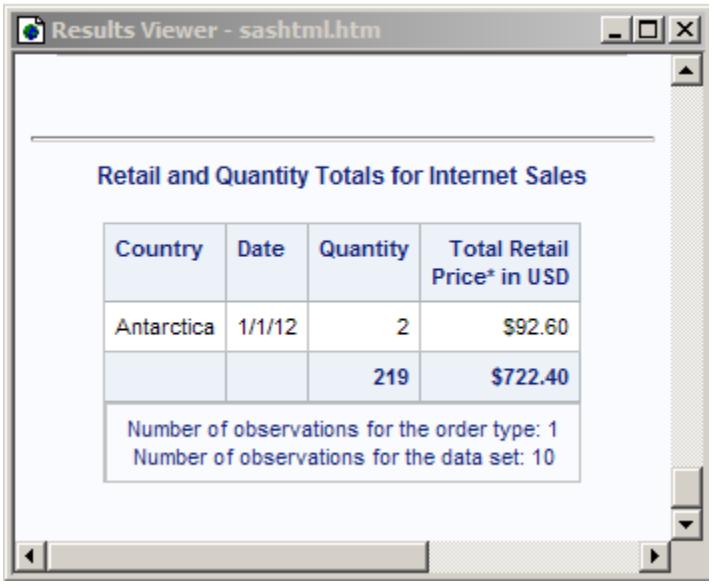
Generate the default BY line. The SAS system option BYLINE resets the printing of the default BY line.

```
options byline;
```

Output: HTML

Output 37.10 *Summing Numeric Variables with One BY Group HTML Output*





Program: Creating a CSV File

```
options nodate pageno=1 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 nodate pageno=1 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 bylines. The FILE= argument specifies the external file that contains the CSV output.

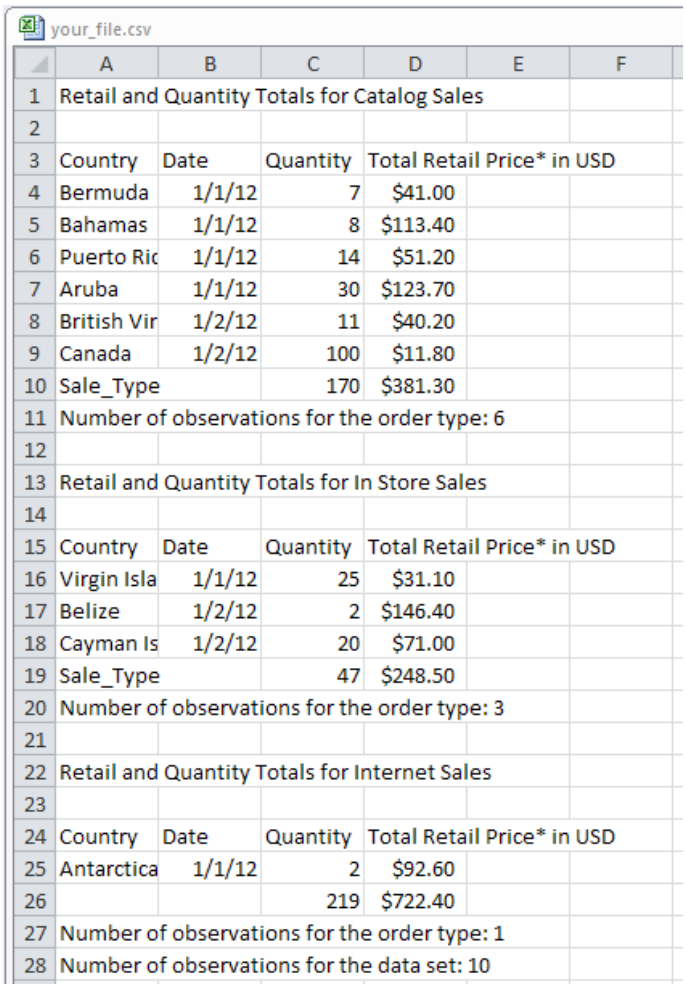
```
ods csvall file='your_file.csv';

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 37.11 *Summing Numeric Variables with One BY Group: CSV Output Viewed with Microsoft Excel*



your_file.csv

	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	1/1/12	7	$41.00		
5	Bahamas	1/1/12	8	$113.40		
6	Puerto Ric	1/1/12	14	$51.20		
7	Aruba	1/1/12	30	$123.70		
8	British Vir	1/2/12	11	$40.20		
9	Canada	1/2/12	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 Isla	1/1/12	25	$31.10		
17	Belize	1/2/12	2	$146.40		
18	Cayman Is	1/2/12	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	1/1/12	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					

Example 5: Summing Numeric Variables with Multiple BY Variables

Features: PROC PRINT statement options:
- N=
- NOOBS
- STYLE
- SUMLABEL

BY statement

SUM statement

Other features: ODS HTML statement

LABEL statement

FORMAT statement

SORT procedure

TITLE statement

Data set: EXPREV

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 the BY group label in place of the BY group variable name on the summary line
- creates a default HTML report
- creates a stylized HTML report

Program: Creating an HTML Report

```
options nodate pageno=1 obs=10;

ods html file='your_file.html';

proc sort data=exprev;
   by sale_type order_date;
run;

proc print data=exprev n noobs sumlabel;
   by sale_type order_date;
   sum price quantity;

   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 nodate pageno=1 obs=10;
```

Produce HTML output and specify the file to store the output in. The HTML destination is open by default. The ODS HTML FILE= statement creates a file that contains HTML output. The FILE= argument specifies the external file that contains the HTML output.

```
ods html file='your_file.html';

proc sort data=exprev;
   by sale_type order_date;
run;

proc print data=exprev n noobs sumlabel;
   by sale_type order_date;
   sum price quantity;

   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: HTML

Output 37.12 *Summing Numeric Variables with Multiple BY Variables: In Store Sales: Default HTML Output*

Results Viewer - SAS Output

Retail and Quantity Totals for Each Sale Date and Sale Type

Sale Type=Catalog Sale Date=1/1/12

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Bermuda	99999999	1/4/12	7	$41.00	$9.25
Bahamas	99999999	1/4/12	8	$113.40	$28.45
Puerto Rico	99999999	1/5/12	14	$51.20	$12.10
Aruba	99999999	1/4/12	30	$123.70	$59.00
Sale Date			**59**	**$329.30**	
N = 4					

Sale Type=Catalog Sale Date=1/2/12

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
British Virgin Islands	99999999	1/5/12	11	$40.20	$20.20
Canada	99999999	1/5/12	100	$11.80	$5.00
Sale Date			**111**	**$52.00**	
Sale Type			**170**	**$381.30**	
N = 2					

Program: Creating an HTML Report with the STYLE Option

```
options nodate pageno=1 obs=10

ods html file='your_file.html';

proc sort data=exprev;
   by sale_type order_date;
run;

proc print data=exprev n noobs sumlabel;

   by sale_type order_date;
sum price / style(GRANDTOTAL) = [background =white color=blue];
sum quantity / style(TOTAL) = [background =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;

ods html close;
```

Program Description

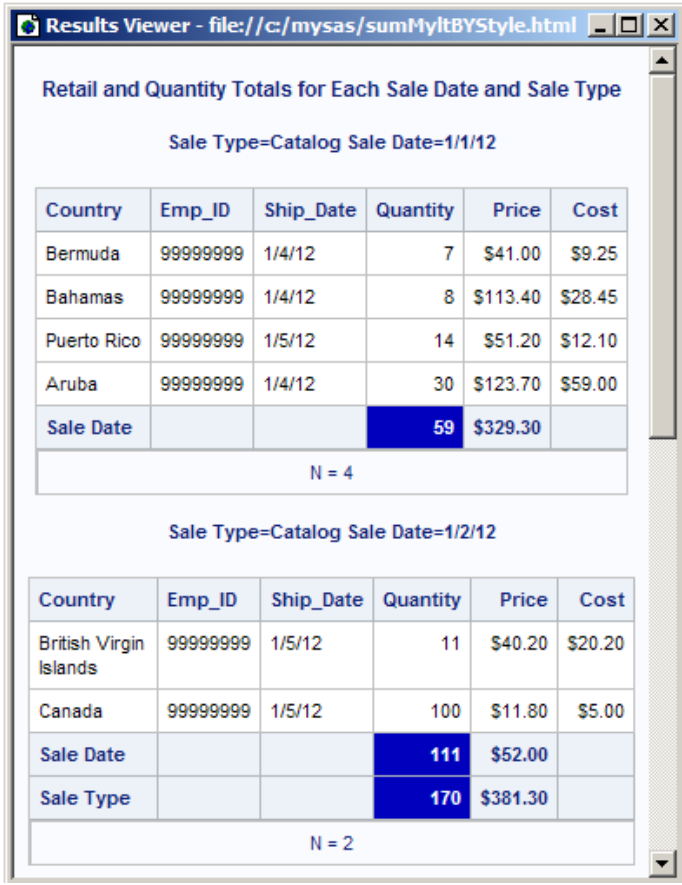
```
options nodate pageno=1 obs=10

ods html file='your_file.html';

proc sort data=exprev;
   by sale_type order_date;
run;

proc print data=exprev n noobs sumlabel;
```

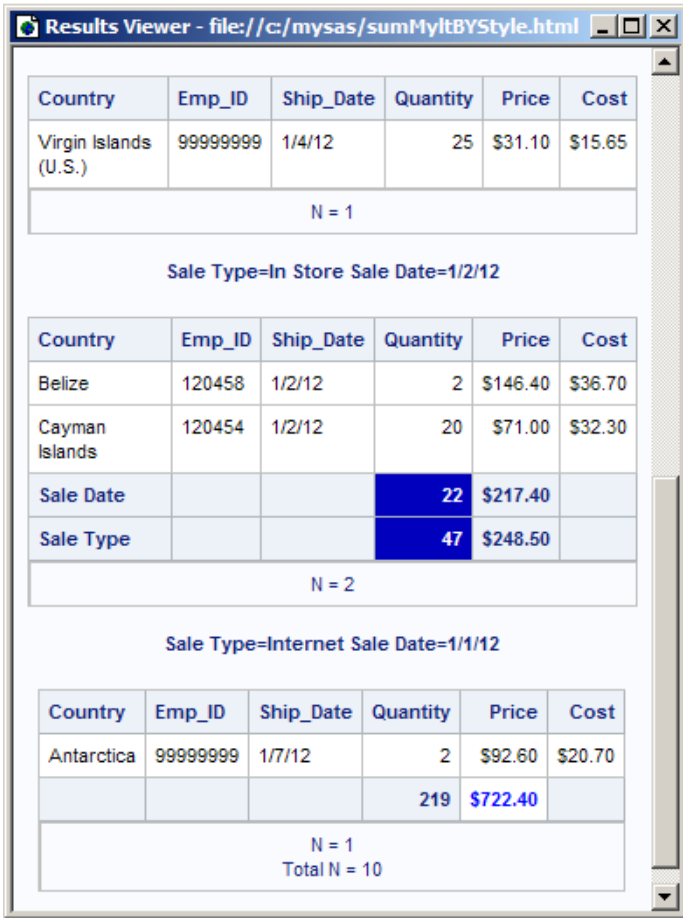
Create stylized HTML output. The STYLE option in the first SUM statement specifies that the background color of the cell containing the grand total for the variable Price be changed to white and the font color be changed to blue. The STYLE option in the second SUM statement specifies that the background 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) = [background =white color=blue];
sum quantity / style(TOTAL) = [background =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;

ods html close;
```

Output: HTML with Styles

Output 37.13 *Summing Numeric Variables with Multiple BY Variables: Catalog Sales: HTML Output Using Styles*





Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Virgin Islands (U.S.)	99999999	1/4/12	25	$31.10	$15.65
N = 1					

Sale Type=In Store Sale Date=1/2/12

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Belize	120458	1/2/12	2	$146.40	$36.70
Cayman Islands	120454	1/2/12	20	$71.00	$32.30
Sale Date			22	$217.40	
Sale Type			47	$248.50	
N = 2					

Sale Type=Internet Sale Date=1/1/12

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Antarctica	99999999	1/7/12	2	$92.60	$20.70
			219	$722.40	
N = 1 Total N = 10					

Program: Creating a Listing Report

```
options nodate pageno=1 linesize=80 pagesize=40;

ods html close;
ods listing;


proc sort data=exprev;
   by sale_type order_date;
run;

proc print data=exprev n noobs sumlabel;

   by sale_type order_date;
   sum price quantity;

   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;

ods listing close;
ods html;
```

Program Description

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.

```
options nodate pageno=1 linesize=80 pagesize=40;
```

Close the HTML destination and open the Listing destination. The HTML destination is open by default.

```
ods html close;
ods listing;
```

Sort the data set. PROC SORT sorts the observations by Sale_Type and Order_Date.

```
proc sort data=exprev;
   by sale_type order_date;
run;
```

Print the report, suppress the printing of observation numbers, print the total number of observations for the selected variables and use the BY variable labels in place of the BY variable names in the summary line. The N option prints the number of observations in a BY group at the end of that BY group and prints the total number of observations used in the report at the bottom of the report. NOOBS suppresses the printing of observation numbers at the beginning of the rows. The SUMLABEL option prints the BY variable labels in the summary line in place of the BY variables.

```
proc print data=exprev n noobs sumlabel;
```

Create a separate section of the report for each BY group, and sum the values for the selected variables. The BY statement produces a separate section of the report for each BY group. The SUM statement alone sums the values of Price and Quantity for the entire data set. Because the program contains a BY statement, the SUM statement also sums the values of Price and Quantity for each BY group that contains more than one observation.

```
   by sale_type order_date;
   sum price quantity;
```

Establish a label for selected variables, format the values of specified variables, and create a title. The LABEL statement associates a label with the variables Sale_Type and Order_Date for the duration of the PROC PRINT step. The labels are used in the BY line at the beginning of each BY group and in the summary line in place of BY variables. The FORMAT statement assigns a format to the variables Price and Cost for this report. The TITLE statement specifies a title.

```
   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;
```

Close the Listing destination and re-open the HTML destination.

```
ods listing close;
ods html;
```

Output: Listing

The report uses default column headings (variable names) because neither the SPLIT= nor the LABEL option is used. Nevertheless, the BY line at the top of each section of the report shows the BY variables' labels and their values. The BY variables' labels identifies the subtotals in the report summary line.

PROC PRINT sums Price and Quantity for each BY group that contains more than one observation. However, sums are shown only for the BY variables whose values change from one BY group to the next. For example, in the first BY group, where the sale type is **`Catalog Sale`** and the sale date is >**`1/1/12`**, Quantity and Price are summed only for the sale date because the next BY group is for the same sale type.

```
        Retail and Quantity Totals for Each Sale Date and Sale Type          1

---------------------- Sale Type=Catalog Sale Date=1/1/12 ----------------------

                          Ship_
   Country         Emp_ID  Date     Quantity         Price          Cost

   Puerto Rico    99999999 1/5/12       14          $51.20        $12.10
   Aruba          99999999 1/4/12       30         $123.70        $59.00
   Bahamas        99999999 1/4/12        8         $113.40        $28.45
   Bermuda        99999999 1/4/12        7          $41.00         $9.25
   -----------                         --------    ----------
   Sale Date                             59         $329.30

                                  N = 4


---------------------- Sale Type=Catalog Sale Date=1/2/12 ----------------------

                                   Ship_
Country                   Emp_ID    Date    Quantity        Price         Cost

British Virgin Islands   99999999 1/5/12       11         $40.20        $20.20
Canada                   99999999 1/5/12      100         $11.80         $5.00
El Salvador              99999999 1/6/12       21        $266.40        $66.70
Brazil                   120127   1/2/12       12         $73.40        $18.45
French Guiana            120935   1/2/12       15         $96.40        $43.85
Grenada                  120931   1/2/12       19         $56.30        $25.05
Paraguay                 120603   1/2/12       17        $117.60        $58.90
Peru                     120845   1/2/12       12         $93.80        $41.75
----------------------                      --------   ----------
Sale Date                                      207        $755.90
Sale Type                                      266      $1,085.20

                                  N = 8
```

Retail and Quantity Totals for Each Sale Date and Sale Type 2

---------- Sale Type=In Store Sale Date=1/1/12 ----------

Country	Emp_ID	Ship_ Date	Quantity	Price	Cost
Virgin Islands (U.S.)	99999999	1/4/12	25	$31.10	$15.65

N = 1

---------- Sale Type=In Store Sale Date=1/2/12 ----------

Country	Emp_ID	Ship_ Date	Quantity	Price	Cost
Belize	120458	1/2/12	2	$146.40	$36.70
Cayman Islands	120454	1/2/12	20	$71.00	$32.30
Guatemala	120931	1/2/12	13	$144.40	$65.70
Jamaica	99999999	1/4/12	23	$169.80	$38.70
Mexico	120127	1/2/12	30	$211.80	$33.65
Montserrat	120127	1/2/12	19	$184.20	$36.90
Anguilla	99999999	1/6/12	15	$233.50	$22.25
Antigua/Barbuda	120458	1/2/12	31	$99.60	$45.35
Argentina	99999999	1/6/12	42	$408.80	$87.15
Barbados	99999999	1/6/12	26	$94.80	$42.60
Bolivia	120127	1/2/12	26	$66.00	$16.60
Chile	120447	1/2/12	20	$19.10	$8.75
Ecuador	121042	1/2/12	11	$100.90	$50.55
Falkland Islands	120932	1/2/12	15	$61.40	$30.80
Guyana	120455	1/2/12	25	$132.80	$30.25
Martinique	120841	1/3/12	16	$56.30	$31.05
Netherlands Antilles	99999999	1/6/12	31	$41.80	$19.45
----------			--------	----------	
Sale Date			365	$2,242.60	
Sale Type			390	$2,273.70	

N = 17

```
         Retail and Quantity Totals for Each Sale Date and Sale Type          3

---------------------- Sale Type=Internet Sale Date=1/1/12 ----------------------

                          Ship_
     Country       Emp_ID  Date     Quantity         Price          Cost

    Antarctica    99999999  1/7/12        2          $92.60        $20.70

                                  N = 1


---------------------- Sale Type=Internet Sale Date=1/2/12 ----------------------

                                  Ship_
 Country                 Emp_ID   Date     Quantity         Price          Cost

 Costa Rica              99999999 1/6/12         31          $53.00        $26.60
 Cuba                    121044   1/2/12         12          $42.40        $19.35
 Dominican Republic      121040   1/2/12         13          $48.00        $23.95
 Haiti                   121059   1/2/12          5          $47.90        $23.45
 Honduras                120455   1/2/12         20          $66.40        $30.25
 Nicaragua               120932   1/2/12         16         $122.00        $28.75
 Panama                  99999999 1/6/12         20          $88.20        $38.40
 Saint Kitts/Nevis       99999999 1/6/12         20          $41.40        $18.00
 St. Helena              120360   1/2/12         19          $94.70        $47.45
 St. Pierre/Miquelon     120842   1/16/12        16         $103.80        $47.25
 Turks/Caicos Islands    120372   1/2/12         10          $57.70        $28.95
 United States           120372   1/2/12         20          $88.20        $38.40
 Colombia                121059   1/2/12         28         $361.40        $90.45
 Dominica                121043   1/2/12         35         $121.30        $57.80
 Guadeloupe              120445   1/2/12         21         $231.60        $48.70
 St. Lucia               120845   1/2/12         19          $64.30        $28.65
 --------------------                          --------   ----------
 Sale Date                                        305      $1,632.30

                                  N = 16
```

```
         Retail and Quantity Totals for Each Sale Date and Sale Type          4

---------------------- Sale Type=Internet Sale Date=1/3/12 ----------------------

                         Ship_
      Country     Emp_ID  Date     Quantity         Price          Cost

     Suriname     120538  1/3/12         22         $110.80        $29.35
     ---------                     --------    ----------
     Sale Type                        329       $1,835.70
                                   ========    ==========
                                      985       $5,194.60

                                N = 1
                           Total N = 48
```

Example 6: Limiting the Number of Sums in a Report

Features: BY statement
SUM statement
SUMBY statement

Other features: FORMAT statement
LABEL statement
ODS PDF statement
SORT procedure
TITLE statement

Data set: EXPREV

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 nodate pageno=1 obs=10;

ods html close;
ods pdf file='your_file.pdf';

proc sort data=exprev;
   by sale_type order_date;
run;

proc print data=exprev noobs sumlabel;

   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

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 OBS= option specifies the number of observations to be displayed.

```
options nodate pageno=1 obs=10;
```

Produce PDF output and specify the file to store the output in. The ODS HTML 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 html 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;
```

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.

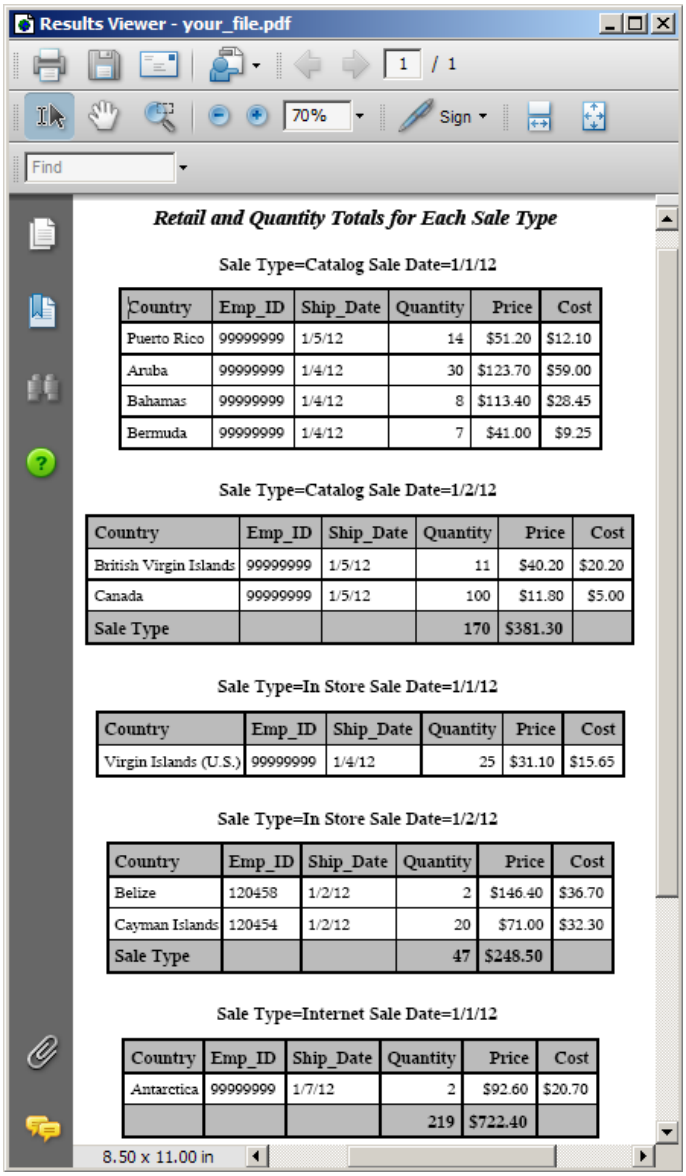
```
   format price dollar10.2 cost dollar10.2;
   title 'Retail and Quantity Totals for Each Sale Type';
run;
```

Close the PS destination. The ODS PDF CLOSE statement closes the PDF destination.

```
ods pdf close;
```

Output: PDF

Output 37.14 *Limiting the Number of Sums in a Report: PDF Output*



Retail and Quantity Totals for Each Sale Type

Sale Type=Catalog Sale Date=1/1/12

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Puerto Rico	99999999	1/5/12	14	$51.20	$12.10
Aruba	99999999	1/4/12	30	$123.70	$59.00
Bahamas	99999999	1/4/12	8	$113.40	$28.45
Bermuda	99999999	1/4/12	7	$41.00	$9.25

Sale Type=Catalog Sale Date=1/2/12

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
British Virgin Islands	99999999	1/5/12	11	$40.20	$20.20
Canada	99999999	1/5/12	100	$11.80	$5.00
Sale Type			**170**	**$381.30**	

Sale Type=In Store Sale Date=1/1/12

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Virgin Islands (U.S.)	99999999	1/4/12	25	$31.10	$15.65

Sale Type=In Store Sale Date=1/2/12

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Belize	120458	1/2/12	2	$146.40	$36.70
Cayman Islands	120454	1/2/12	20	$71.00	$32.30
Sale Type			**47**	**$248.50**	

Sale Type=Internet Sale Date=1/1/12

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Antarctica	99999999	1/7/12	2	$92.60	$20.70
			219	**$722.40**	

Program: Creating a PDF Report with the STYLE Option

```
options nodate pageno=1 linesize=80 pagesize=40 obs=10;

ods pdf file='your_file.pdf';

proc sort data=exprev;
   by sale_type order_date;
run;

proc print data=exprev noobs;

   by sale_type order_date;
```

```
   sum price / style(TOTAL) = [background =blue color=white];
   sum quantity / style(GRANDTOTAL) = [background =yellow color=red];
   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

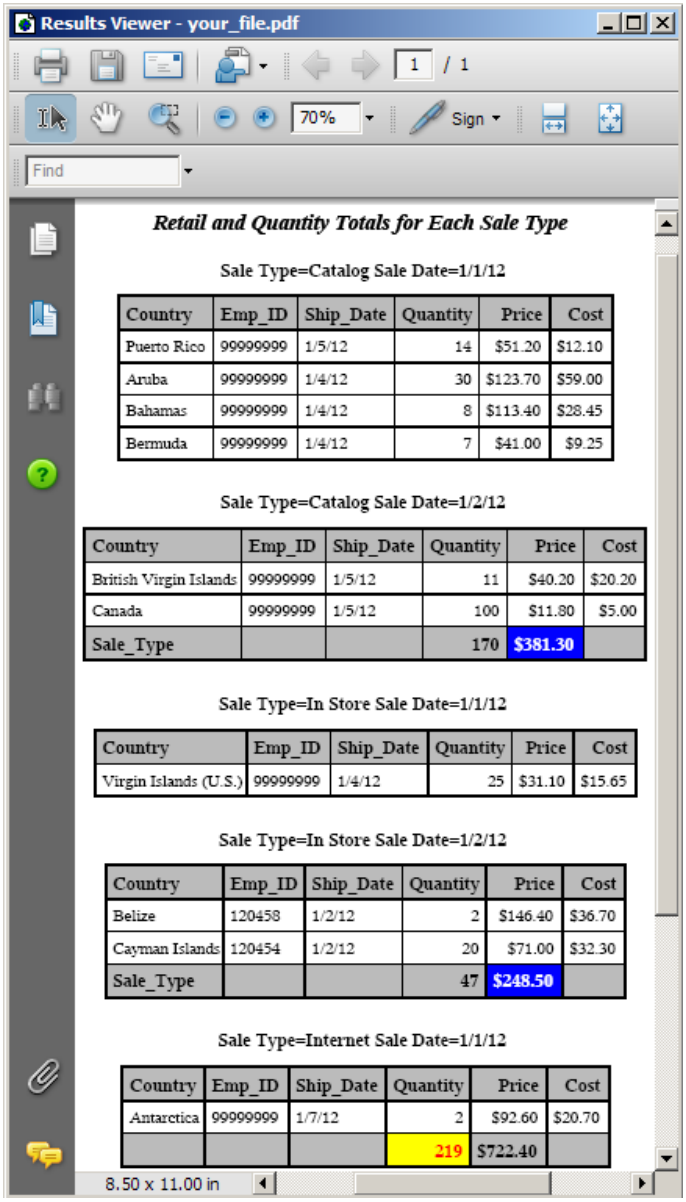
```
options nodate pageno=1 linesize=80 pagesize=40 obs=10;

ods pdf file='your_file.pdf';

proc sort data=exprev;
   by sale_type order_date;
run;

proc print data=exprev noobs;

   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 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 yellow and the font color be changed to red.

```
   sum price / style(TOTAL) = [background =blue color=white];
   sum quantity / style(GRANDTOTAL) = [background =yellow color=red];
   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 37.15 *Limiting the Number of Sums in a Report: PostScript Output Using Styles*



Example 7: Controlling the Layout of a Report with Many Variables

Features: PROC PRINT statement options:
ROWS=

ID statement options:
STYLE

Other features: ODS RTF statement

SAS data set options:
OBS=

Data set: EMPDATA

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 1005 and "Page Layout " on page 998.

These reports use a page size of 24 and a line size of 64 to help illustrate the different layouts.

Program: Creating a Listing Report

```
data empdata;
   input IdNumber $ 1-4 LastName $ 9-19 FirstName $ 20-29
         City $ 30-42 State $ 43-44 /
         Gender $ 1 JobCode $ 9-11 Salary 20-29 @30 Birth date9.
         @43 Hired date9. HomePhone $ 54-65;
   format birth hired date9.;
   datalines;
1919    Adams      Gerald    Stamford     CT
M       TA2        34376     15SEP1970      07JUN2005    203/781-1255
1653    Alexander  Susan     Bridgeport   CT
F       ME2        35108     18OCT1972      12AUG1998    203/675-7715

. . . more lines of data . . .

1407    Grant      Daniel    Mt. Vernon   NY
M       PT1        68096     26MAR1977      21MAR1998    914/468-1616
1114    Green      Janice    New York     NY
F       TA2        32928     21SEP1977      30JUN2006    212/588-1092
;

proc print data=empdata(obs=12);
   id idnumber;
   title 'Personnel Data';
run;

proc print data=empdata(obs=12) rows=page;
   id idnumber;
   title 'Personnel Data';
run;
```

Program Description

Create the EMPDATA data set. The data set EMPDATA contains personal and job-related information about a company's employees. The DATA step creates this data set.

```
data empdata;
   input IdNumber $ 1-4 LastName $ 9-19 FirstName $ 20-29
         City $ 30-42 State $ 43-44 /
         Gender $ 1 JobCode $ 9-11 Salary 20-29 @30 Birth date9.
         @43 Hired date9. HomePhone $ 54-65;
   format birth hired date9.;
   datalines;
```

```
1919    Adams      Gerald    Stamford     CT
M       TA2        34376     15SEP1970      07JUN2005    203/781-1255
1653    Alexander  Susan     Bridgeport   CT
F       ME2        35108     18OCT1972      12AUG1998    203/675-7715

. . . more lines of data . . .

1407    Grant      Daniel    Mt. Vernon   NY
M       PT1        68096     26MAR1977      21MAR1998    914/468-1616
1114    Green      Janice    New York     NY
F       TA2        32928     21SEP1977      30JUN2006    212/588-1092
;
```

Print only the first 12 observations in a data set. The OBS= data set option uses only the first 12 observations to create the report. (This is just to conserve space here.) The ID statement identifies observations with the formatted value of IdNumber rather than with the observation number. This report is shown in "Output: Listing" on page 1058.

```
proc print data=empdata(obs=12);
   id idnumber;
   title 'Personnel Data';
run;
```

Print a report that contains only one row of variables on each page. ROWS=PAGE prints only one row of variables for each observation on a page. This report is shown in Output 37.17 on page 1060.

```
proc print data=empdata(obs=12) rows=page;
   id idnumber;
   title 'Personnel Data';
run;
```

Output: Listing

In the traditional procedure output, each page of this report contains values for all variables in each observation. In the HTML output, this report is identical to the report that uses ROWS=PAGE.

Note that PROC PRINT automatically splits the variable names that are used as column headings at a change in capitalization if the entire name does not fit in the column. Compare the column headings for LastName (which fits in the column) and FirstName (which does not fit in the column).

Output 37.16 *Default Layout for a Report with Many Variables: Listing Output*

```
                Personnel Data                          1

  Id                    First
Number    LastName      Name            City        State    Gender

 1919     Adams         Gerald       Stamford        CT         M
 1653     Alexander     Susan        Bridgeport      CT         F
 1400     Apple         Troy         New York        NY         M
 1350     Arthur        Barbara      New York        NY         F
 1401     Avery         Jerry        Paterson        NJ         M
 1499     Barefoot      Joseph       Princeton       NJ         M
 1101     Baucom        Walter       New York        NY         M

  Id       Job
Number     Code     Salary       Birth       Hired      HomePhone

 1919      TA2       34376     15SEP70     07JUN05     203/781-1255
 1653      ME2       35108     18OCT72     12AUG98     203/675-7715
 1400      ME1       29769     08NOV85     19OCT06     212/586-0808
 1350      FA3       32886     03SEP63     01AUG00     718/383-1549
 1401      TA3       38822     16DEC68     20NOV93     201/732-8787
 1499      ME3       43025     29APR62     10JUN95     201/812-5665
 1101      SCP       18723     09JUN80     04OCT98     212/586-8060
```

```
                Personnel Data                          2

  Id                    First
Number    LastName      Name            City        State    Gender

 1333     Blair         Justin       Stamford        CT         M
 1402     Blalock       Ralph        New York        NY         M
 1479     Bostic        Marie        New York        NY         F
 1403     Bowden        Earl         Bridgeport      CT         M
 1739     Boyce         Jonathan     New York        NY         M

  Id       Job
Number     Code     Salary       Birth       Hired      HomePhone

 1333      PT2       88606     02APR79     13FEB03     203/781-1777
 1402      TA2       32615     20JAN71     05DEC98     718/384-2849
 1479      TA3       38785     25DEC66     08OCT03     718/384-8816
 1403      ME1       28072     31JAN79     24DEC99     203/675-3434
 1739      PT1       66517     28DEC82     30JAN00     212/587-1247
```

Each page of this report contains values for only some of the variables in each observation. However, each page contains values for more observations than the default report does.

Output 37.17 *Layout Produced by the ROWS=PAGE Option: Listing Output*

```
                          Personnel Data                          3

  Id                  First
Number    LastName    Name           City       State    Gender

 1919     Adams       Gerald      Stamford       CT         M
 1653     Alexander   Susan       Bridgeport     CT         F
 1400     Apple       Troy        New York       NY         M
 1350     Arthur      Barbara     New York       NY         F
 1401     Avery       Jerry       Paterson       NJ         M
 1499     Barefoot    Joseph      Princeton      NJ         M
 1101     Baucom      Walter      New York       NY         M
 1333     Blair       Justin      Stamford       CT         M
 1402     Blalock     Ralph       New York       NY         M
 1479     Bostic      Marie       New York       NY         F
 1403     Bowden      Earl        Bridgeport     CT         M
 1739     Boyce       Jonathan    New York       NY         M
```

```
                          Personnel Data                          4

  Id      Job
Number    Code     Salary      Birth      Hired      HomePhone

 1919     TA2       34376    15SEP70    07JUN05    203/781-1255
 1653     ME2       35108    18OCT72    12AUG98    203/675-7715
 1400     ME1       29769    08NOV85    19OCT06    212/586-0808
 1350     FA3       32886    03SEP63    01AUG00    718/383-1549
 1401     TA3       38822    16DEC68    20NOV93    201/732-8787
 1499     ME3       43025    29APR62    10JUN95    201/812-5665
 1101     SCP       18723    09JUN80    04OCT98    212/586-8060
 1333     PT2       88606    02APR79    13FEB03    203/781-1777
 1402     TA2       32615    20JAN71    05DEC98    718/384-2849
 1479     TA3       38785    25DEC66    08OCT03    718/384-8816
 1403     ME1       28072    31JAN79    24DEC99    203/675-3434
 1739     PT1       66517    28DEC82    30JAN00    212/587-1247
```

Program: Creating an RTF Report

```
options nodate pageno=1 linesize=64 pagesize=24;

ods rtf file='your_file.rtf';

proc print data=empdata(obs=12);
   id idnumber;
   title 'Personnel Data';
run;

ods rtf close;
```

Program Description

The RTF output shows all data in one row. The ROWS= option is valid only for the Listing destination.

```
options nodate pageno=1 linesize=64 pagesize=24;
```

Create output for Microsoft Word and specify the file to store the output in. The ODS RTF statement opens the RTF destination and creates output formatted for Microsoft Word. The FILE= argument specifies the external file that contains the RTF output.

```
ods rtf file='your_file.rtf';

proc print data=empdata(obs=12);
   id idnumber;
   title 'Personnel Data';
run;
```

Close the RTF destination. The ODS RTF CLOSE statement closes the RTF destination.

```
ods rtf close;
```

Output: RTF

Output 37.18 *Layout for a Report with Many Variables: RTF Output*

1

Personnel Data

IdNumber	LastName	FirstName	City	State	Gender	JobCode	Salary	Birth	Hired	HomePhone
1919	Adams	Gerald	Stamford	CT	M	TA2	34376	15SEP70	07JUN05	203/781-1255
1653	Alexander	Susan	Bridgeport	CT	F	ME2	35108	18OCT72	12AUG98	203/675-7715
1400	Apple	Troy	New York	NY	M	ME1	29769	08NOV85	19OCT06	212/586-0808
1350	Arthur	Barbara	New York	NY	F	FA3	32886	03SEP63	01AUG00	718/383-1549
1401	Avery	Jerry	Paterson	NJ	M	TA3	38822	16DEC68	20NOV93	201/732-8787
1499	Barefoot	Joseph	Princeton	NJ	M	ME3	43025	29APR62	10JUN95	201/812-5665
1101	Baucom	Walter	New York	NY	M	SCP	18723	09JUN80	04OCT98	212/586-8060
1333	Blair	Justin	Stamford	CT	M	PT2	88606	02APR79	13FEB03	203/781-1777
1402	Blalock	Ralph	New York	NY	M	TA2	32615	20JAN71	05DEC98	718/384-2849
1479	Bostic	Marie	New York	NY	F	TA3	38785	25DEC66	08OCT03	718/384-8816
1403	Bowden	Earl	Bridgeport	CT	M	ME1	28072	31JAN79	24DEC99	203/675-3434
1739	Boyce	Jonathan	New York	NY	M	PT1	66517	28DEC82	30JAN00	212/587-1247

Example 8: Creating a Customized Layout with BY Groups and ID Variables

Features: Procedure features:
- BY statement
- ID statement
- SUM statement
- VAR statement

Other features: SORT procedure

Data set: EMPDATA

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 a Listing Report

```
options nodate pageno=1 linesize=64 pagesize=60;
proc sort data=empdata out=tempemp;
   by jobcode gender;
run;

proc print data=tempemp split='*';

   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 'Salay Expenses';
run;
```

Program Description

Create and sort a temporary data set. PROC SORT creates a temporary data set in which the observations are sorted by JobCode and Gender.

```
options nodate pageno=1 linesize=64 pagesize=60;
proc sort data=empdata out=tempemp;
   by jobcode gender;
run;
```

Identify the character that starts a new line in column headings. SPLIT= identifies the asterisk as the character that starts a new line in column headings.

```
proc print data=tempemp split='*';
```

Specify the variables to include in the report. The VAR statement and the ID statement together select the variables to include in the report. The ID statement and the BY statement produce the special format.

```
   id jobcode;
   by jobcode;
   var gender salary;
```

Calculate the total value for each BY group. The SUM statement totals the values of Salary for each BY group and for the whole report.

```
sum salary;
```

Assign labels to the appropriate variables. The LABEL statement associates a label with each variable for the duration of the PROC PRINT step. When you use SPLIT= in the PROC PRINT statement, the procedure uses labels for column headings.

```
label jobcode='Job Code*========'
      gender='Gender*======'
      salary='Annual Salary*=============';
```

Create formatted columns. The FORMAT statement assigns a format to Salary for this report. The WHERE statement selects for the report only the observations for job codes that contain the letters 'FA' or 'ME'. The TITLE statements specify two titles.

```
format salary dollar11.2;
where jobcode contains 'FA' or jobcode contains 'ME';
title 'Salay Expenses';
run;
```

Output: Listing

The ID and BY statements work together to produce this layout. The ID variable is listed only once for each BY group. The BY lines are suppressed. Instead, the value of the ID variable, JobCode, identifies each BY group.

Output 37.19 *Creating a Customized Layout with BY Groups and ID Variables: Listing Output*

```
                        Salay Expenses                          1

              Job Code    Gender    Annual Salary
              ========    ======    =============

                FA1         F         $23,177.00
                            F         $22,454.00
                            M         $22,268.00
              --------              -------------
                FA1                   $67,899.00

                FA2         F         $28,888.00
                            F         $27,787.00
                            M         $28,572.00
              --------              -------------
                FA2                   $85,247.00

                FA3         F         $32,886.00
                            F         $33,419.00
                            M         $32,217.00
              --------              -------------
                FA3                   $98,522.00

                ME1         M         $29,769.00
                            M         $28,072.00
                            M         $28,619.00
              --------              -------------
                ME1                   $86,460.00

                ME2         F         $35,108.00
                            F         $34,929.00
                            M         $35,345.00
                            M         $36,925.00
                            M         $35,090.00
                            M         $35,185.00
              --------              -------------
                ME2                  $212,582.00

                ME3         M         $43,025.00
                                     =============
                                      $593,735.00
```

Program: Creating an HTML Report

```
options nodate pageno=1 linesize=64 pagesize=60 obs=15;
proc sort data=empdata out=tempemp;
   by jobcode gender;
run;

ods html file='your_file.html';

proc print data=tempemp (obs=10) split='*';

 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

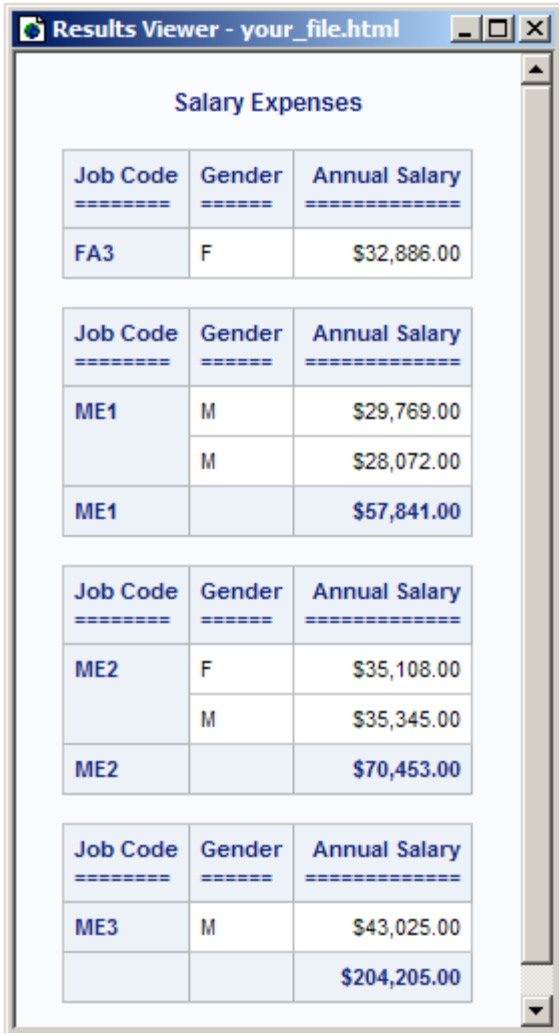
```
options nodate pageno=1 linesize=64 pagesize=60 obs=15;
proc sort data=empdata out=tempemp;
   by jobcode gender;
run;
```

Produce HTML output and specify the file to store the output in. The HTML destination is the default ODS destination. The ODS HTML statement FILE= option specifies the external file that contains the HTML output.

```
ods html file='your_file.html';

proc print data=tempemp (obs=10) split='*';

 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: HTML

Output 37.20 *Creating a Customized Layout with BY Groups and ID Variables: Default HTML Output*



Program: Creating an HTML Report with the STYLE Option

```
options nodate pageno=1 linesize=64 pagesize=60 obs=15;
proc sort data=empdata out=tempemp;
   by jobcode gender;
run;

ods html file='your_file.html';

proc print data=tempemp (obs=10) split='*' 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

```
options nodate pageno=1 linesize=64 pagesize=60 obs=15;
proc sort data=empdata out=tempemp;
   by jobcode gender;
run;

ods html file='your_file.html';
```

Create stylized HTML 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 input data be changed to blue and the foreground of these cells be changed to white.

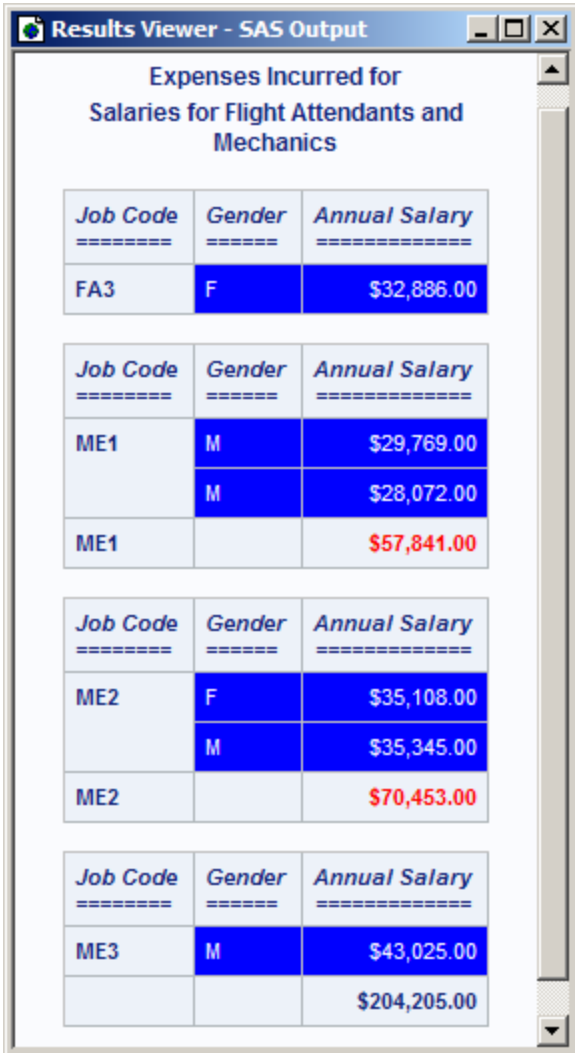
```
proc print data=tempemp (obs=10) split='*' style(HEADER)
=
                                    {fontstyle=italic}
                                    style(DATA) =
                                    {backgroundcolor=blue foreground =
white};

 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: HTML with Styles

Output 37.21 *Creating a Customized Layout with BY Groups and ID Variables: HTML Output Using Styles*



Example 9: 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

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;

options nodate pageno=1 linesize=70 pagesize=60;
%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));
```

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;
```

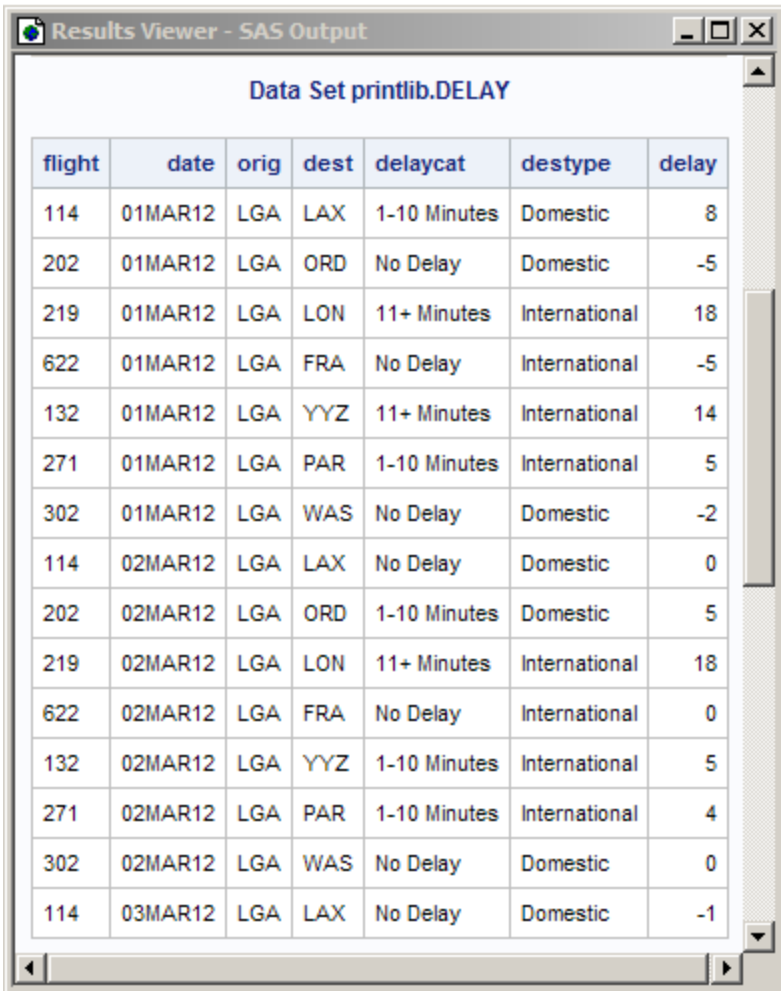
Print the data sets and end the macro. The %DO loop issues a PROC PRINT step for each data set. The %MEND statement ends the macro.

```
   %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.

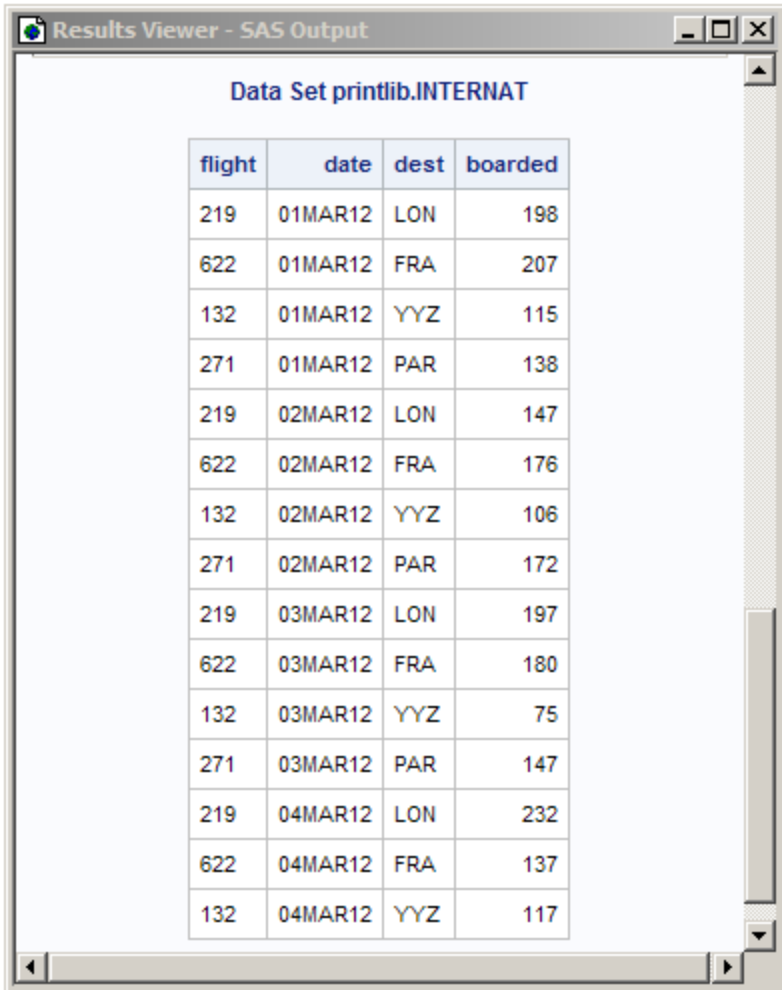
```
options nodate pageno=1 linesize=70 pagesize=60;
%printall(printlib)
```

Output: HTML



Data Set printlib.DELAY

flight	date	orig	dest	delaycat	destype	delay
114	01MAR12	LGA	LAX	1-10 Minutes	Domestic	8
202	01MAR12	LGA	ORD	No Delay	Domestic	-5
219	01MAR12	LGA	LON	11+ Minutes	International	18
622	01MAR12	LGA	FRA	No Delay	International	-5
132	01MAR12	LGA	YYZ	11+ Minutes	International	14
271	01MAR12	LGA	PAR	1-10 Minutes	International	5
302	01MAR12	LGA	WAS	No Delay	Domestic	-2
114	02MAR12	LGA	LAX	No Delay	Domestic	0
202	02MAR12	LGA	ORD	1-10 Minutes	Domestic	5
219	02MAR12	LGA	LON	11+ Minutes	International	18
622	02MAR12	LGA	FRA	No Delay	International	0
132	02MAR12	LGA	YYZ	1-10 Minutes	International	5
271	02MAR12	LGA	PAR	1-10 Minutes	International	4
302	02MAR12	LGA	WAS	No Delay	Domestic	0
114	03MAR12	LGA	LAX	No Delay	Domestic	-1



flight	date	dest	boarded
219	01MAR12	LON	198
622	01MAR12	FRA	207
132	01MAR12	YYZ	115
271	01MAR12	PAR	138
219	02MAR12	LON	147
622	02MAR12	FRA	176
132	02MAR12	YYZ	106
271	02MAR12	PAR	172
219	03MAR12	LON	197
622	03MAR12	FRA	180
132	03MAR12	YYZ	75
271	03MAR12	PAR	147
219	04MAR12	LON	232
622	04MAR12	FRA	137
132	04MAR12	YYZ	117

Chapter 38

PRINTTO Procedure

Overview: PRINTTO Procedure

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 38.1 *Default Destinations for SAS Log and Procedure Output*

Method of Running SAS	SAS Log Destination	Procedure Output Destination
Windowing environment	LOG window	Results Viewer window
Interactive line mode	Display monitor (as statements are entered)	Display monitor (as each step executes)
Noninteractive mode or batch mode	Depends on the host operating system	Depends on the operating environment

Operating Environment Information
For information and examples specific to your operating system or environment, see the documentation for your operating environment.

Syntax: PRINTTO Procedure

See: "PRINTTO Procedure: UNIX" in *SAS Companion for UNIX Environments*
"PRINTTO Procedure: Windows" in *SAS Companion for Windows*
"PRINTTO Procedure: z/OS" in *SAS Companion for z/OS*

PROC PRINTTO <*option(s)*>;

Statement	Task	Example
"PROC PRINTTO Statement"	Define destinations, other than ODS destinations, for SAS procedure output and for the SAS log	Ex. 1, Ex. 2, Ex. 3, Ex. 4

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 "Routing SAS Log or Procedure Output Directly to a Printer" on page 1078 and "Example 4: Routing to a Printer" on page 1092.

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.

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 1079
"Example 2: Routing to SAS Catalog Entries" on page 1083
"Example 3: Using Procedure Output as an Input File" on page 1087
"Example 4: Routing to a Printer" on page 1092

Syntax

PROC PRINTTO <*option(s)*>;

Summary of Optional Arguments

LABEL='*description*'
: provides a description for a SAS log or procedure output stored in a SAS catalog entry.

LOG=LOG | *file-specification* | *SAS-catalog-entry*
: routes the SAS log to a permanent external file or SAS catalog entry.

NEW
: replaces the file instead of appending to it.

PRINT= PRINT | *file-specification* | *SAS-catalog-entry*
: routes procedure output to a permanent external file or SAS catalog entry or printer.

UNIT=*nn*
: routes the output to the file identified by the fileref.

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.

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 1079

"Example 2: Routing to SAS Catalog Entries" on page 1083

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= or the PRINT= option.

Example: "Example 2: Routing to SAS Catalog Entries" on page 1083

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.

Operating environment: For more information about *log-filename*, see the documentation for your operating environment.

fileref
a fileref previously assigned to an external file.

SAS-catalog-entry
routes the SAS log to a SAS catalog entry. By default, *libref* is SASUSER, *catalog* is PROFILE, and *type* is LOG. Express *SAS-catalog-entry* in one of the following ways:

libref.catalog.entry<.LOG>
a SAS catalog entry stored in the SAS library and SAS catalog specified.

catalog.entry<.LOG>
a SAS catalog entry stored in the specified SAS catalog in the default SAS library SASUSER.

entry.LOG
a SAS catalog entry stored in the default SAS library and catalog: SASUSER.PROFILE.

fileref
a fileref previously assigned to a SAS catalog entry. Search for "FILENAME, CATALOG Access Method" in the SAS online documentation.

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.)

Examples:
"Example 1: Routing to External Files" on page 1079

"Example 2: Routing to SAS Catalog Entries" on page 1083

"Example 3: Using Procedure Output as an Input File" on page 1087

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 1079

"Example 2: Routing to SAS Catalog Entries" on page 1083

"Example 3: Using Procedure Output as an Input File" on page 1087

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.

Operating Environment Information
For more information about using *print-filename*, see the documentation for your operating environment.

fileref
a fileref previously assigned to an external file.

Operating Environment Information
For additional information about *file-specification* for the PRINT option, see the documentation for your operating environment.

SAS-catalog-entry
routes procedure output to a SAS catalog entry. By default, *libref* is SASUSER, *catalog* is PROFILE, and *type* is OUTPUT. Express *SAS-catalog-entry* in one of the following ways:

libref.catalog.entry<.OUTPUT>
a SAS catalog entry stored in the SAS library and SAS catalog specified.

catalog.entry<.OUTPUT>
a SAS catalog entry stored in the specified SAS catalog in the default SAS library SASUSER.

entry.OUTPUT
a SAS catalog entry stored in the default SAS library and catalog: SASUSER.PROFILE.

fileref
a fileref previously assigned to a SAS catalog entry. Search for "FILENAME, CATALOG Access Method" in the SAS online documentation.

Alias: FILE=, NAME=

Default: PRINT

Requirement: To route procedure output to a file, you must open the LISTING destination. Specify `ods listing;` before you run the PRINTTO procedure.

Interactions:
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.

Example: "Example 3: Using Procedure Output as an Input File" on page 1087

UNIT=*nn*

routes the output to the file identified by the fileref FT*nn*F001, where *nn* is an integer between 1 and 99.

Range: 1–99, integer only.

Tip: You can define this fileref yourself. However, some operating systems predefine certain filerefs in this form.

Setting 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.

Routing 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 1092.

For more information, see "FILENAME Statement" in *SAS Statements: Reference*.

Operating Environment Information

For examples of printer names, see the documentation for your operating system.

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.

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
;

ods listing;
proc printto print='output-file'
new;
run;

proc print data=numbers;
   title 'Listing of NUMBERS Data Set';
run;

proc printto;
run;
ods listing 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. 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. The LISTING destination must be open in order to route SAS output to an external file. Because NEW is specified, any output written to *output-file* will overwrite the file's current contents.

```
ods listing;
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. Then, close the LISTING destination.

```
proc printto;
run;
ods listing close;
```

Log

Log 38.1 *Portion of Log Routed to the Default Destination*

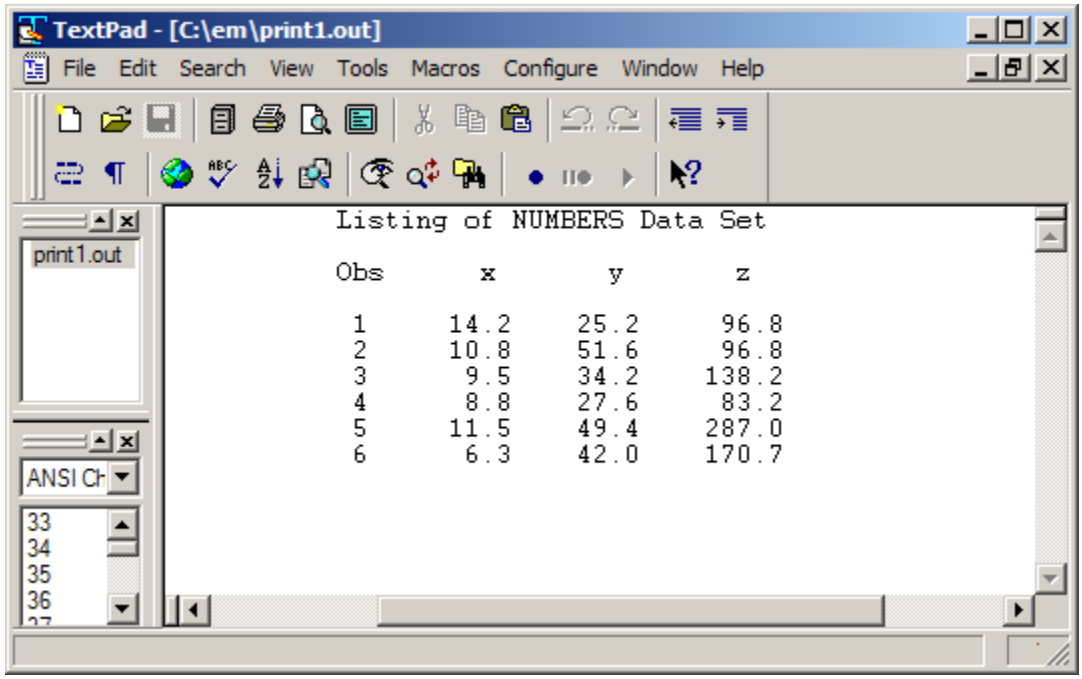
```
01   options nodate pageno=1 linesize=80 pagesize=60 source;
02
03   proc printto log='log-file';
04   run;

NOTE: PROCEDURE PRINTTO used (Total process time):
      real time           0.01 seconds
      cpu time            0.00 seconds
```

Log 38.2 *Portion of Log Routed to an External File*

```
  NOTE: PROCEDURE PRINTTO used (Total process time):
      real time           0.01 seconds
      cpu time            0.00 seconds


68
69   data numbers;
70      input x y z;
71      datalines;

NOTE: The data set WORK.NUMBERS has 6 observations and 3 variables.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds


78   ;
79   ods listing;
80   proc printto
81   print='c:\em\print1.out'
82   new;
83   run;

NOTE: PROCEDURE PRINTTO used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds


84
85   proc print data=numbers;
86      title 'Listing of NUMBERS Data Set';
87   run;

NOTE: There were 6 observations read from the data set WORK.NUMBERS.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.32 seconds
      cpu time            0.07 seconds


88
89   proc printto;
90   run;
NOTE: PROCEDURE PRINTTO used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds


91   ods listing close;
```

Output

Output 38.1 *Procedure Output Routed to an External File*



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=test.log label='Inventory program' new;
run;

data lib1.inventry;
   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
;

ods listing;
proc printto print=lib1.cat1.inventry.output
             label='Inventory program' new;
run;

proc report data=lib1.inventry nowindows headskip;
   column dept item season year;
   title 'Current Inventory Listing';
run;

proc printto;
run;
ods listing close;
```

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 SAS catalog entry named SASUSER.PROFILE.TEST.LOG. The PRINTTO procedure uses the default libref and catalog SASUSER.PROFILE because only the entry name and type are specified. LABEL= assigns a description for the catalog entry.

```
proc printto log=test.log label='Inventory program' new;
run;
```

Create the LIB1.INVENTORY data set. The DATA step creates a permanent SAS data set.

```
data lib1.inventry;
   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. To write to a SAS catalog entry you must open the LISTING destination. PROC PRINTTO routes procedure output from

the subsequent PROC REPORT step to the SAS catalog entry LIB1.CAT1.INVENTRY.OUTPUT. LABEL= assigns a description for the catalog entry.

```
ods listing;
proc printto print=lib1.cat1.inventry.output
             label='Inventory program' new;
run;

proc report data=lib1.inventry 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. Close the Listing destination.

```
proc printto;
run;
ods listing close;
```

Log

To view this log using SAS Explorer, select **Sasuser** ⇨ **Profile**. Double-click on Test. The log opens in NOTEPAD.

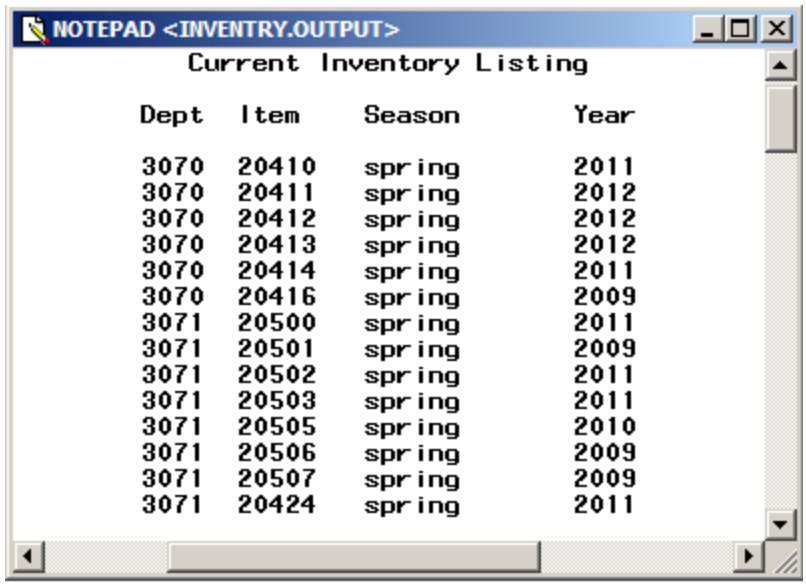
Log 38.3 *SAS Log Routed to SAS Catalog Entry SASUSER.PROFILE.TEST.LOG.*

```
NOTE: PROCEDURE PRINTTO used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds


49
50   data lib1.inventry;
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.INVENTRY 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.inventry.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.inventry 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.INVENTRY.
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

To view this log using SAS Explorer, select **Lib1** ⇨ **Cat1**. Double-click on **Inventry**. The output opens in NOTEPAD.

Output 38.2 *Procedure Output Routed to SAS Catalog Entry LIB1.CAT1.INVENTRY.OUTPUT.*



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';

ods listing;
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. In order to write to an external file, the LISTING destination must be open. See PROC FREQ Output Routed to the External File Referenced as ROUTED below.

```
filename routed 'output-filename';

ods listing;
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 PROBTEST. The DATA step uses ROUTED, the file containing PROC FREQ output, as an input file and creates the data set PROBTEST. This DATA step reads all records in ROUTED but creates an observation only from a record that begins with **`Chi-Squa`**.

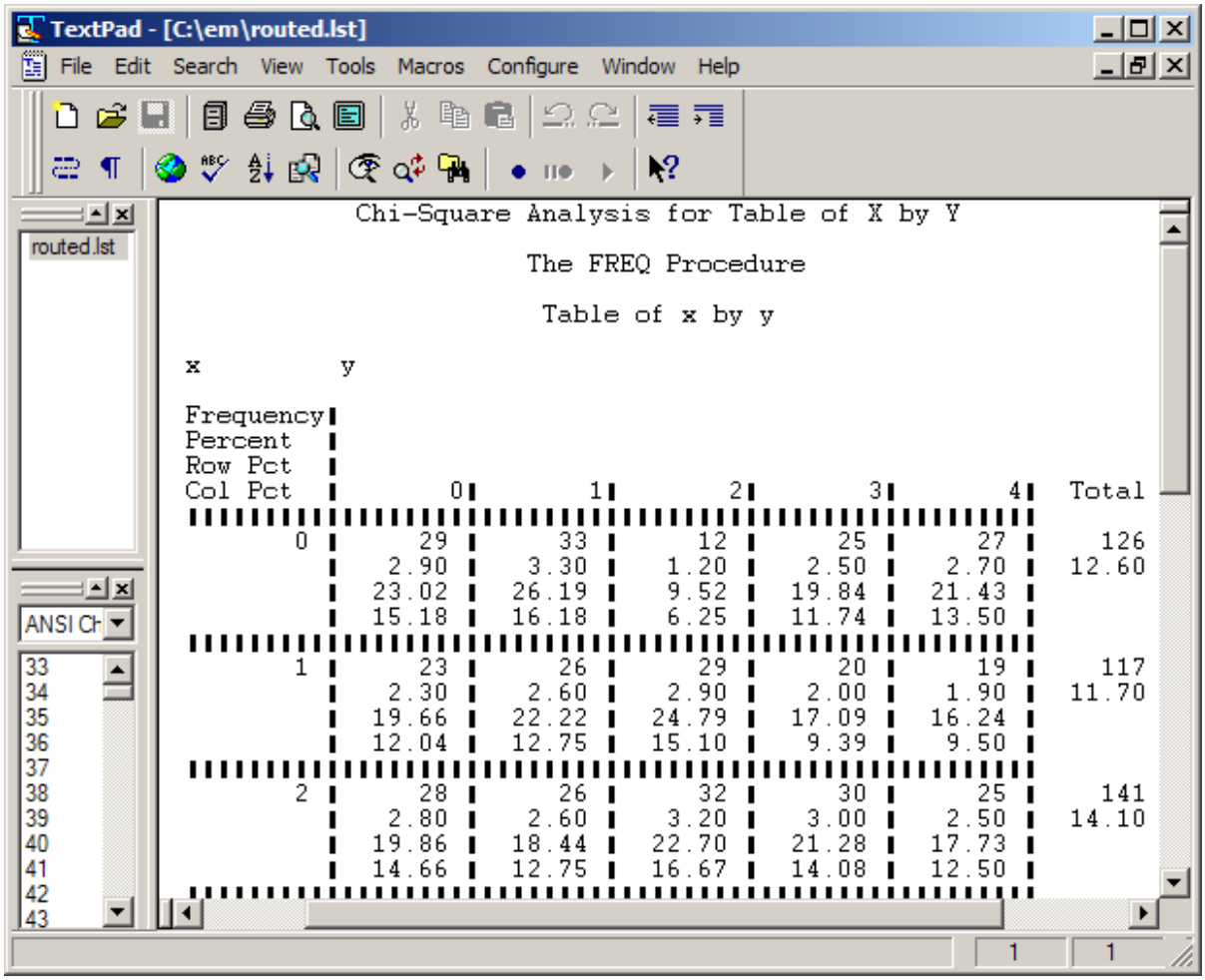
```
data probtest;
   infile routed;
   input word1 $ @;
   if word1='Chi-Squa' then
      do;
         input df chisq prob;
         keep chisq prob;
         output;
      end;
run;
```

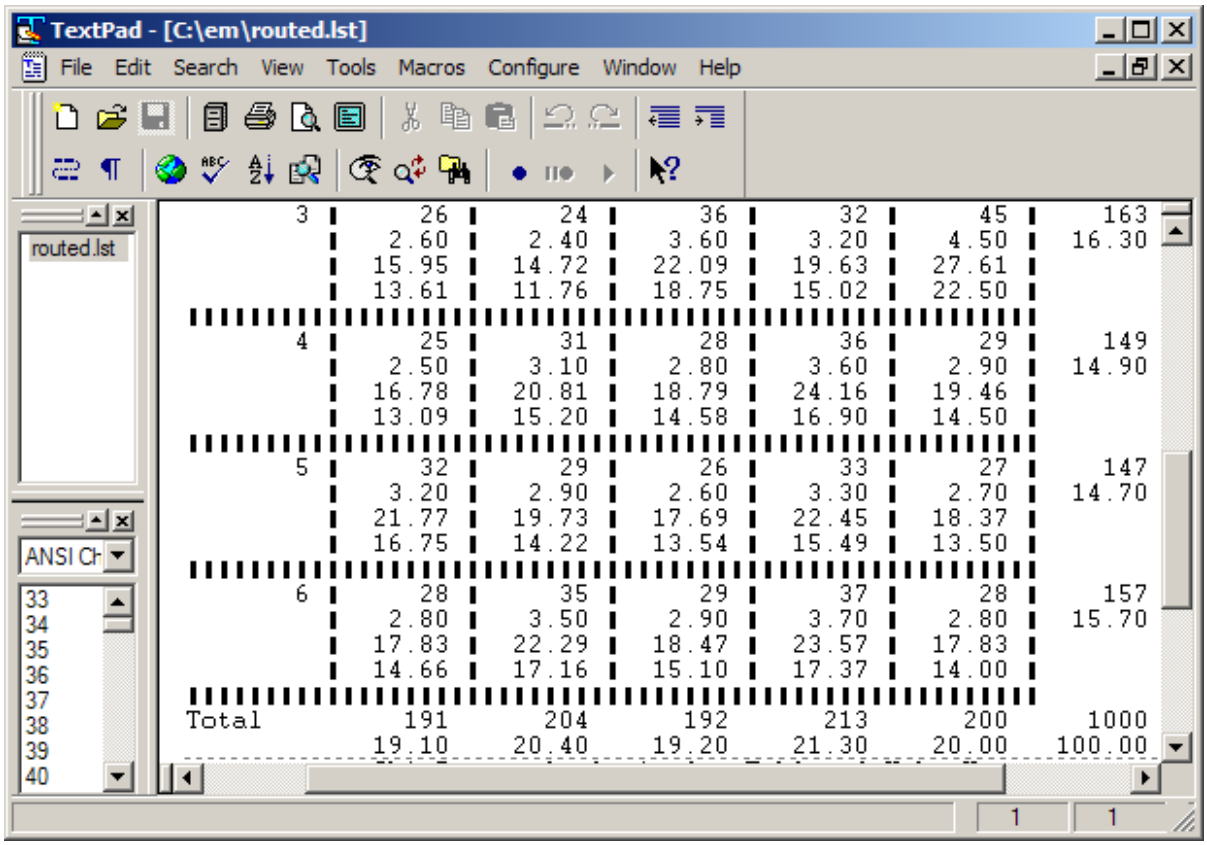
Print the PROBTEST data set. PROC PRINT produces a simple listing of data set PROBTEST. This output is routed to the default destination. See PROC PRINT Output of Data Set PROBTEST, Routed to Default Destination in the Output section.

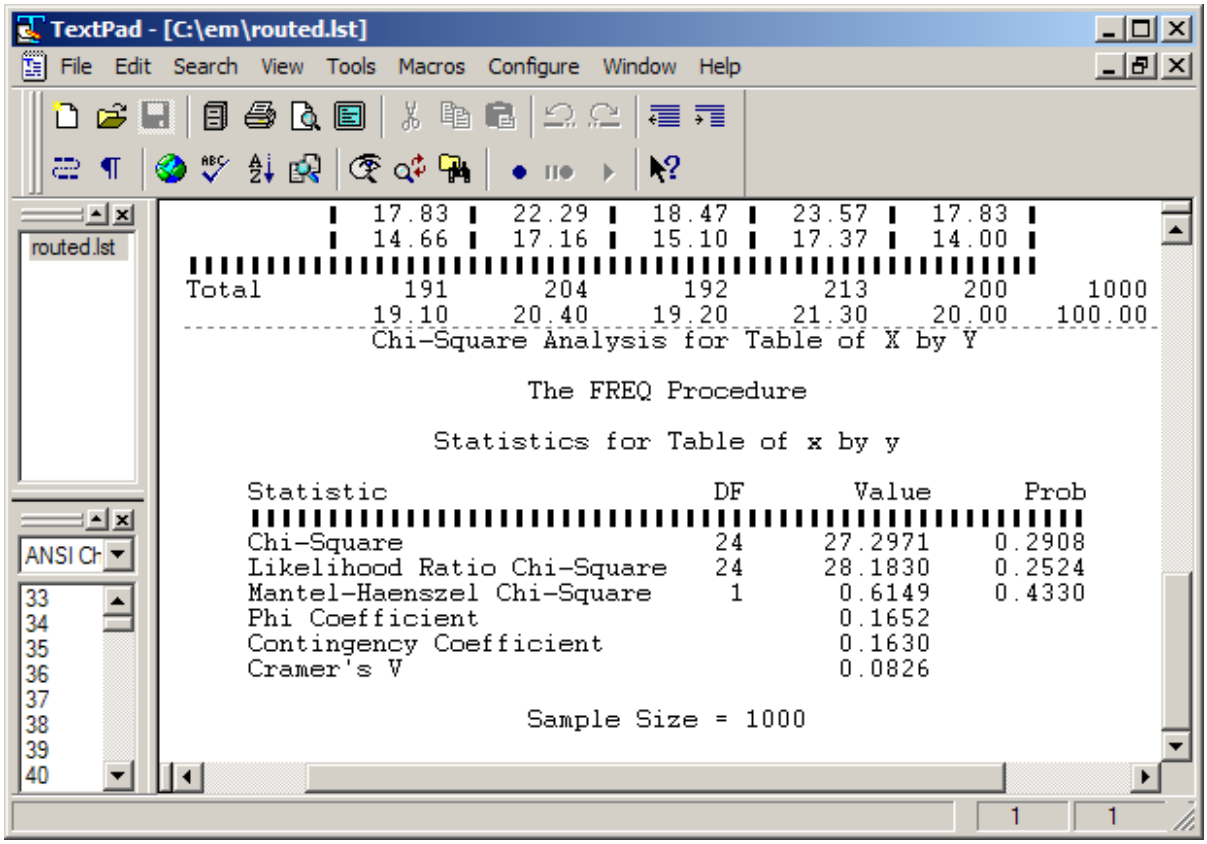
```
proc print data=probtest;
   title 'Chi-Square Analysis for Table of X by Y';
run;
```

Output

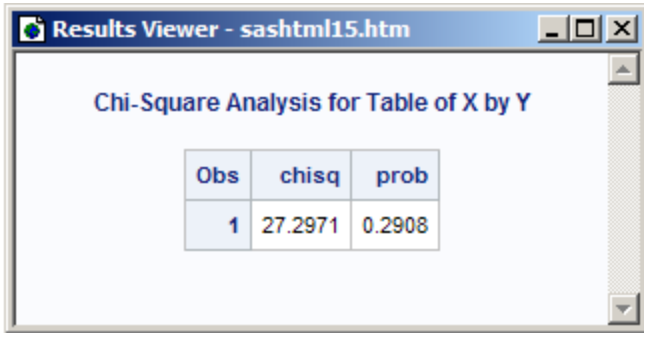
Output 38.3 *PROC FREQ Output Routed to the External File Referenced as ROUTED*




TextPad - [C:\em\routed.lst]
File Edit Search View Tools Macros Configure Window Help
routed.lst
ANSI Ch
33
34
35
36
37
38
39
40
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
1 1


TextPad - [C:\em\routed.lst]
File Edit Search View Tools Macros Configure Window Help
routed.lst
ANSI Ch
33
34
35
36
37
38
39
40
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
Chi-Square Analysis for Table of X by Y
The FREQ Procedure
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
1 1

Output 38.4 *PROC PRINT Output of Data Set PROBTEST, Routed to the Default Destination*



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;
```

Chapter 39

PROTO Procedure

Overview: PROTO Procedure

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, as well as from any SAS function, subroutine, or method block that is declared in the COMPILE procedure.

Concepts: PROTO Procedure

Registering Function Prototypes

Function prototypes are registered (declared) in the PROTO procedure. (See "Syntax " on page 1101.) Use the following form:

return-type function-name (*arg-type* <*arg-name*> / <*iotype*> <*arg-label*>, ...) <*options*>;

return-type
specifies a C language type for the returned value. See "Supported C Return Types" on page 1095.

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.

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.

arg-type
specifies the C language type for the function argument. See "Supported C Argument Types" on page 1095.

You must specify *arg-type* for each argument in the function's argument list. The argument list must be between the left and closed 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: *Arg-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.

arg-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.

Tips:

In the program code, use IOTYPE=I | O | U. The alias for IOTYPE is IO.

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 1112 for an example of how *iotype* is used.

arg-label
specifies a description or label for the argument.

Options

LABEL="*text-string*"
specifies a description or a label for the function. Enclose the text string in quotation marks.

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.

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.

Supported C Return Types

The following C return types are supported in the PROTO procedure.

Table 39.1 *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 **
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 39.2 *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 **
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 =
sasuser.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 = sasuser.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 =
sasuser.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 = sasuser.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:

```
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 available:

Table 39.3 *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)
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

Function	Description
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=sasuser.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.
- 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

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 Statement"	Register, in batch mode, external functions that are written in the C or C++ programming languages	Ex. 1
"LINK Statement"	Specify the name, path, and load module that contains your functions	
"MAPMISS Statement"	Specify alternative values, by type, to pass to functions if values are missing	

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 1112

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*n* | PACK*n*
: for Windows PC platforms only, specifies that all structures in a package be compiled with a specific N-BYTE packing pragma.

Required Argument

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.

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*n* | PACK*n*
for Windows PC platforms 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.

LINK Statement

Specifies the name, and optionally the path, of the load module that contains your functions.

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
specifies that selected libraries remain loaded when the SAS session ends.

Details

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);
```

MAPMISS Statement

Specifies alternative values, by type, to pass to functions if values are missing.

Syntax

MAPMISS <POINTER=*pointer-value*> <INT=*integer-value*> <DOUBLE=*double-value*> < LONG=*long-value*> <SHORT=*short-value*>;

Optional Arguments

POINTER=*pointer-value*
specifies the pointer value to pass to functions for pointer values that are missing. The default value is NULL.

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, 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 an 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.

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 a blank.

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 function TRIM 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.

There is no way to return and save a pointer to any type in a SAS variable (see Table 39.4 on page 1106.) 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 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 1103.

Table 39.4 *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
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 39.5 *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 **
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 CMPLIB 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.p1.test1;
    typedef struct { int a; int b; } AB_t;
    #define NUM 1;
    int p1(void);
    externc p1;
        int p1(void)
        {
            return NUM;
        }
    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 39.6 *C Helper Functions and CALL Routines*

C Helper Function or CALL Routine	Description
"ISNULL C Helper Function" on page 1109	Determines whether a pointer element of a structure is NULL
"SETNULL C Helper CALL Routine" on page 1110	Sets a pointer element of a structure to NULL
"STRUCTINDEX C Helper CALL Routine" on page 1110	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*);

where *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;
```

The program writes the following results to the SAS log:

```
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.

When you specify a variable that has a pointer value (a structure entry), then SETNULL sets the pointer to null:

```
call setnull(12.next);
```

The following example assumes that the same LINKLIST structure that is described in "ISNULL C Helper Function" on page 1109 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 arrays.

In the first part of this example, the following structures and function are defined using PROC PROTO:

```
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);
```

In the second part of this example, the PROC FCMP code segment shows how to use the STRUCTUREINDEX CALL routine to get and set each POINT structure element of an array called P in the POINT_ARRAY structure:

```
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;
```

The program writes the following results to the SAS log.

Log 39.1 *Output 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}
```

Results: PROTO Procedure

The PROTO procedure creates functions and subroutines that you can use with other SAS procedures.

Example: Splitter Function Example

Features: INT statements
KIND= prototype argument

Other features: 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=O)
       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;
```

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=O)
        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;
```

OUTPUT: LISTING

```
                                        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
```

Chapter 40

PRTDEF Procedure

Overview: PRTDEF Procedure

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 41, “PRTEXP Procedure,” on page 1129

Syntax: PRTDEF Procedure

PROC PRTDEF <*option(s)*>;

Statement	Task	Example
"PROC PRTDEF Statement"	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 1122

"Example 2: Creating a Ghostview Printer in SASUSER to Preview PostScript Printer Output in SASUSER" on page 1123

"Example 3: Creating a Single Printer Definition That Is Available to All Users" on page 1124

"Example 4: Adding, Modifying, and Deleting Printer Definitions" on page 1125

"Example 5: Deleting a Single Printer Definition" on page 1126

Syntax

PROC PRTDEF <*option(s)*>;

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: PRTDEF Procedure" on page 1117

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 1126

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 1123

"Example 4: Adding, Modifying, and Deleting Printer Definitions" on page 1125

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 1123

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 1124

Input Data Set: PRTDEF Procedure

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 the optional variables for this data set.

Table 40.1 *Required and Optional Variable for Creating Printer Definition Records*

Variable Name	Variable Description
Required	
DEST	Destination
DEVICE	Device
MODEL	Prototype

Variable Name	Variable Description
NAME	Printer name
Optional	
BOTTOM	Default bottom margin
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                         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                         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;
```

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 = " ";

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 = " ";
```

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;
```

Example 3: Creating a Single Printer Definition That Is Available to All Users

Features: PROC PRTDEF statement options:
DATA=
USESASHELP

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                            DISK    sasprt.pcl
del  Gray PostScript    PostScript Level 2 (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 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 Gray PostScript and 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                            DISK    sasprt.pcl
del  Gray PostScript    PostScript Level 2 (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 list;
run;
```

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;
```

Chapter 41

PRTEXP Procedure

Overview: PRTEXP Procedure

The PRTEXP procedure enables you to extract printer attributes from the SAS registry for replication and modification. 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.

See Also

Chapter 40, “PRTDEF Procedure,” on page 1115

Concepts: PRTEXP Procedure

The PRTEXP procedure, along with the PRTDEF procedure, can replicate, modify, and create printer definitions either for an individual user or for all SAS users at your site. PROC PRTEXP can extract only the attributes that are used to create printer definitions from the registry. If you write them to a SAS data set, then 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: PRTDEF Procedure” on page 1117.

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<*option(s)*>;
<SELECTprinter_1 …<printer_n>>;
<EXCLUDEprinter_1 … <printer_n>>;

Statement	Task	Example
"PROC PRTEXP Statement"	Obtain printer attributes from the SAS registry	Ex. 1, Ex. 2
"EXCLUDE Statement"	Obtain printer attributes for the specified printers	
"SELECT Statement"	Obtain printer attributes for all printers except 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 1131
"Example 2: Writing Attributes to a SAS Data Set" on page 1132

Syntax

PROC PRTEXP<*option(s)*>;

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 to create 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_1* … <*printer_n*>;

Required Argument

printer_1 <printer_n>
specifies the 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 1131
"Example 2: Writing Attributes to a SAS Data Set" on page 1132

Syntax

SELECT *printer_1* … <*printer_n*>;

Required Argument

printer_1 <printer_n>
specifies the 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

```
proc prtexp usesashelp;
select postscript;
run;
```

Program Description

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;
```

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

```
proc prtexp out=PRDVTER;
select pcl4 pcl5 pcl5e pcl5c;
run;
```

Program Description

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;
```

Chapter 42

PWENCODE Procedure

Overview: PWENCODE Procedure

The PWENCODE procedure enables you to encode passwords. Encoded passwords can be used in place of plaintext passwords in SAS programs that access relational database management systems (RDBMSs) and various servers, such as SAS/CONNECT servers, SAS/SHARE servers, and SAS Integrated Object Model (IOM) servers (such as the SAS Metadata Server).

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 about SAS

naming rules, see “Rules for Most SAS Names” in Chapter 3 of *SAS Language Reference: Concepts*.

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

PROC PWENCODE uses encoding to disguise passwords. With encoding, one character set is translated to another character set through some form of table lookup. 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. PROC PWENCODE is intended to prevent casual, non-malicious viewing of passwords. You should not depend on PROC PWENCODE for all your data security needs; a determined and knowledgeable attacker can decode the encoded passwords.

Syntax: PWENCODE Procedure

PROC PWENCODE IN='*password*' <OUT=*fileref*> <METHOD=*encoding-method*>;

Statement	Task	Example
“PROC PWENCODE Statement”	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 1136
“Example 2: Using an Encoded Password in a SAS Program” on page 1136
“Example 3: Saving an Encoded Password to the Paste Buffer” on page 1138
“Example 4: Specifying an Encoding Method for a Password” on page 1139

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

about SAS naming rules, see "Rules for Most SAS Names" in Chapter 3 of *SAS Language Reference: Concepts*.

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 the SAS Constants in Expressions chapter of *SAS Language Reference: Concepts*.

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 1136

"Example 2: Using an Encoded Password in a SAS Program" on page 1136

"Example 3: Saving an Encoded Password to the Paste Buffer" on page 1138

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: Using an Encoded Password in a SAS Program" on page 1136

METHOD=*encoding-method*

specifies the encoding method. Here are the supported values for *encoding-method*:

Table 42.1 *Supported Encoding Methods*

Encoding Method	Description	Supported Data Encryption Algorithm
sas001	Uses base64 to encode passwords.	None
sas002, which can also be specified as **sasenc**	Uses a 32-bit key to encode passwords. This is the default.	SASProprietary, which is included in SAS software.
sas003	Uses a 256-bit key to encode passwords.	AES (Advanced Encryption Standard), which is supported in SAS/SECURE.

Note: SAS/SECURE is an add-on product that requires a separate license. For details about SAS/SECURE, the SASProprietary algorithm, and the AES algorithm, see *Encryption in SAS*.

If the METHOD= option is omitted, the default encoding method is used. When the FIPS 140-2 compliance option, -encryptfips, is specified, the encoding default method is **sas003**. For all other cases, encoding method **sas002** is the default method used.

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

```
proc pwencode in='my password';
run;
```

Program Description

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: Using an Encoded Password in a SAS Program

Features: IN= argument
OUT= option

Details

This example illustrates the following:

- encoding a password and saving it to an external file
- reading the encoded password with a DATA step, storing it in a macro variable, and using it in a SAS/ACCESS LIBNAME statement

Program 1: Encoding the Password

```
filename pwfile
'external-filename'

proc pwencode in='mypass1' out=pwfile;
run;
```

Program Description

Declare a fileref.

```
filename pwfile
'external-filename'
```

Encode the password and write it to the external file. The OUT= option specifies which external fileref the encoded password will be written to.

```
proc pwencode in='mypass1' out=pwfile;
run;
```

Program 2: Using the Encoded Password

```
filename pwfile
'external-filename';

options symbolgen;

data _null_;
infile pwfile truncover;
input line :$50.;
call symputx('dbpass',line);
run;

libname x odbc dsn=SQLServer user=testuser password="&dbpass";
```

Program Description

Declare a fileref for the encoded-password file.

```
filename pwfile
'external-filename';
```

Set the SYMBOLGEN SAS system option. The purpose of this step is to show that the actual password cannot be revealed, even when the macro variable that contains the encoded password is resolved in the SAS log. This step is not required in order for the program to work properly.

```
options symbolgen;
```

Read the file and store the encoded password in a macro variable. The DATA step stores the encoded password in the macro variable DBPASS.

```
data _null_;
infile pwfile truncover;
input line :$50.;
call symputx('dbpass',line);
run;
```

Use the encoded password to access a DBMS. You must use double quotation marks (“ ”) so that the macro variable resolves properly.

```
libname x odbc dsn=SQLServer user=testuser password="&dbpass";
```

Log

```
1    filename pwfile 'external-filename';
2    options symbolgen;
3    data _null_;
4    infile pwfile truncover;
5    input line :$50.;
6    call symputx('dbpass',line);
7    run;

NOTE: The infile PWFILE is:
      Filename=external-filename
      RECFM=V,LRECL=256,File Size (bytes)=4,
      Last Modified=12Apr2012:13:23:49,
      Create Time=12Apr2012:13:23:39

NOTE: 1 record was read from the infile PWFILE.
      The minimum record length was 4.
      The maximum record length was 4.
NOTE: DATA statement used (Total process time):
      real time           0.57 seconds
      cpu time            0.04 seconds
8
9   libname x odbc
SYMBOLGEN:  Macro variable DBPASS resolves to {sas002}bXlwYXNzMQ==
9 !               dsn=SQLServer user=testuser password="&dbpass";
NOTE: Libref X was successfully assigned as follows:
      Engine:        ODBC
      Physical Name: SQLServer
```

Example 3: Saving an Encoded Password to the Paste Buffer

Features: IN= argument
OUT= option

Other features: 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 4: Specifying an Encoding Method for a Password

Features: 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.

Program

```
proc pwencode in='my password' method=sas003;
run;
```

Program Description

Encode the password.

```
proc pwencode in='my password' method=sas003;
run;
```

Log

Note that each character of the password is replaced by an X in the SAS log.

```
8   proc pwencode in=XXXXXXXXXXXXX method=sas003;
29   run;

{SAS003}08D7B93810D390916F615117D71B2639B4BE

NOTE: PROCEDURE PWENCODE used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

Chapter 43

QDEVICE Procedure

Overview: QDEVICE Procedure

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.

Concepts: QDEVICE Procedure

About Universal Previewers

A universal previewer is a special form of a universal printer that is used only to preview printer output. Examples of universal previewers are GhostView to view PostScript output or Adobe Acrobat Reader to view PDF output.

In the QDEVICE procedure, the reports that are available or unavailable for universal printers are also available or unavailable for universal previewers.

Reports for Windows Operating Environments

By default, SAS starts on Windows using the NOUNIVERSALPRINT (NOUPRINT) system option in order to use Windows printing. The SYSPRINT= system option determines the default Windows printer. Because printers on Windows are associated with a SAS printer interface device, reports that you create for the default Windows printer have a device or printer name of one of the following SAS printer interface devices:

- WINPRTC (color)
- WINPRTG (gray scale)
- WINPRTM (monochrome)

If SAS starts with Universal Printing active on Windows, the default printer report is for the default universal printer and not a Windows printer.

See “Example 4: Generate a Report for the Default Printer” on page 1173 for example reports.

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 | LINESTYLE | RECTANGLE | SYMBOL>
<OUT=*SAS-data-set*>
<DEVLOC = GDEVICE0...GDEVICE9 | SASHELP>
<REGISTRY = SASHELP | SASUSER>
<SUPPORT = YES | NO | ALL>
<UNITS = IN | CM>;
DEVICE *device-name-1 <...device-name-n>*
<_ALL_> <_HTML_> <_LISTING_> <_RTF_>;
PRINTER *printer-name-1 <...printer-name-n>*
<_ALL_> <_PCL_> <_PDF_> <_PRINTER_> <_PS_>;
VAR *variable-1 <…variable-n>*;

Statement	Task	Example
"PROC QDEVICE Statement"	Specifies 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
"DEVICE Statement"	Specify which SAS/GRAPH devices to generate a report for	Ex. 2
"PRINTER Statement"	Specify which universal printers to generate a report for	Ex. 6, Ex. 3
"VAR Statement"	Specify the information (variables) to include in the generated reports	Ex. 5

PROC QDEVICE Statement

Specifies an (optional) output data set, which report to generate, which locations to search, whether to report supported, non-supported, or all features, and sizing units for GENERAL reports.

Syntax

PROC QDEVICE

<REPORT = GENERAL | FONT | DEVOPTION | LINESTYLE | RECTANGLE | SYMBOL>

<OUT=*SAS-data-set*>

<DEVLOC = GDEVICE0...GDEVICE9 | SASHELP>

<REGISTRY = SASHELP | SASUSER>

<SUPPORT = YES | NO | ALL>

<UNITS = IN | CM>;

Summary of Optional Arguments

DEVLOC = GDEVICE0...GDEVICE9 | SASHELP
: specifies which libraries to search when querying a device.

OUT = *SAS-data-set*
: specifies an output SAS data set for the report.

REGISTRY = SASHELP | SASUSER
: specifies which portion of the SAS registry to search when querying a universal printer.

REPORT = DEVOPTION | FONT | GENERAL | LINESTYLE | 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, centimeters, pixels-per-inch, or pixels-per-centimeter in the GENERAL report.

Optional Arguments

DEVLOC = GDEVICE0...GDEVICE9 | SASHELP
specifies which libraries to search when querying a device. If you do not specify the DEVLOC= option, libraries are searched in the following order:

1. GDEVICE0 to GDEVICE9
2. SASHELP

OUT = *SAS-data-set*
specifies an output SAS data set for the report.

Default: SAS Log

REGISTRY = SASHELP | SASUSER
specifies which portion of the SAS registry to search when querying a universal printer. If you do not specify the REGISTRY= option, both SASUSER and SASHELP are searched.

Alias: REG

REPORT = DEVOPTION | FONT | GENERAL | LINESTYLE | RECTANGLE | SYMBOL
specifies the type of report that you want to generate. You can request only one type of report. See "Variables Valid for All Reports" on page 1148 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 Chapter 13 of *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.

LINESTYLE
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 "Variables Valid for All Reports" on page 1148 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 only applies to devices when producing a DEVOPTION, LINESTYLE, RECTANGLE, or SYMBOL report.

UNITS = IN | CM
specifies whether the values for 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 only applies 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-name1...device-name-n*>
<_ALL_>
<_HTML_>
<_LISTING_>
<_RTF_>;

Optional Arguments

device-name1...device-name-n
specifies the device for which you want to generate a report. Separate device names with a blank space.

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-name1...printer-name-n*>
<_ALL_>
<_PCL_>
<_PDF_>
<_PRINTER_>
<_PS_>;

Optional Arguments

printer-name1...printer-name-n
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.

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.

Windows specifics: By default, SAS uses Windows printing and not Universal Printing. When SAS uses Windows printing, the report that is generated when you specify the _PRINTER_ argument has information for the SAS printer interface device that is associated with the default Windows printer. The default Windows printer is specified by the SYSPRINT= system option. The SAS printer interface devices are WINPRTC (color), WINPRTG (gray scale), or WINPRTM (monochrome).

_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, in the report.

Tip: If you specify the VAR statement, you must specify at least one variable. Otherwise, the statement is ignored.

Syntax

VAR *variable1*<*...variable-n*>;

Variables Valid for All Reports

DESC
displays the default description of the device or printer.

LOCATION
displays the physical location of the GDEVICE0-DEVICE9 or SASHELP library containing the DEVICES catalog where the device entry was found. 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.

NAMETYPE
displays the type of device or printer:

- Graph Device
- Shortcut Device
- System Printer
- System Display
- System Metafile
- Universal Previewer
- Universal Printer

See: "Device Categories and Modifying Default Output Attributes" in Chapter 6 of *SAS/GRAPH: Reference*

DEVOPTION Report Variables

For more information, see "Creating a DEVOPTION Report" on page 1162.

BIT
displays the bit position in the DEVOPTS string for the corresponding device option.

See: "DEVOPTS" in *SAS/GRAPH: Reference*

BITSTRING
displays the bit pattern of the corresponding device option.

See: "DEVOPTS" in *SAS/GRAPH: Reference*

DESC
displays the default description of the device or printer.

LOCATION
displays the physical location of the GDEVICE0-DEVICE9 or SASHELP library containing the DEVICES catalog where the device entry was found. 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.

NAMETYPE
displays the type of device or printer:

- Graph Device
- Shortcut Device
- System Printer
- System Display
- System Metafile
- Universal Previewer
- Universal Printer

See: "Device Categories and Modifying Default Output Attributes" in Chapter 6 of *SAS/GRAPH: Reference*

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.

PROTOTYPE
displays the prototype (model) that was used to define the universal printer.

Note: The PROTOTYPE variable has a value for the DEVOPTION report only for shortcut devices or printer interface devices such as WINPRTC, WINPRTG, WINPRTM, SASPRTC, SASPRTG, or SASPRTM. These devices are interfaces to universal printers only when Universal Printing is in effect.

See: "Universal Printing" in Chapter 15 of *SAS Language Reference: Concepts* and "Define a New Printer" in Chapter 15 of *SAS Language Reference: Concepts*

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.

FONT Report Variables

For more information, see "Creating a FONT Report" on page 1160.

DESC
displays the default description of the device or printer.

FONT
displays the name of the default font.

See:
"Default Fonts" in Chapter 13 of *SAS/GRAPH: Reference*
"Variable Labels in a FONT Report" on page 1161

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 display 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" in *SAS/GRAPH: Reference* for more information.

See: "Variable Labels in a FONT Report" on page 1161

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 display 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" in *SAS/GRAPH: Reference* for more information.

LOCATION
displays the physical location of the GDEVICE0-DEVICE9 or SASHELP library containing the DEVICES catalog where the device entry was found. 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.

NAMETYPE
displays the type of device or printer:

- Graph Device
- Shortcut Device
- System Printer
- System Display
- System Metafile
- Universal Previewer
- Universal Printer

See: "Device Categories and Modifying Default Output Attributes" in Chapter 6 of *SAS/GRAPH: Reference*

PROTOTYPE
displays the prototype (model) that was used to define the universal printer.

See: "Universal Printing" in Chapter 15 of *SAS Language Reference: Concepts* and "Define a New Printer" in Chapter 15 of *SAS Language Reference: Concepts*

GENERAL Report Variables

For more information, see "Creating a GENERAL Report" on page 1157.

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 Chapter 14 of *SAS/GRAPH: Reference*

COLS
displays the number of horizontal columns in the output.

See: "Cells" in Chapter 5 of *SAS/GRAPH: Reference*

COMPRESSION
indicates the compression method used, if compression is supported by device or printer. If compression is not supported, the value is NONE.

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" in *SAS/GRAPH: Reference* for more information.

FONT
displays the name of the default font.

See: "Default Fonts" in Chapter 13 of *SAS/GRAPH: Reference*

FORMAT
displays the output format type. For example, EMF, PostScript, GIF, Host Display, and so on.

See: "Commonly Used Devices" in Chapter 6 of *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" in *SAS/GRAPH: Reference* for more information.

FWEIGHT
displays the weight of the default font. For example, Normal, Medium, and so on.

Note: If the font name is acquired from a CHARREC list in a device entry, the weight is not available. See "CHARREC" in *SAS/GRAPH: Reference* for more information.

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 Native Devices" in Chapter 9 of *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 Statements: Reference* and "DEVTYPE" in *SAS/GRAPH: Reference*

LEFT
displays the size of the left margin of output.

LMIN
displays the minimum left margin.

LOCATION
displays the physical location of the GDEVICE0-DEVICE9 or SASHELP library containing the DEVICES catalog where the device entry was found. 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 Chapter 14 of *SAS/GRAPH: Reference*

NAME
displays the name of the device or printer.

NAMETYPE
displays the type of device or printer:

- Graph Device
- Shortcut Device
- System Printer
- System Display
- System Metafile
- Universal Previewer
- Universal Printer

See: "Device Categories and Modifying Default Output Attributes" in Chapter 6 of *SAS/GRAPH: Reference*

PROTOTYPE
displays the prototype (model) that was used to define the universal printer.

See: "Universal Printing" in Chapter 15 of *SAS Language Reference: Concepts* and "Define a New Printer" in Chapter 15 of *SAS Language Reference: Concepts*

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 Chapter 5 of *SAS/GRAPH: Reference*

TMIN
displays the minimum top margin of output.

TOP
displays the size of the top margin.

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.

Interactions:
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 Native Devices" in Chapter 9 of *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" in *SAS/GRAPH: Reference* and "Using the XPIXELS=, XMAX=, YPIXELS=, and YMAX= Graphics Options to Set the Resolution for the Native Devices" in Chapter 9 of *SAS/GRAPH: Reference*

YPIXELS
displays the height of the output in pixels.

See: "YPIXELS" in *SAS/GRAPH: Reference* and "Using the XPIXELS=, XMAX=, YPIXELS=, and YMAX= Graphics Options to Set the Resolution for the Native Devices" in Chapter 9 of *SAS/GRAPH: Reference*

LINESTYLE Report Variables

For more information, see "Creating a LINESTYLE Report" on page 1166.

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: Figure 16.22, "Line Types," in *SAS/GRAPH: Reference*

LOCATION
displays the physical location of the GDEVICE0-DEVICE9 or SASHELP library containing the DEVICES catalog where the device entry was found. 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.

NAMETYPE
displays the type of device or printer:

- Graph Device
- Shortcut Device
- System Printer
- System Display
- System Metafile
- Universal Previewer
- Universal Printer

See: "Device Categories and Modifying Default Output Attributes" in Chapter 6 of *SAS/GRAPH: Reference*

PROTOTYPE
displays the prototype (model) that was used to define the universal printer.

Note: The PROTOTYPE variable has a value for the LINESTYLE report only for shortcut devices or printer interface devices such as WINPRTC, WINPRTG, WINPRTM, SASPRTC, SASPRTG, or SASPRTM. These devices are interfaces to universal printers only when Universal Printing is in effect.

See: "Universal Printing" in Chapter 15 of *SAS Language Reference: Concepts* and "Define a New Printer" in Chapter 15 of *SAS Language Reference: Concepts*

SUPPORT
displays the device or printer 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.

RECTANGLE Report Variables

For more information, see "Creating a RECTANGLE Report" on page 1167.

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 containing the DEVICES catalog where the device entry was found. 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.

NAMETYPE
displays the type of device or printer:

- Graph Device
- Shortcut Device
- System Printer
- System Display
- System Metafile
- Universal Previewer
- Universal Printer

See: "Device Categories and Modifying Default Output Attributes" in Chapter 6 of *SAS/GRAPH: Reference*

PROTOTYPE
displays the prototype (model) that was used to define the universal printer.

Note: The PROTOTYPE variable has a value for the RECTANGLE report only for shortcut devices or printer interface devices such as WINPRTC, WINPRTG, WINPRTM, SASPRTC, SASPRTG, or SASPRTM. These devices are interfaces to universal printers only when Universal Printing is in effect.

See: "Universal Printing" in Chapter 15 of *SAS Language Reference: Concepts* and "Define a New Printer" in Chapter 15 of *SAS Language Reference: Concepts*

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.

SYMBOL Report Variables

For more information, see "Creating a SYMBOL Report" on page 1168.

DESC
displays the default description of the device or printer.

LOCATION
displays the physical location of the GDEVICE0-DEVICE9 or SASHELP library containing the DEVICES catalog where the device entry was found. 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.

NAMETYPE
displays the type of device or printer:

- Graph Device
- Shortcut Device
- System Printer
- System Display
- System Metafile
- Universal Previewer
- Universal Printer

See: "Device Categories and Modifying Default Output Attributes" in Chapter 6 of *SAS/GRAPH: Reference*

PROTOTYPE
displays the prototype (model) that was used to define the universal printer.

Note: The PROTOTYPE variable has a value for the SYMBOL report only for shortcut devices or printer interface devices such as WINPRTC, WINPRTG, WINPRTM, SASPRTC, SASPRTG, or SASPRTM. These devices are interfaces to universal printers only when Universal Printing is in effect.

See: "Universal Printing" in Chapter 15 of *SAS Language Reference: Concepts* and "Define a New Printer" in Chapter 15 of *SAS Language Reference: Concepts*

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" in *SAS/GRAPH: Reference* and "SYMBOLS" in *SAS/GRAPH: Reference*

Variables Common to All Reports

You can use the following variables in any of the reports:

- NAME
- DESC
- NAMETYPE
- LOCATION
- PROTOTYPE

For a description of the variables, see "Variables Valid for All Reports" on page 1148.

Creating 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 1150.

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 that 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
NAMETYPE	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

Variable	SAS Log Label	Output Data Set Label
FONT	Default Typeface	FONT TYPEFACE DEFAULT
FSTYLE	Font Style	FONT STYLE DEFAULT
FWEIGHT	Font Weight	FONT WEIGHT DEFAULT
FHEIGHT	Font Height	FONT HEIGHT DEFAULT
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 *	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 Method	COMPRESSION METHOD
EMBEDDING	Font Embedding	FONT EMBEDDING SUPPORT

* If the type of units displays with the value of a variable, such as 0 inches for the left margin, the Units label does not display in 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:

```
134  proc qdevice;
135     printer svg;
136  run;

                 Name: SVG
          Description: Scalable Vector Graphics 1.1
                 Type: Universal Printer
             Registry: SASHELP
            Prototype: SVG 1.1
     Default Typeface: Cumberland AMT
           Font Style: Regular
          Font Weight: Normal
          Font Height: 8 points
       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 Method: None
       Font Embedding: Option
```

Creating 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 1149.

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 that they appear in the table.

Variable	SAS Log Label	Output Data Set Label
NAME	Output data set	NAME OF DEVICE OR PRINTER

Variable	SAS Log Label	Output Data Set Label
DESC	Description	DESCRIPTION OF DEVICE OR PRINTER
NAMETYPE	Type	TYPE OF DEVICE OR PRINTER
LOCATION	Device Catalog for a device Registry for a printer	LOCATION OF DEVICE OR PRINTER DEFINTION
PROTOTYPE	Prototype	PRINTER PROTOTYPE
FONT	depends on the font or type	FONT TYPEFACE
FTYPE	depends on the type of font	FONT TYPE
FSTYLE	none	FONT STYLE
FWEIGHT	none	FONT WEIGHT

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, or FWEIGHT, the SAS log reports only the font type labels and the font family names. The variable 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.

Font styles and weights are not reported to the SAS log even if the FSTYLE and FWEIGHT variables are specified in the VAR statement.

Example: FONT Report

The following QDEVICE procedure creates a FONT report for the ACTIVEX graphics device:

```
proc qdevice report=font;
   device activex;
run;
```

Here is a partial FONT report in the SAS log:

```
41   proc qdevice report=font;
42   device activex;
43   run;

                    Name: ACTIVEX
             Description: ActiveX enabled GIF Driver
                    Type: Graph Device
          Device Catalog: ( 'your-font-catalog-path' )
Supported Font Typefaces: System (7x16) 10pt
                          Terminal (8x12) 9pt
                          Terminal (4x6) 5pt
                          Terminal (5x12) 9pt
                          Terminal (6x8) 6pt
                          Terminal (7x12) 9pt
                          Terminal (10x18) 14pt
                          Terminal (12x16) 12pt
                          Fixedsys (8x15) 9pt
                          Roman
                          Script
                          Modern
```

Creating 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 1148.

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 that they appear in the table.

Variable	SAS Log Label	Output Data Set Label
NAME	Output data set	NAME OF DEVICE OR PRINTER
DESC	Description	DESCRIPTION OF DEVICE OR PRINTER
NAMETYPE	Type	TYPE OF DEVICE OR PRINTER
LOCATION	Device Catalog	LOCATION OF DEVICE OR PRINTER DEFINTION
PROTOTYPE	Prototype	PRINTER PROTOTYPE

Variable	SAS Log Label	Output Data Set Label
BIT	Bit Position	DEVICE OPTION BIT POSITON
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 SASEMF graphics device, reporting supported options because the default for SUPPORT is YES:

```
proc qdevice report=devoption;
   device sasemf;
run;
```

Here is the report in the SAS log:

```
104  proc qdevice report=devoption;
105     device sasemf;
106  run;

NOTE: Writing HTML Body file: sashtml.htm
              Name: SASEMF
       Description: Enhanced Metafile Driver
              Type: Graph Device
    Device Catalog:
( 'C:\SASv9\sasgen\dev\mva-v930\sas_dvd\sio\dntno\en\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: 2
       Bit Pattern: 2000000000000000
     Device Option: GDXCHR
Option Description: Hardware has scalable fonts available
           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

      Bit Position: 11
       Bit Pattern: 0010000000000000
     Device Option: GDLWIDTH
Option Description: Hardware can draw lines of varying widths
           Support: Yes

      Bit Position: 14
       Bit Pattern: 0002000000000000
     Device Option: GDHRDCHR
Option Description: Hardware characters are supported by the device
           Support: Yes
```

```
      Bit Position: 18
       Bit Pattern: 0000200000000000
     Device Option: GDTXJUSTIFY
Option Description: Hardware is capable of justifying proportional text
           Support: Yes

      Bit Position: 19
       Bit Pattern: 0000100000000000
     Device Option: GDSTREAM
Option Description: Driver is capable of producing a device dependent catalog
entry
           Support: Yes

      Bit Position: 20
       Bit Pattern: 0000080000000000
     Device Option: GDCBACK
Option Description: Device cannot draw in the default background color
           Support: Yes

      Bit Position: 24
       Bit Pattern: 0000008000000000
     Device Option: GDUNICODE
Option Description: Device supports the use of the Unicode font attribute
           Support: Yes

      Bit Position: 25
       Bit Pattern: 0000004000000000
     Device Option: GDPOLYLINE
Option Description: Hardware is capable of supporting polylines
           Support: Yes

      Bit Position: 28
       Bit Pattern: 0000000800000000
     Device Option: GDTRUETYPE
Option Description: Device supports the use of TrueType fonts
           Support: Yes

      Bit Position: 35
       Bit Pattern: 0000000010000000
     Device Option: GDTARGETFONT
Option Description: Device supports target fonts
           Support: Yes

      Bit Position: 36
       Bit Pattern: 0000000008000000
     Device Option: GDIMAGE
Option Description: Device is capable of drawing images
           Support: Yes

      Bit Position: 37
       Bit Pattern: 0000000004000000
     Device Option: GDMULTICMAP
Option Description: Device supports multiple color maps
           Support: Yes

      Bit Position: 44
       Bit Pattern: 0000000000080000
     Device Option: GDTEXTCLIP
Option Description: Hardware will clip text at the device limits
           Support: Yes
```

Creating a LINESTYLE Report

The LINESTYLE report produces a report of the hardware (dashed) line styles that are supported by the specified device.

About LINESTYLE Report Variables

For a description of the variables, see "LINESTYLE Report Variables" on page 1154.

The following table lists the variables that you can use in a LINESTYLE 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 that they appear in the table.

Variable	SAS Log Label	Output Data Set Label
NAME	Output data set	NAME OF DEVICE OR PRINTER
DESC	Description	DESCRIPTION OF DEVICE OR PRINTER
NAMETYPE	Type	TYPE OF DEVICE OR PRINTER
LOCATION	Device Catalog for a device Registry for a printer	LOCATION OF DEVICE OR PRINTER DEFINTION
PROTOTYPE	Prototype	PRINTER PROTOTYPE
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 lj5ps;
run;
```

Here is the LINESTYLE report in the SAS log:

```
202  proc qdevice report=linestyle;
203     device lj5ps;
204  run;

                Name: LJ5PS
         Description: LaserJet 5P -- 600 dpi -- PostScript
                Type: Graph Device
      Device Catalog: ( 'your-sas-path\sashelp' )
Supported Line Styles: 0-43
```

Creating 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 1155.

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 that they appear in the table.

Variable	SAS Log Label	Output Data Set Label
NAME	Output data set	NAME OF DEVICE OR PRINTER
DESC	Description	DESCRIPTION OF DEVICE or PRINTER
NAMETYPE	Type	TYPE OF DEVICE OR PRINTER
LOCATION	Device Catalog for a device Registry for a printer	LOCATION OF DEVICE OR PRINTER DEFINTION
PROTOTYPE	Prototype	PRINTER PROTOTYPE
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 sasprtg;
run;
```

Here is the RECTANGLE report in the SAS log:

```
205  proc qdevice report=rectangle;
206     device sasprtg;
207  run;

                    Name: SASPRTG
             Description: Generic PostScript Level 1 Printer
                    Type: Universal Printer
          Device Catalog: ( 'your-sas-path\sashelp' )
               Prototype: PostScript Level 1 (Color)
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 Generic PostScript Level 1 Printer as a universal printer.

Creating 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 1156.

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 that they appear in the table.

Variable	SAS Log Label	Output Data Set Label
NAME	Output data set	NAME OF DEVICE OR PRINTER
DESC	Description	DESCRIPTION OF DEVICE OR PRINTER
NAMETYPE	Type	TYPE OF DEVICE OR PRINTER

Variable	SAS Log Label	Output Data Set Label
LOCATION	Device Catalog for a device Registry for a printer	LOCATION OF DEVICE OR PRINTER DEFINTION
PROTOTYPE	Prototype	PRINTER PROTOTYPE
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:

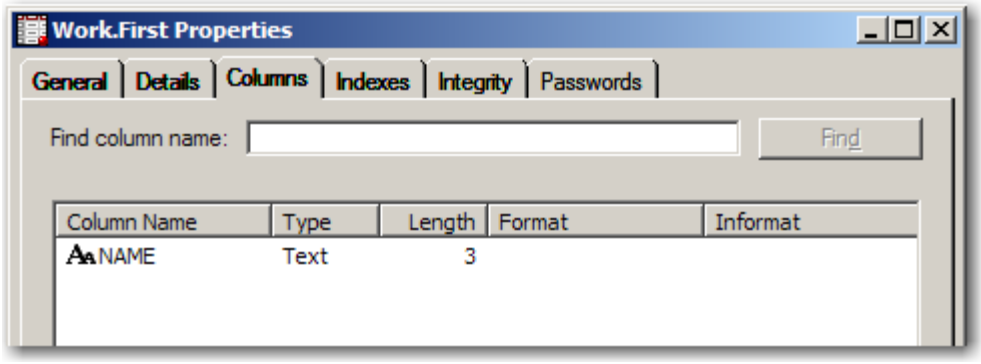
```
235  proc qdevice report=symbol;
236     device cgm;
237  run;

                      Name: CGM
               Description: CGM generator--binary output
                      Type: Graph Device
            Device Catalog:
( 'C:\SASv9\sasgen\dev\mva-v930\sas_dvd\sio\dntno\en\sashelp' )
Supported Hardware Symbols: Plus,X,Star
```

Character Variable Lengths in Report Output Data Sets

When you create an output data set report, the length of the character variables are not always a fixed length. The length of each character variable is determined by the maximum length of the data for each variable in the report. In the following examples, the length of the NAME variable is 3 in the first data set. In the second data set, the length is 6.

```
proc qdevice out=first;
   device emf;
   var name;
run;
```

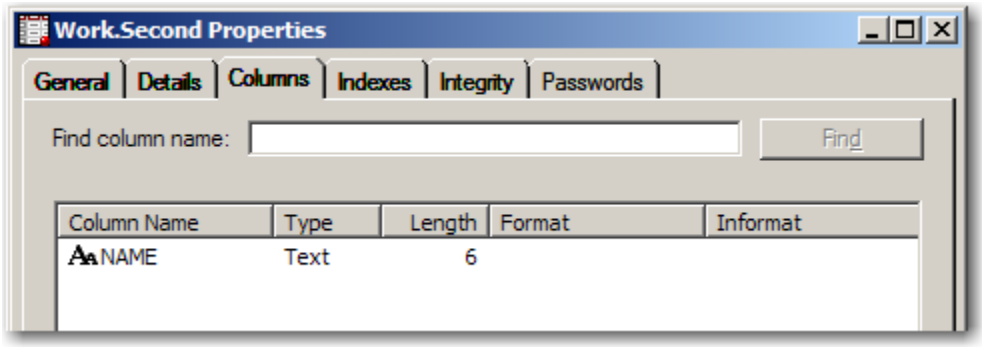
Here is a partial display of the data set properties:



The name in the SASEMF device has a length of 6.

```
proc qdevice out=second;
   device sasemf;
   var name;
run;
```

Here is a partial display of the data set properties:



In order to merge or concatenate these reports, you must use a LENGTH statement that is set to the maximum length of the two NAME variables:

```
data third;
   length name $6.;
   set first second;
run;
```

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. This example assumes that you are running in an interactive mode on Windows.

For the WIN device, the number of colors is controlled by your Windows display settings. The size is controlled by your monitor and resolution settings.

Program

```
proc qdevice;
run;
```

SAS Log

If you do not specify the OUT= option, the QDEVICE procedure sends its output to the SAS log. The output for the Windows operating environment is shown below as it appears in the SAS log.

```
                Name: WIN
         Description: Microsoft Windows Display
                Type: System Display
      Device Catalog: ('your-device-catalog')
    Default Typeface: Sasfont
          Font Style: Roman
         Font Weight: Normal
         Font Height: 8 points
       Maximum Colors: 65535
        Visual Color: True Color
       Color Support: RGB
            I/O Type: GTERM
         Data Format: Host Display
              Height: 8.4 inches
               Width: 14.25 inches
             Ypixels: 806
             Xpixels: 1368
          Rows(vpos): 62
       Columns(hpos): 171
         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 Method: None
       Font Embedding: Never
```

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.

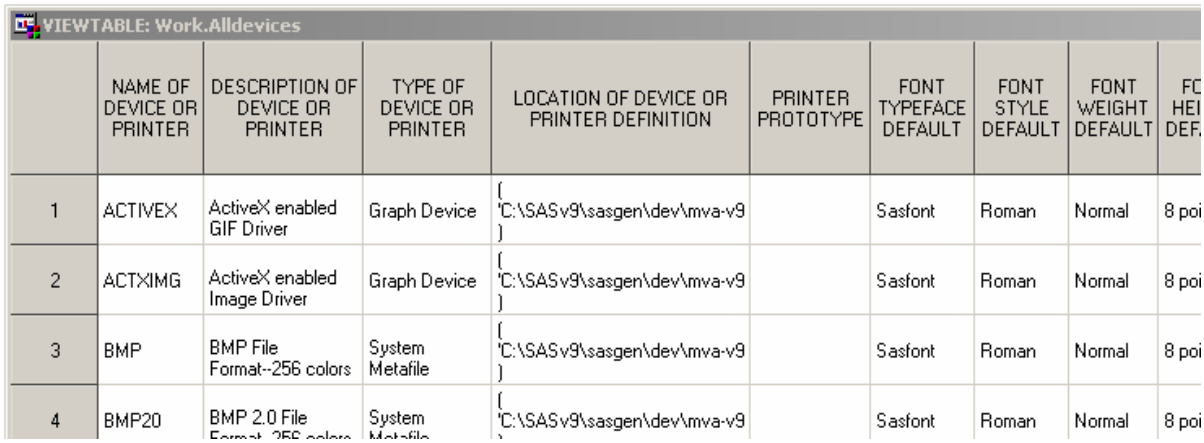
You can use the _ALL_ keyword to generate a report for all devices.

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.



VIEWTABLE: Work.Alldevices

	NAME OF DEVICE OR PRINTER	DESCRIPTION OF DEVICE OR PRINTER	TYPE OF DEVICE OR PRINTER	LOCATION OF DEVICE OR PRINTER DEFINITION	PRINTER PROTOTYPE	FONT TYPEFACE DEFAULT	FONT STYLE DEFAULT	FONT WEIGHT DEFAULT	FC HEI DEF
1	ACTIVEX	ActiveX enabled GIF Driver	Graph Device	('C:\SASv9\sasgen\dev\mva-v9)		Sasfont	Roman	Normal	8 poi
2	ACTXIMG	ActiveX enabled Image Driver	Graph Device	('C:\SASv9\sasgen\dev\mva-v9)		Sasfont	Roman	Normal	8 poi
3	BMP	BMP File Format--256 colors	System Metafile	('C:\SASv9\sasgen\dev\mva-v9)		Sasfont	Roman	Normal	8 poi
4	BMP20	BMP 2.0 File Format--256 colors	System Metafile	('C:\SASv9\sasgen\dev\mva-v9		Sasfont	Roman	Normal	8 poi

Example 3: Generate a Report for SAS/GRAPH Device Drivers and Universal Printers

Features: PROC QDEVICE statement option: OUT=
DEVICE statement
PRINTER statement

Details

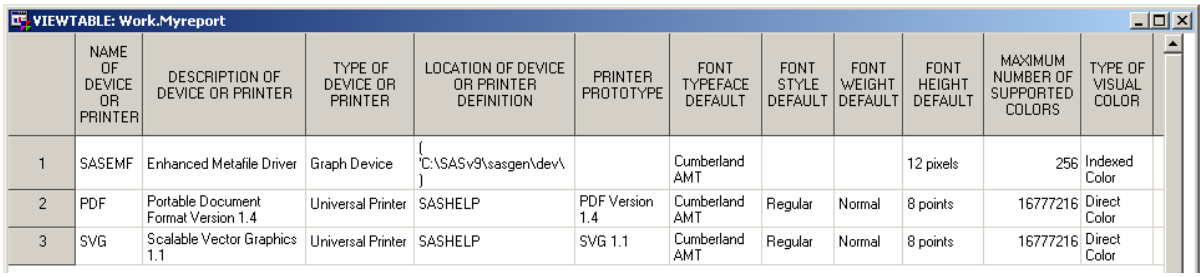
The following example creates a General report for the SASEMF device and the PDF and SVG universal printers. The results are written to the WORK.MYREPORT data set. If you do not specify the REPORT= option, the QDEVICE procedure generates a General report.

Program

```
proc qdevice out=myreport;
  device sasemf;
  printer pdf svg;
run;
```

Output

The following image shows a portion of the report as it appears in the Viewtable window.



VIEWTABLE: Work.Myreport

	NAME OF DEVICE OR PRINTER	DESCRIPTION OF DEVICE OR PRINTER	TYPE OF DEVICE OR PRINTER	LOCATION OF DEVICE OR PRINTER DEFINITION	PRINTER PROTOTYPE	FONT TYPEFACE DEFAULT	FONT STYLE DEFAULT	FONT WEIGHT DEFAULT	FONT HEIGHT DEFAULT	MAXIMUM NUMBER OF SUPPORTED COLORS	TYPE OF VISUAL COLOR
1	SASEMF	Enhanced Metafile Driver	Graph Device	('C:\SASv9\sasgen\dev\ )		Cumberland AMT			12 pixels	256	Indexed Color
2	PDF	Portable Document Format Version 1.4	Universal Printer	SASHELP	PDF Version 1.4	Cumberland AMT	Regular	Normal	8 points	16777216	Direct Color
3	SVG	Scalable Vector Graphics 1.1	Universal Printer	SASHELP	SVG 1.1	Cumberland AMT	Regular	Normal	8 points	16777216	Direct Color

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 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 these topics:

- “Printing” in Chapter 6 of *SAS Companion for Windows*
- “UNIVERSALPRINT System Option” in *SAS Companion for Windows*
- “Universal Printing” in Chapter 15 of *SAS Language Reference: Concepts*

Program: Windows

```
proc qdevice;
  printer _PRINTER_;
run;
```

SAS Log: Default Windows Printer Report

```
4    proc qdevice;
5    printer _PRINTER_;
6    run;

NOTE: The "chpljj24" printer will be used by default with the ODS PRINTER
destination.

                Name: WINPRTC
         Description: Microsoft Windows Hardcopy (color)
                Type: System Printer
      Device Catalog: ( 'path-to-Sashelp' )
    Default Typeface: SAS Monospace
          Font Style: Roman
         Font Weight: Normal
         Font Height: 10 points
       Maximum Colors: 2097152
        Visual Color: True Color
       Color Support: RGB
            I/O Type: PRINTER
         Data Format: Host Printer
              Height: 10.67 inches
               Width: 8.15 inches
             Ypixels: 6392
             Xpixels: 4892
          Rows(vpos): 55
       Columns(hpos): 97
         Left Margin: 0.17 inches
  Minimum Left Margin: 0.17 inches
        Right Margin: 0.18 inches
 Minimum Right Margin: 0.18 inches
       Bottom Margin: 0.17 inches
Minimum Bottom Margin: 0.17 inches
          Top Margin: 0.18 inches
   Minimum Top Margin: 0.18 inches
       XxY Resolution: 600x600 pixels per inch
   Compression Method: None
       Font Embedding: Never
```

Program: UNIX

```
proc qdevice;
   printer _PRINTER_;
run;
```

SAS Log: Default Universal Printer Report under UNIX

```
4    proc qdevice;
5    printer _PRINTER_;
6    run;

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
                Type: Universal Printer
            Registry: SASHELP
           Prototype: PostScript Level 1 (Color)
     Default Typeface: Cumberland AMT
          Font Style: Regular
         Font Weight: Normal
         Font Height: 8 points
       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): 78
       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 Method: None
       Font Embedding: Option
```

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.

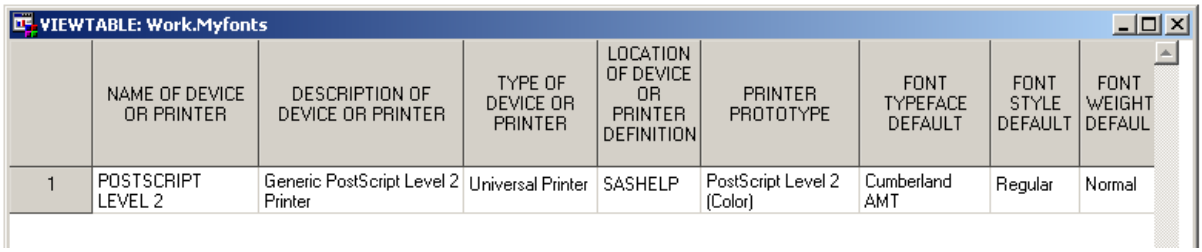
The second example is a SAS program that uses a macro, the DATA step, and the PRINT procedure to create a list of fonts for devices.

Program

```
proc qdevice report=font out=myfonts;
  printer 'postscript level 2';
run;
```

Output

The following output shows the report as it appears in the Viewtable window.



VIEWTABLE: Work.Myfonts

	NAME OF DEVICE OR PRINTER	DESCRIPTION OF DEVICE OR PRINTER	TYPE OF DEVICE OR PRINTER	LOCATION OF DEVICE OR PRINTER DEFINITION	PRINTER PROTOTYPE	FONT TYPEFACE DEFAULT	FONT STYLE DEFAULT	FONT WEIGHT DEFAUL
1	POSTSCRIPT LEVEL 2	Generic PostScript Level 2 Printer	Universal Printer	SASHELP	PostScript Level 2 (Color)	Cumberland AMT	Regular	Normal

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, pcl5c)
```

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.

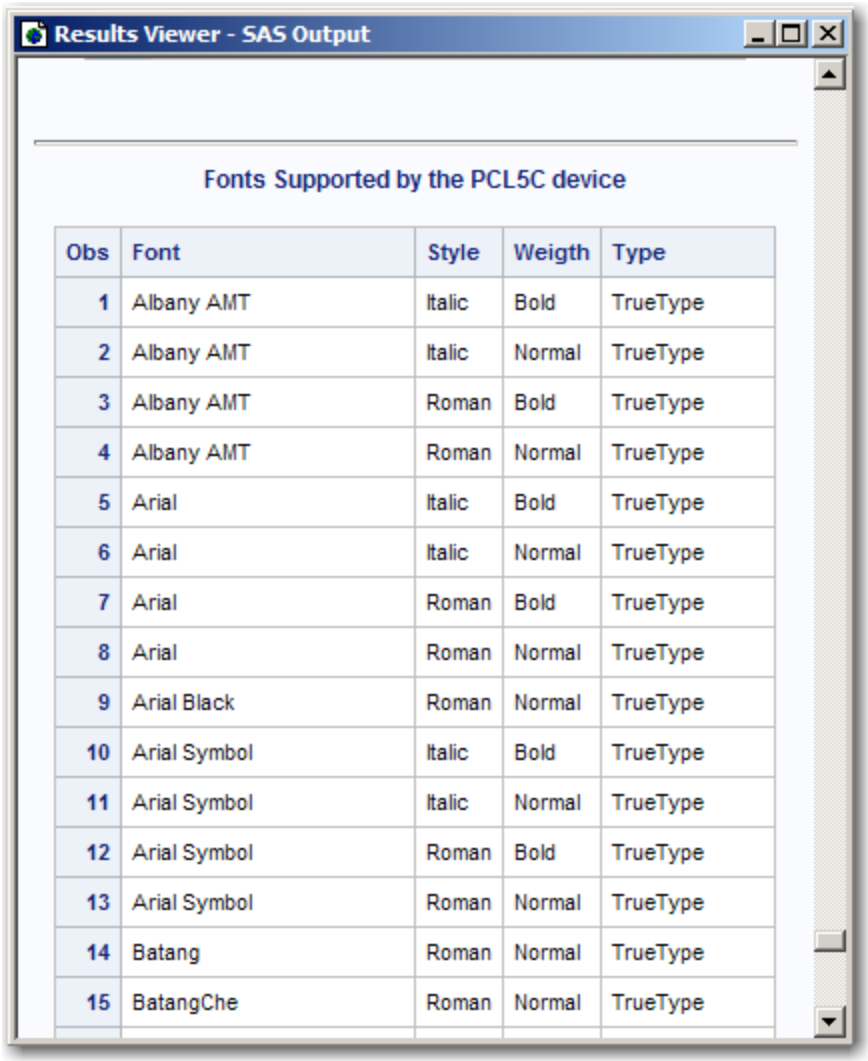
```
%mend fontlist;
```

Use the macro &fontlist to create and output data set for the PCL5c device.

```
%fontlist(device, pcl5c)
```

Output

Output 43.1 *A Partial View of the Fonts Supported by the PCL5c Device*



Results Viewer - SAS Output

Fonts Supported by the PCL5C device

Obs	Font	Style	Weigth	Type
1	Albany AMT	Italic	Bold	TrueType
2	Albany AMT	Italic	Normal	TrueType
3	Albany AMT	Roman	Bold	TrueType
4	Albany AMT	Roman	Normal	TrueType
5	Arial	Italic	Bold	TrueType
6	Arial	Italic	Normal	TrueType
7	Arial	Roman	Bold	TrueType
8	Arial	Roman	Normal	TrueType
9	Arial Black	Roman	Normal	TrueType
10	Arial Symbol	Italic	Bold	TrueType
11	Arial Symbol	Italic	Normal	TrueType
12	Arial Symbol	Roman	Bold	TrueType
13	Arial Symbol	Roman	Normal	TrueType
14	Batang	Roman	Normal	TrueType
15	BatangChe	Roman	Normal	TrueType

Example 6: Generate A Device Option Report

Features: PROC QDEVICE statement options: REPORT=, OUT=, SUPPORT=
DEVICE statement

Details

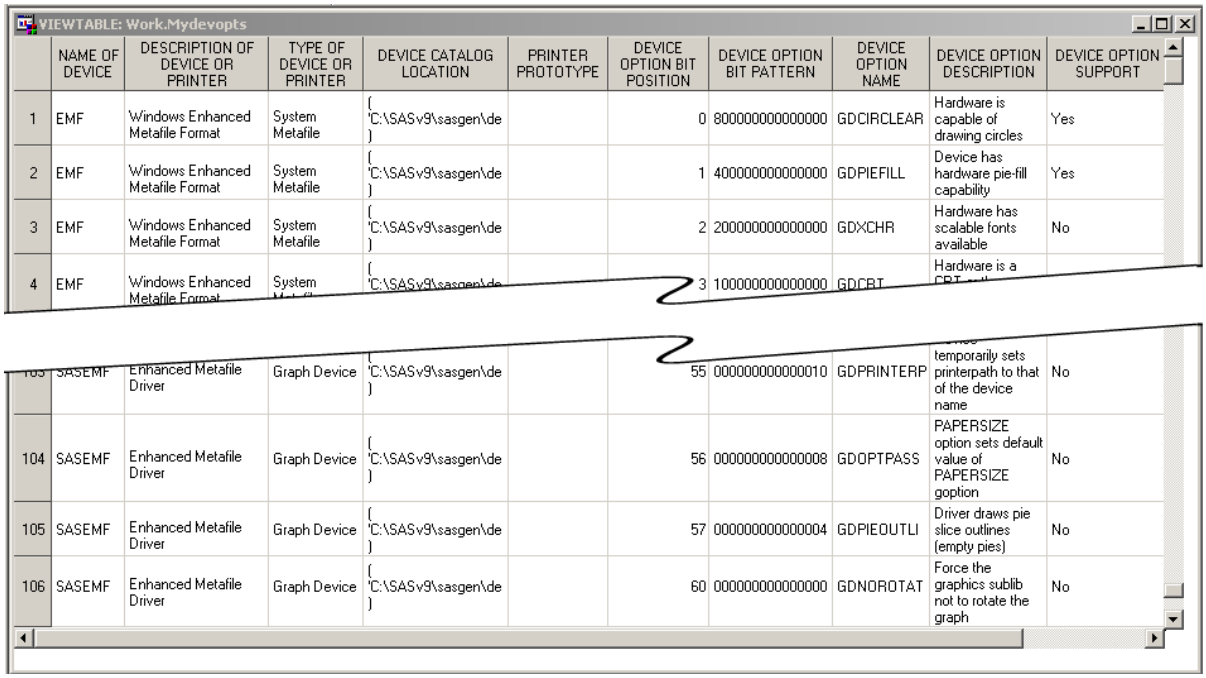
The following example creates a Device Options (DEVOPTIONS) report for the EMF and SASEMF devices. The report is written to the WORK.MYDEVOPTS data set.

Program

```
proc qdevice report=devoption out=mydevopts support=all ;
  device emf sasemf;
run;
```

Output

The following output shows the report as it appears in the Viewtable window.



VIEWTABLE: Work.Mydevopts

	NAME OF DEVICE	DESCRIPTION OF DEVICE OR PRINTER	TYPE OF DEVICE OR PRINTER	DEVICE CATALOG LOCATION	PRINTER PROTOTYPE	DEVICE OPTION BIT POSITION	DEVICE OPTION BIT PATTERN	DEVICE OPTION NAME	DEVICE OPTION DESCRIPTION	DEVICE OPTION SUPPORT
1	EMF	Windows Enhanced Metafile Format	System Metafile	('C:\SASv9\sasgen\de)		0	800000000000000	GDCIRCLEAR	Hardware is capable of drawing circles	Yes
2	EMF	Windows Enhanced Metafile Format	System Metafile	('C:\SASv9\sasgen\de)		1	400000000000000	GDPIEFILL	Device has hardware pie-fill capability	Yes
3	EMF	Windows Enhanced Metafile Format	System Metafile	('C:\SASv9\sasgen\de)		2	200000000000000	GDXCHR	Hardware has scalable fonts available	No
4	EMF	Windows Enhanced Metafile Format	System	('C:\SASv9\sasgen\de		3	100000000000000	GDCRT	Hardware is a	
	SASEMF	Enhanced Metafile Driver	Graph Device	('C:\SASv9\sasgen\de)		55	000000000000010	GDPRINTERP	temporarily sets printerpath to that of the device name	No
104	SASEMF	Enhanced Metafile Driver	Graph Device	('C:\SASv9\sasgen\de)		56	000000000000008	GDOPTPASS	PAPERSIZE option sets default value of PAPERSIZE goption	No
105	SASEMF	Enhanced Metafile Driver	Graph Device	('C:\SASv9\sasgen\de)		57	000000000000004	GDPIEOUTLI	Driver draws pie slice outlines (empty pies)	No
106	SASEMF	Enhanced Metafile Driver	Graph Device	('C:\SASv9\sasgen\de)		60	000000000000000	GDNOROTAT	Force the graphics sublib not to rotate the graph	No

Chapter 44

RANK Procedure

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 outputs 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 44.1 *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 and 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 1195.

Output 44.2 *Assignment of the Lowest Rank Value to the Highest Variable Value within Each BY Group*

```
                      Results of City Council Election                      1

---------------------------------- District=1 ----------------------------------

                                                 Vote    Years
              Obs    Candidate    Vote    Years  Rank     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 ----------------------------------

                                                 Vote    Years
             Obs    Candidate      Vote    Years  Rank     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:

- DB2
- Netezza
- Oracle
- Teradata

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 on 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 on 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. Beginning with SAS 9.3, 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 35. You can also use any global statement. For a list, see “Global Statements” in Chapter 1 of *SAS 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 1185.

PROC RANK <*option(s)*>;
BY <DESCENDING> *variable-1*
<…<DESCENDING>*variable-n*>
<NOTSORTED>;
VAR *data-set-variables(s)*;
RANKS *new-variables(s)*;

Statement	Task	Example
"PROC RANK Statement"	Compute the ranks for one or more numeric variables in a SAS data set and outputs the ranks to a new SAS data set	Ex. 1, Ex. 2, Ex. 3
"BY Statement"	Calculate a separate set of ranks for each BY group	Ex. 2, Ex. 3
"RANKS Statement"	Identify a variable to which the ranks are assigned	Ex. 1, Ex. 2
"VAR Statement"	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 on ranking method can be specified in a single PROC RANK step.

Examples: "Example 1: Ranking Values of Multiple Variables" on page 1193
"Example 2: Ranking Values within BY Groups" on page 1195
"Example 3: Partitioning Observations into Groups Based on Ranks" on page 1198

Syntax

PROC RANK <*option(s)*>;

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, it is necessary that the data set specification refer to a table residing on a supported DBMS.

See: "Input Data Sets" on page 20

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 1193
"Example 2: Ranking Values within BY Groups" on page 1195

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, with the smallest values receiving, 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:

$$FLOOR(rank * k / (n + 1))$$

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 1198

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.

Alias: 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):

$$y_i = \left[\sum_{j=n-r_i+1} \left(\frac{1}{j} \right) \right] - 1$$

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 1184

"Example 1: Ranking Values of Multiple Variables" on page 1193

"Example 2: Ranking Values within BY Groups" on page 1195

BY Statement

Produces a separate set of ranks for each BY group.

Interactions: If the NOTSORTED option is specified on 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: "BY" on page 36

"Example 3: Partitioning Observations into Groups Based on Ranks" on page 1198

Examples: "Example 2: Ranking Values within BY Groups" on page 1195

"Example 3: Partitioning Observations into Groups Based on Ranks" on page 1198

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, 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.

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 1193
"Example 2: Ranking Values within BY Groups" on page 1195

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.

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 1193
"Example 2: Ranking Values within BY Groups" on page 1195
"Example 3: Partitioning Observations into Groups Based on Ranks" on page 1198

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

Other features: 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 when 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 output data set ORDER. 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 output data set ORDER.

```
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

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

Other features: 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 when 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 output data set RESULTS. 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 output data set RESULTS.

```
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.

```
                    Results of City Council Election                    1

---------------------------------- District=1 ----------------------------------

                                                    Vote    Years
              Obs    Candidate    Vote    Years    Rank     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 ----------------------------------

                                                    Vote    Years
             Obs    Candidate      Vote    Years    Rank     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

Other features: 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 when 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 output data set PAIRS. 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 output data set PAIRS.

```
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 output data set RANKPAIR.

```
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.

```
              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 Iman, R.L. "On Some Alternative Procedures Using Ranks for the Analysis of Experimental Designs." 1976. *Communications in Statistics* A5 (14): 1348–1368.

Conover, W.J. and Iman, R.L. "Rank Transformations as a Bridge between Parametric and Nonparametric Statistics." 1981. *The American Statistician* 35: 124–129.

Iman, R.L. and Conover, W.J. "The Use of the Rank Transform in Regression." 1979. *Technometrics* 21: 499–509.

Lehman, E.L. 1998. *Nonparametrics: Statistical Methods Based on Ranks*. Upper Saddle River, New Jersey: Prentice Hall.

Quade, D. "On Analysis of Variance for the k-Sample Problem." 1966. *Annals of Mathematical Statistics* 37: 1747–1758.

Tukey, John W. "The Future of Data Analysis." 1962. *Annals of Mathematical Statistics* 33: 22.

Chapter 45

REGISTRY Procedure

Overview: REGISTRY Procedure

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 section in *SAS Language Reference: Concepts*.

Syntax: REGISTRY Procedure

PROC REGISTRY <*option(s)*>;

Statement	Task	Example
"PROC REGISTRY Statement"	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 Registry" on page 1212
"Example 3: Comparing the Registry to an External File" on page 1214
"Example 2: Listing and Exporting the Registry" on page 1213
"Example 4: Comparing Registry Files" on page 1215
"Example 5: Specifying an Entire Key Sequence with the STARTAT= Option" on page 1217
"Example 6: Displaying a List of Fonts" on page 1217

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 the UNINSTALL= options.

IMPORT=*file-specification*
imports the specified file to a registry.

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.

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

CLEARSASUSER
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: 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 1215

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: Must be used with COMPAREREG1.

Example: "Example 4: Comparing Registry Files" on page 1215

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:

'*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: For information about how to structure a file that contains registry information, see "Creating Registry Files with the REGISTRY Procedure " on page 1209.

Example: "Example 3: Comparing the Registry to an External File" on page 1214

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:

'*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" on page 1213

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 the 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: .sasxreg file extension is not required.

file-specification is one of the following:

'*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: For information about how to structure a file that contains registry information, see "Creating Registry Files with the REGISTRY Procedure " on page 1209.

Example: "Example 1: Importing a File to the Registry" on page 1212

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.

Example: `proc registry listreg='sashelp.regstry'; 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" on page 1213

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, COMPAREREG1=, COMPAREREG2= and the LISTREG options.

Example: "Example 4: Comparing Registry Files" on page 1215

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:

'*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 (For information about the FILENAME statement, see the section on statements in *SAS 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: For information about how to structure a file that contains registry information, see "Creating Registry Files with the REGISTRY Procedure " on page 1209.

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.

Creating Registry Files with the 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\ICONS\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 1211.

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 anything inside the quotes, including nothing ("").

  Note: To include a backslash in the quoted string, 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 display shows how the different types of values that are described above appear in the Registry Editor:

Display 45.1 *Types of Registry Values, Displayed in the Registry Editor*

Name	Data
A double value	2.4E-44
A string	"my data"
Binary data	01,00,76,63,62,6B
Dword	66051
Signed integer value	-123
Unsigned integer value (decimal)	123456

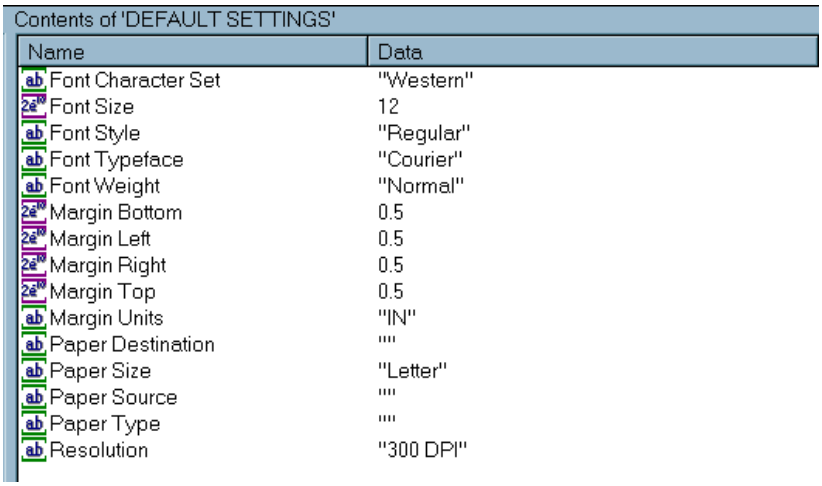
The following list contains a sample of valid registry values:

- a double value=double:2.4E-44
- a string="my data"
- binary data=hex: 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. The following display shows a registry entry that contains default PostScript printer settings.

Display 45.2 *Portion of a Registry Editor Showing Settings for a PostScript Printer*



Contents of 'DEFAULT SETTINGS'

Name	Data
Font Character Set	"Western"
Font Size	12
Font Style	"Regular"
Font Typeface	"Courier"
Font Weight	"Normal"
Margin Bottom	0.5
Margin Left	0.5
Margin Right	0.5
Margin Top	0.5
Margin Units	"IN"
Paper Destination	""
Paper Size	"Letter"
Paper Source	""
Paper Type	""
Resolution	"300 DPI"

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.

Output 45.1 *Log Output of 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 Registry

Features: IMPORT=

Other features: 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;
```

Output: Log

Log 45.1 *Log Output from Importing a File to the Registry*

```
Parsing REG file and loading the registry please wait....
Registry IMPORT is now complete.
```

Example 2: Listing and Exporting the Registry

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;
```

Output: Log

Log 45.2 *Log Output from Exporting a File from the SAS Registry*

```
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

Other features: 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;
```

Output: Log

Log 45.3 *Log Output 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.regstry'

   startat='CORE\EXPLORER'
```

```
   comparereg2='proclib.regstry';
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;
```

Output: Log

Log 45.4 *Log Output from Comparing 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;
```

Output: Log

Output 45.2 *Partial Log Output from Displaying a List of Fonts*

```
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"
```

Chapter 46

REPORT Procedure

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 three ways:

- in a nonwindowing environment. In this case, you submit a series of statements with the PROC REPORT statement, just as you do in other SAS procedures. You can submit these statements from the Program Editor with the NOWINDOWS option in the PROC REPORT statement, or you can run SAS in batch, noninteractive, or interactive line mode. (See the information about "Ways to Run Your SAS Session " in Chapter 1 of *SAS Language Reference: Concepts*.)
- in an interactive report window environment with a prompting facility that guides you as you build a report.
- in an interactive report window environment without the prompting facility.

This documentation provides reference information about using PROC REPORT in a windowing or nonwindowing environment. For task-oriented documentation for the nonwindowing environment, see SAS Technical Report P-258, **Using the REPORT Procedure in a Nonwindowing Environment, Release 6.07**.

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, 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 1234.)

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 for a Report" on page 1307. The formats are stored in a permanent SAS library. See the REPORT procedure examples for more reports and for the statements that create them.

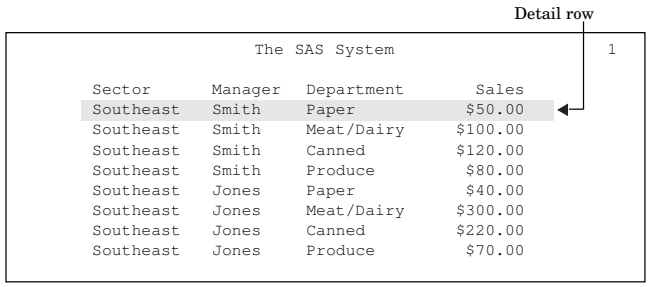
What Do the Various Types of Reports Look Like?

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 46.1 on page 1221 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 for a Report" on page 1307. 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 nowd;
   where sector='se';
   format sector $sctrfmt. manager $mgrfmt.
          dept $deptfmt. sales dollar10.2;
run;
```

Figure 46.1 *Simple Detail Report with a Detail Row for Each Observation*

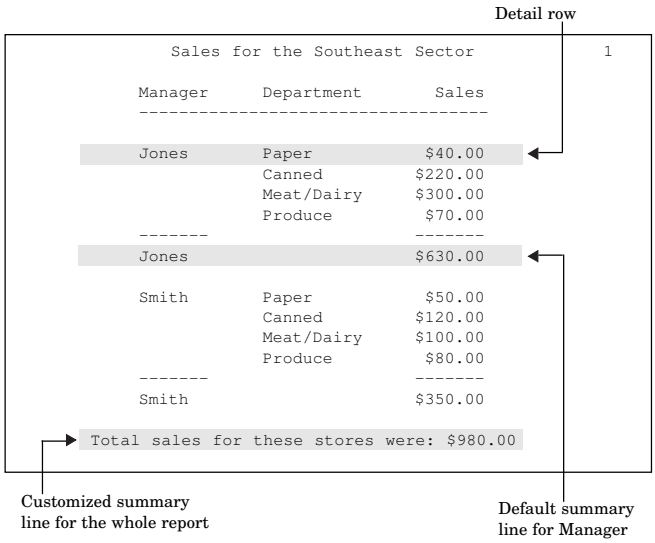


The report in the following figure uses the same observations as the above figure. However, the statements that produce this report

- 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 1311.

Figure 46.2 *Ordered Detail Report with Default and Customized Summaries*

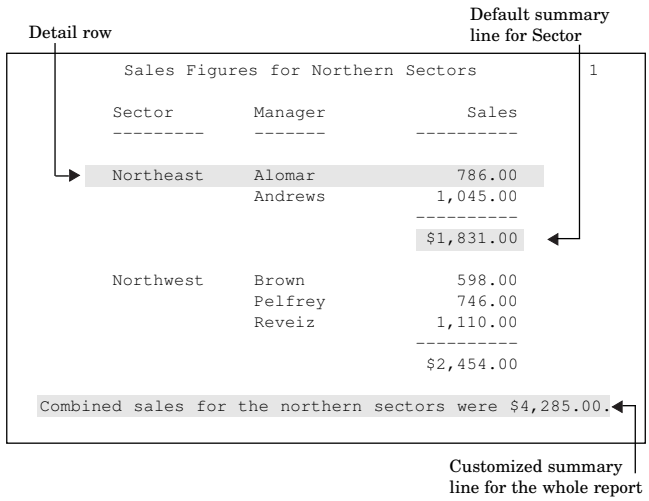


The summary report in the following figure contains one row for each store in the northern sector. Each detail row represents four observations in the input data set, one observation for each department. Information about individual departments does not appear in this report. Instead, the value of Sales 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 4: Consolidating Multiple Observations into One Row of a Report" on page 1318.

Figure 46.3 *Summary Report with Default and Customized Summaries*

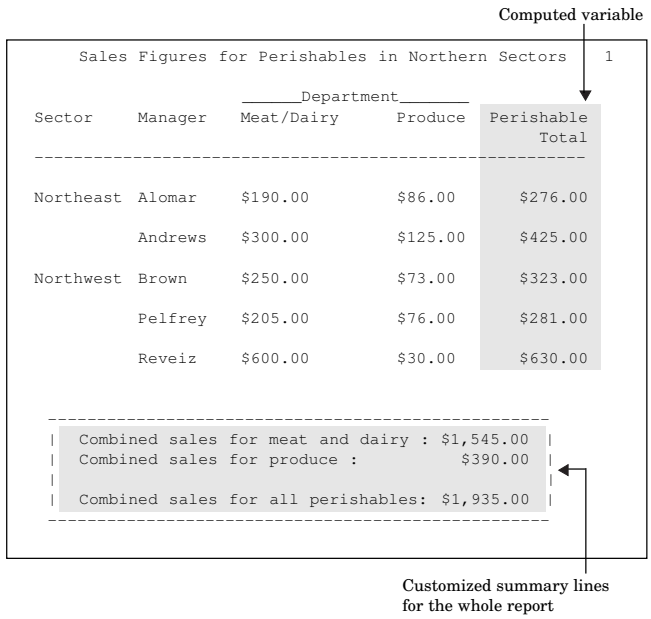


The summary report in the following figure is similar to the above figure. The major difference is that it also includes information for individual departments. Each selected value of Department forms a column in the report. In addition, the statements that create this report

- compute and display a variable that is not in the input data set
- double-space the report
- put blank lines in some of the column headings

For an explanation of the program that produces this report, see "Example 5: Creating a Column for Each Value of a Variable" on page 1321.

Figure 46.4 *Summary Report with a Column for Each Value of a Variable*

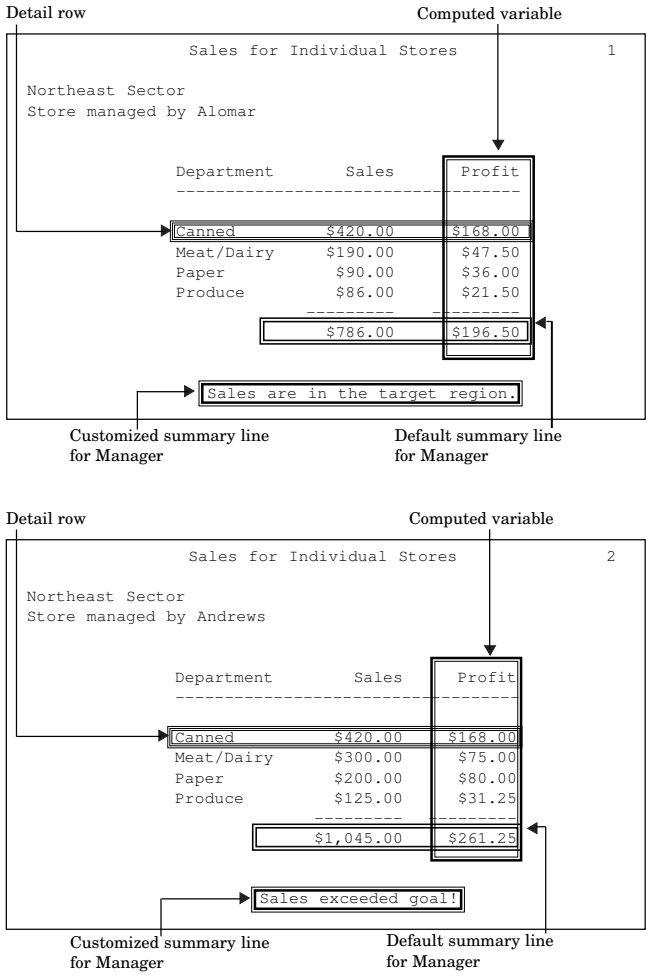


The customized report in the following figure shows each manager's store on a separate page. Only the first two pages appear here. The statements that create this report create

- 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 9: Writing a Customized Summary on Each Page" on page 1333.

Figure 46.5 *Customized Summary Report*



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 16: Specifying Style Elements for ODS Output in Multiple Statements” on page 1362. For information about ODS, see “Output Delivery System” on page 34.

Figure 46.6 *HTML Output*

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.

Note: If you start PROC REPORT in the interactive report window environment without the COLUMN statement, then the initial report includes only as many variables as will fit on one page.

The DEFINE statement (or, in the interactive report window environment, the DEFINITION window) 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 or DEFINITION windows. 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 for a Report" on page 1307.

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 or with the DEFINITION window.

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 1311.

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 or with the DEFINITION window.

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 will 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 1271.) In the interactive report window environment, if PROC REPORT cannot immediately create groups, then the procedure changes all display and order variables to group variables so that it can create the group variable that you requested. In the nonwindowing environment, it returns to the SAS log a message that explains why it could not create groups. Instead, it 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 4: Consolidating Multiple Observations into One Row of a Report" on page 1318.

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 1262 and "RBREAK Statement" on page 1290 statements.

For examples, refer to "Example 2: Ordering the Rows in a Report" on page 1311, "Example 3: Using Aliases to Obtain Multiple Statistics for the Same Variable" on page 1314, "Example 4: Consolidating Multiple Observations into One Row of a Report" on page 1318, and "Example 5: Creating a Column for Each Value of a Variable" on page 1321.

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, 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 or with the DEFINITION window. 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 1271.

If you are familiar with procedures that use class variables, then you will 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 5: Creating a Column for Each Value of a Variable" on page 1321.

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.

In the interactive report window environment, you add a computed variable to a report from the COMPUTED VAR window.

In the nonwindowing environment, 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 5: Creating a Column for Each Value of a Variable" on page 1321, "Example 10: Calculating Percentages" on page 1338, and "Example 13: Storing Computed Variables as Part of a Data Set" on page 1349.

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 values of order and group variables, considered from left to right as specified in the report window or 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. You create this type of report with the COLUMN statement or, in the interactive report window environment, by placing report items above or below each other. This arrangement is called stacking items in the report because each item generates a heading, and the headings are stacked one above the other.

```
title 'The SAS System'       ;
options nodate pageno=1 linesize=64 pagesize=60 fmtsearch=(proclib);
proc report data=grocery nowd headskip headline 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 46.7 *Stacking Department and Sales*

```
      Sales Figures for Perishables in Northern Sectors

                      _______Department________
 Sector     Manager    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
```

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 46.1 *Report Items That Can Share Columns*

	Display	Analysis	Order	Group	Computed	Across	Statistic
Display						X*	X
Analysis						X	X
Order							
Group						X	

	Display	Analysis	Order	Group	Computed	Across	Statistic
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.

Statistics That Are Available in PROC REPORT

Table 46.2 *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	P90
P5	P95
P10	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 1666.

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 or in the DEFINITION window for an analysis variable.

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 Compute Blocks

What Is a Compute Block?

A **compute block** is one or more programming statements that appear either between a COMPUTE and an ENDCOMP statement or in a COMPUTE window. PROC REPORT executes these statements as it builds the report. A compute block can be associated with a report item (a data set variable, a statistic, or a computed variable) or with a location (at the top or bottom of the report; before or after a set of observations). You create a compute block with the COMPUTE window or with the COMPUTE statement. One form of the COMPUTE statement associates the compute block with a report item. Another form associates the compute block with a location in the report. (See "Using Break Lines" on page 1234.)

Note: When you use the COMPUTE statement, you do not have to use a corresponding BREAK or RBREAK statement. (See "Example 2: Ordering the Rows in a Report" on page 1311, 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 9: Writing a

Customized Summary on Each Page" on page 1333, which uses both COMPUTE AFTER MANAGER and BREAK AFTER MANAGER.)

The Purpose of Compute Blocks

A compute block that is associated with a report item can

- 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 1268.)
- 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 1232.) A PROC REPORT step can contain multiple compute blocks, but they cannot be nested.

The Contents of Compute Blocks

In the interactive report window environment, a compute block is in a COMPUTE window. In the nonwindowing environment, 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 the "LINE Statement" on page 1289.)
- 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 1268.)
- Any compute block can reference the automatic variable _BREAK_. (See "The Automatic Variable _BREAK_" on page 1235.)

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

  '_C*n*_'
  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 1284 and "NOZERO" on page 1284.

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 1314, which references analysis variables by their aliases; "Example 5:

Creating a Column for Each Value of a Variable" on page 1321, which references variables by column number; and "Example 10: Calculating Percentages" on page 1338, which references group variables and computed variables by name.

Compute Block Processing

PROC REPORT processes compute blocks in two different ways.

- If a compute block is associated with a location, then PROC REPORT executes the compute block only at that location. Because PROC REPORT calculates statistics for groups before it actually constructs the rows of the report, statistics for sets of report rows are available before or after the rows are displayed, as are values for any variables based on these statistics.
- If a compute block is associated with a report item, then PROC REPORT executes the compute block on every row of the report when it comes to the column for that item. The value of a computed variable in any row of a report is the last value assigned to that variable during that execution of the DATA step statements in the compute block. 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.

Note: PROC REPORT recalculates computed variables at breaks. For details about compute block processing see "How PROC REPORT Builds a Report " on page 1297.

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

- 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

- 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, or the BREAK window. 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

1. overlining or double overlining (in traditional SAS monospace output only)
2. summary line
3. underlining or double underlining
4. blank line (in traditional SAS monospace output only)
5. page break

In traditional SAS monospace output only, if you define a customized summary for the same location, then customized break lines appear after underlining or double underlining.

The Automatic Variable _BREAK_

PROC REPORT automatically creates a variable called _BREAK_. This variable contains

- 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 1309.

```
libname proclib
'SAS-library';

options nodate pageno=1 linesize=64
        pagesize=60 fmtsearch=(proclib);
proc report data=grocery nowindows;
   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 skip ol;
   rbreak after / summarize dol dul;
```

```
   compute after;
      sector='Total:';
   endcomp;
run;
```

Log 46.1 *Three Different Meanings of Sales.sum*

```
                          The SAS System
1

                  Sector     Manager      Sales
                  Northeast  Alomar     $786.00     1
                             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 1233.)

Using Style Elements in PROC REPORT

Using the STYLE= Option

If you use the Output Delivery System to create HTML, RTF, or Printer output from PROC REPORT, then you can use the STYLE= option to specify style elements for the procedure to use in various parts of the report. Style elements determine presentation

attributes like font type, font weight, color, and so on. For information about style attributes and their values, see "Style Attributes Tables " in Chapter 13 of *SAS Output Delivery System: User's Guide*.

The general form of the STYLE= option is

STYLE<(*location(s)*)>=<*style-element-name*><[*style-attribute-specification(s)*]>

Note: You can use braces ({ and }) instead of square brackets ([and]).

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*.

Table 46.3 *Location Values*

Location Value	Part of Report Affected
CALLDEF	Cells identified by a CALL DEFINE statement
COLUMN	Column cells
HEADER|HDR	Column headings
LINES	Lines generated by LINE statements
REPORT	Report as a whole
SUMMARY	Summary lines

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.

style-element-name
is the name of a style element that is part of a style definition that is registered with the Output Delivery System. SAS provides some style definitions. Refer to "ODS Style Elements" in Chapter 5 of *SAS Output Delivery System: User's Guide* for a list of SAS provided style elements. Users can create their own style definitions with the TEMPLATE procedure. See the "TEMPLATE Procedure: Overview" in Chapter 9 of *SAS Output Delivery System: User's Guide*. The following table shows the default style elements for each statement.

Table 46.4 *Locations and Default Style Elements for Each Statement in PROC REPORT*

Statement	Valid Location Values	Default Location Value	Default Style Element
PROC REPORT	REPORT, COLUMN, HEADER|HDR, SUMMARY, LINES, CALLDEF	REPORT	Table
BREAK	SUMMARY, LINES	SUMMARY	DataEmphasis
CALL DEFINE	CALLDEF	CALLDEF	Data

Statement	Valid Location Values	Default Location Value	Default Style Element
COMPUTE	LINES	LINES	NoteContent
DEFINE	COLUMN, HEADER|HDR	COLUMN and HEADER	COLUMN: Data HEADER: Header
RBREAK	SUMMARY, LINES	SUMMARY	DataEmphasis

style-attribute-specification(s)
describes the style attribute to change. Each *style-attribute-specification* has this general form:

style-attribute-name=style-attribute-value
To specify more than one *style-attribute-specification*, separate each one with a space.

The following table shows valid values of *style-attribute-name* for PROC REPORT. Note that not all style attributes are valid in all destinations. See “Style Attributes Tables ” in Chapter 13 of *SAS Output Delivery System: User's Guide* for more information about style attributes, their valid values, and their applicable destinations.

Table 46.5 *Style Attributes for PROC REPORT and PROC TABULATE*

Attribute	PROC REPORT STATEMENT: REPORT Area	PROC REPORT Areas: CALLDEF, COLUMN, HEADER, LINES, SUMMARY	PROC TABULATE STATEMENT: TABLE	PROC TABULATE STATEMENTS: VAR, CLASS, BOXOpt, CLASSLEV, KEYWORD
ASIS=	X	X		X
BACKGOUNDCOLOR=	X	X	X	X
BACKGOUNDIMAGE=	X	X	X	X
BORDERBOTTOMCOLOR=	X	X		X
BORDERBOTTOMSTYLE=	X	X	X	X
BORDERBOTTOMWIDTH=	X	X	X	X
BORDERCOLOR=	X	X		X
BORDERCOLORDARK=	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, BOXOpt, CLASSLEV, KEYWORD
BORDERCOLORLIGHT=	X	X	X	X
BORDERTOPCOLOR=	X	X		X
BORDERTOPSTYLE=	X	X	X	X
BORDERTOPWIDTH=	X	X	X	X
BORDERWIDTH=	X	X	X	X
CELLPADDING=	X		X	
CELLSPACING=	X		X	
CELLWIDTH=	X	X	X	X
CLASS=	X	X	X	X
COLOR=	X	X	X	
FLYOVER=	X	X		X
FONT=	X	X	X	X
FONTFAMILY=	X	X	X	X
FONTSIZE=	X	X	X	X
FONTSTYLE=	X	X	X	X
FONTWEIGHT=	X	X	X	X
FONTWIDTH=	X	X	X	X
FRAME=	X		X	
HEIGHT=	X	X		X
HREFTARGET=		X		X
HTMLSTYLE=	X	X	X	X
NOBREAKSPACE=	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, BOXOpt, CLASSLEV, KEYWORD
OUTPUTWIDTH=	X	X	X	X
POSTHTML=	X	X	X	X
POSTIMAGE=	X	X	X	X
POSTTEXT=	X	X	X	X
PREHTML=	X	X	X	X
PREIMAGE=	X	X	X	X
PRETEXT=	X	X	X	X
PROTECTSPECIALCHARS=		X		X
RULES=	X		X	
TAGATTR=	X	X		X
TEXTALIGN=	X	X	X	X
URL=		X		X
VERTICALALIGN=		X		X
WIDTH=	X	X	X	X

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).

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 Profit column for which the value is negative, and a green background color where the values are positive:

```
proc format;
   value proffmt low-<0='red'
                 0-high='green';
run;
```

```
ods html body='external-HTML-file';
proc report data=profits nowd;
   title 'Profits for Individual Stores';
   column Store Profit;
   define Store / display 'Store';
   define Profit / display 'Profit' style=[backgroundcolor=proffmt.];
run;
ods html close;
```

Controlling the Spacing between Rows

Users frequently 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 user, 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 nowindows
data=libref.data-set-name
            style(report)=[fontsize=8pt]
            style(column)=[font=(Arial, 8pt)];
```

Printing a Report

Printing with ODS

Printing reports with the Output Delivery System is much simpler and provides more attractive output than the older methods of printing that are documented here. For best results, use an output destination in the ODS printer family or RTF. For details about these destinations and on using the ODS statements, see *SAS Output Delivery System: User's Guide*.

Printing from the REPORT Window

By default, if you print from the REPORT window, then the report is routed directly to your printer. If you want, you can specify a form to use for printing. (See "Printing with a Form" on page 1241.) Forms specify things like the type of printer that you are using, text format, and page orientation.

Note: Forms are available only when you run SAS from an interactive report window environment.

Operating Environment Information
Printing is implemented differently in different operating environments. See "Printing with SAS" in Chapter 15 of *SAS Language Reference: Concepts* for information related to printing. Additional information might be available in the SAS documentation for your operating environment.

Printing with a Form

To print with a form from the REPORT window:

1. Specify a form. You can specify a form with the FORMNAME command or, in some cases, through the **File** menu.
2. Specify a print file if you want the output to go to a file instead of directly to the printer. You can specify a print file with the PRTFILE command or, in some cases, through the **File** menu.

3. Issue the PRINT or PRINT PAGE command from the command line or from the **File** menu.
4. If you specified a print file, then do the following:
   a. Free the print file. You can free a file with the FREE command or, in some cases, through **Print utilities** in the **File** menu. You cannot view or print the file until you free it.
   b. Use operating environment commands to send the file to the printer.

Printing from the Output Window

If you are running PROC REPORT with the NOWINDOWS option, then the default destination for the output is the Output window. Use the commands in the **File** menu to print the report.

Printing from Noninteractive or Batch Mode

If you use noninteractive or batch mode, then SAS writes the output either to the display or to external files, depending on the operating environment and on the SAS options that you use. Refer to the SAS documentation for your operating environment for information about how these files are named and where they are stored.

You can print the output file directly or use PROC PRINTTO to redirect the output to another file. In either case, no form is used, but carriage-control characters are written if the destination is a print file.

Use operating environment commands to send the file to the printer.

Printing from Interactive Line Mode

If you use interactive line mode, then by default the output and log are displayed on the screen immediately following the programming statements. Use PROC PRINTTO to redirect the output to an external file. Then use operating environment commands to send the file to the printer.

Using PROC PRINTTO

PROC PRINTTO defines destinations for the SAS output and the SAS log. (See Chapter 38, “PRINTTO Procedure,” on page 1073.)

PROC PRINTTO does not use a form, but it does write carriage-control characters if you are writing to a print file.

Note: You need two PROC PRINTTO steps. The first PROC PRINTTO step precedes the PROC REPORT step. It redirects the output to a file. The second PROC PRINTTO step follows the PROC REPORT step. It reestablishes the default destination and frees the output file. You cannot print the file until PROC PRINTTO frees it.

Storing and Reusing a Report Definition

The OUTREPT= option in the PROC REPORT statement stores a report definition in the specified catalog entry. If you are working in the nonwindowing environment, then the definition is based on the PROC REPORT step that you submit. If you are in the interactive report window environment, then the definition is based on the report that is in the REPORT window when you end the procedure. SAS assigns an entry type of REPT to the entry.

In the interactive report window environment, you can save the definition of the current report by selecting **File** ⇨ **Save Report**. A report definition might differ from the SAS program that creates the report. See the discussion of OUTREPT= on page 1255.

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. In the interactive report window environment, load a report definition from the LOAD REPORT window by selecting **File** ⇨ **Open Report**.

ODS Destinations Supported by PROC REPORT

Prior to SAS 9.2, the ODS DOCUMENT and the ODS OUTPUT destinations were unsupported by PROC REPORT. Now, PROC REPORT supports all ODS destinations. Refer to “Understanding ODS Destinations” in Chapter 3 of *SAS Output Delivery System: User's Guide* for detailed information.

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. When the output is in a DOCUMENT form, it is possible to rearrange, restructure, and reformat without rerunning your analysis. Refer to “The DOCUMENT Procedure” in Chapter 8 of *SAS Output Delivery System: User's Guide* for additional information.

The OUTPUT destination produces 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. Refer to the “ODS OUTPUT Statement ” in *SAS Output Delivery System: User's Guide* for additional information.

Syntax: REPORT Procedure

Tips: Supports the Output Delivery System. For details, see Chapter 3, “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 more information, see “Statements with the Same Function in Multiple Procedures” on page 35. You can also use any global statement. For a list, see “Global Statements” in Chapter 1 of *SAS Statements: Reference*.

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 1296.

```
PROC REPORT <option(s)>;
  BREAK location break-variable</ option(s)>;
  BY <DESCENDING> variable-1
    <…<DESCENDING>variable-n> <NOTSORTED>;
  COLUMN column-specification(s);
    COMPUTE location <target>
      </ STYLE=<style-element-name><[style-attribute-specification(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;
  DEFINE report-item / <usage>
    <attribute(s)>
    <option(s)>
    <justification>
    <COLOR=color>
    <'column-header-1' <…'column-header-n'>>
    <style>;
  FREQ variable;
  RBREAK location </ option(s)>;
  WEIGHT variable;
```

Statement	Task	Example
"PROC REPORT Statement"	Produce a summary or detail report	Ex. 1, Ex. 2, Ex. 5, Ex. 6, Ex. 7, Ex. 8, Ex. 11, Ex. 12, Ex. 13, Ex. 15, Ex. 16
"BREAK Statement"	Produce a default summary at a change in the value of a group or order variable	Ex. 2, Ex. 4, Ex. 5, Ex. 8, Ex. 9
"BY Statement"	Create a separate report for each BY group	
"CALL DEFINE Statement"	Set the value of an attribute for a particular column in the current row	Ex. 4, Ex. 16
"COLUMN Statement"	Describe the arrangement of all columns and of headings that span more than one column	Ex. 1, Ex. 3, Ex. 5, Ex. 6, Ex. 10, Ex. 11
"COMPUTE Statement"	Specify one or more programming statements that PROC REPORT executes as it builds the report	Ex. 2, Ex. 3, Ex. 4, Ex. 5, Ex. 9, Ex. 10, Ex. 13, Ex. 16

Statement	Task	Example
"ENDCOMP Statement"	Specify one or more programming statements that PROC REPORT executes as it builds the report	Ex. 2
"DEFINE Statement"	Describe how to use and display a report item	Ex. 2, Ex. 3, Ex. 4, Ex. 5, Ex. 6, Ex. 9, Ex. 10, Ex. 12, Ex. 13, Ex. 14, Ex. 16
"FREQ Statement"	Treat observations as if they appear multiple times in the input data set	
"LINE Statement"	Provide a subset of features of the PUT statement for writing customized summaries	Ex. 2, Ex. 3, Ex. 4, Ex. 5, Ex. 9
"RBREAK Statement"	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. 10
"WEIGHT Statement"	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.

Examples:

"Example 1: Selecting Variables for a Report" on page 1307
"Example 2: Ordering the Rows in a Report" on page 1311
"Example 5: Creating a Column for Each Value of a Variable" on page 1321
"Example 6: Displaying Multiple Statistics for One Variable" on page 1325
"Example 7: Storing and Reusing a Report Definition" on page 1327
"Example 8: Condensing a Report into Multiple Panels" on page 1330
"Example 11: How PROC REPORT Handles Missing Values" on page 1342
"Example 12: Creating and Processing an Output Data Set" on page 1346
"Example 13: Storing Computed Variables as Part of a Data Set" on page 1349
"Example 15: Specifying Style Elements for ODS Output in the PROC REPORT Statement" on page 1356
"Example 16: Specifying Style Elements for ODS Output in Multiple Statements" on page 1362
"Example 18: Using the WIDTH= and CELLWIDTH= Style Attributes with PROC REPORT" on page 1372

Syntax

PROC REPORT <*option(s)*>;

Summary of Optional Arguments

DATA=*SAS-data-set*
: specifies the input data set.

NOALIAS
: uses a report that was created before compute blocks required aliases (prior to Release 6.11).

NOCENTER
: See CENTER | NOCENTER.

NOCOMPLETECOLS
: See COMPLETECOLS | NOCOMPLETECOLS.

NOCOMPLETEROWS
: See COMPLETEROWS | NOCOMPLETEROWS.

NOTHREADS
: See THREADS | NOTHREADS.

NOWINDOWS
: See WINDOWS | NOWINDOWS.

OUT=*SAS-data-set*
: specifies the output data set.

PCTLDEF=
: See QNTLDEF=.

THREADS | NOTHREADS
: overrides the SAS system option THREADS | NOTHREADS.

WINDOWS | NOWINDOWS
: selects the interactive report window or the nonwindowing environment.

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*'
: specifies text for the HTML or PDF table of contents entry for the output.

SPANROWS
: specifies that a single cell will occupy the column in all the rows for which the value is the same. Only applies to ODS MARKUP, PDF, RTF, and PRINTER destinations.

STYLE<(*location(s)*)>=<*style-element-name*><[*style-attribute-specification(s)*]>
: specifies one or more style elements (for the Output Delivery System) to use for different parts of the report.

Control the interactive report window environment

COMMAND
: displays command lines rather than menu bars in all REPORT windows.

HELP=*libref.catalog*
: identifies the library and catalog containing user-defined help for the report.

PROMPT
: opens the REPORT window and starts the PROMPT facility.

Control the layout of the report

BOX
: uses formatting characters to add line-drawing characters to 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.

COLWIDTH=*column-width*
: specifies the default number of characters for columns containing computed variables or numeric data set variables.

FORMCHAR <(*position(s)*)>='*formatting-character(s)*'
: defines the characters to use as line-drawing characters in the report.

LS=*line-size*
: specifies the length of a line of the report.

MISSING
: considers missing values as valid values for group, order, or across variables.

PANELS=*number-of-panels*
: specifies the number of panels on each page of the report.

PS=*page-size*
: specifies the number of lines in a page of the report. Use for monospace output only.

PSPACE=*space-between-panels*
: specifies the number of blank characters between panels.

SHOWALL
: overrides options in the DEFINE statement that suppress the display of a column.

SPACING=*space-between-columns*
: specifies the number of blank characters between columns.

WRAP
: displays one value from each column of the report, on consecutive lines if necessary, before displaying another value from the first 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

HEADLINE
: underlines all column headings and the spaces between them.

HEADSKIP
: writes a blank line beneath all 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.

PROFILE=*libref.catalog*
identifies the report profile to use.

REPORT=*libref.catalog.entry*
specifies the report definition to use.

Optional Arguments

BOX
uses formatting characters to add line-drawing characters to the report. These characters

- surround each page of the report
- separate column headings from the body of the report
- separate rows and columns from each other
- separate values in a summary line from other values in the same columns
- separate a customized summary from the rest of the report

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Interaction: You cannot use BOX if you use WRAP in the PROC REPORT statement or in the ROPTIONS window or if you use FLOW in any item definition.

See: the discussion of "FORMCHAR <(position(s))>='formatting-character(s)' " on page 1250

Example: "Example 12: Creating and Processing an Output Data Set" on page 1346

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.

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.

COLWIDTH=*column-width*

specifies the default number of characters for columns containing computed variables or numeric data set variables.

When setting the width for a column, PROC REPORT first looks at WIDTH= in the definition for that column. If WIDTH= is not present, then PROC REPORT uses a column width large enough to accommodate the format for the item. If no format is associated with the item, then the column width depends on variable types as shown in the following table.

Table 46.6 *Using References in a Compute Block*

Variable	Resulting Column Width
Character variable in the input data set	Length of the variable
Numeric variable in the input data set	Value of the COLWIDTH= option
Computed variable (numeric or character)	Value of the COLWIDTH= option

For information about formats, see the discussion of "FORMAT=format" on page 1282.

Default: 9

Range: 1 to the line size

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output. For ODS destinations, use the STYLE= option with the WIDTH=, CELLWIDTH=, or OUTPUTWIDTH = style attributes. Refer to "Style Attributes Tables " in Chapter 13 of *SAS Output Delivery System: User's Guide* for details. See how style attributes WIDTH= and CELLWIDTH= can be used with PROC REPORT in "Example 18: Using the WIDTH= and CELLWIDTH= Style Attributes with PROC REPORT" on page 1372.

COMMAND

displays command lines rather than menu bars in all REPORT windows.

After you have started PROC REPORT in the interactive report window environment, you can display the menu bars in the current window by issuing the COMMAND command. You can display the menu bars in all PROC REPORT windows by issuing the PMENU command. The PMENU command affects all the windows in your SAS session. Both of these commands are toggles.

You can store a setting of COMMAND in your report profile. PROC REPORT honors the first of these settings that it finds:

- the COMMAND option in the PROC REPORT statement
- the setting in your report profile

Restriction: This option has no effect in the nonwindowing environment.

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*'
specifies the text for the entries in the HTML contents file or PDF table of contents for the output that is produced by PROC REPORT. For information about HTML and PDF output, see "Output Delivery System" on page 34.

Restriction: For HTML output, the CONTENTS= option has no effect in the HTML body file. It affects only the HTML contents file.

Interaction: 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 will then show up in this separate Table of Contents page.

DATA=*SAS-data-set*
specifies the input data set.

See: "Input Data Sets" on page 20

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 1295

FORMCHAR <(*position(s)*)>='*formatting-character(s)*'
defines the characters to use as line-drawing characters in the report.

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.

Note: PROC REPORT uses 12 of the 20 formatting characters that SAS provides. Table 46.7 on page 1251 shows the formatting characters that PROC REPORT uses. Figure 46.8 on page 1252 illustrates the use of some commonly used formatting character in the output from PROC REPORT.

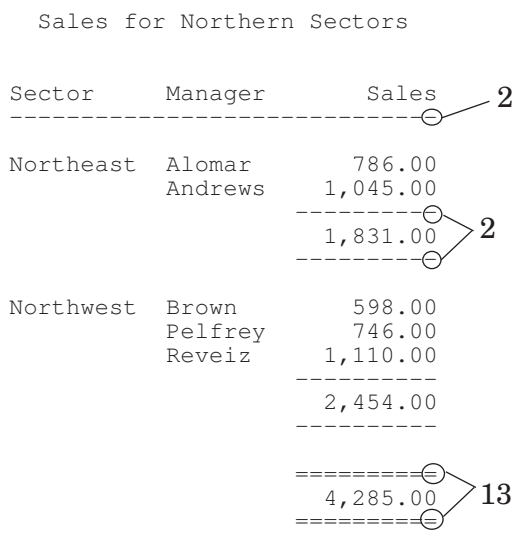
formatting-character(s)
lists the characters to use for the specified positions. PROC REPORT assigns characters in *formatting-character(s)* to *position(s)*, in the order in which they are listed. For example, the following option assigns the asterisk (*) to the third formatting character, the pound sign (#) to the seventh character, and does not alter the remaining characters: `formchar(3,7)='*#'`

Table 46.7 *Formatting Characters Used by PROC REPORT*

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; also underlining and overlining in break lines as well as the underlining that the HEADLINE option draws
3	-	Top character in the left border
4	-	Top character in a line of characters that separates 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

11	-	Bottom character in the right border
13	=	Double overlining and double underlining in break lines

Figure 46.8 *Formatting Characters in PROC REPORT Output*



Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

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.

Tip: 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 2D to the third formatting character, the hexadecimal character 7C to the seventh character, and does not alter the remaining characters: **`formchar(3,7)='2D7C'x`**

HEADLINE

underlines all column headings and the spaces between them at the top of each page of the report.

The HEADLINE option underlines with the second formatting character. (See the discussion of "FORMCHAR <(position(s))>='formatting-character(s)' " on page 1250.)

Default: hyphen (-)

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Tip: In traditional (monospace) SAS output, you can underline column headings without underlining the spaces between them, by using two hyphens **`('--')`** as the last line of each column heading instead of using HEADLINE.

Examples:

"Example 2: Ordering the Rows in a Report" on page 1311

"Example 8: Condensing a Report into Multiple Panels" on page 1330

HEADSKIP
writes a blank line beneath all column headings (or beneath the underlining that the HEADLINE option writes) at the top of each page of the report.

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Example: "Example 2: Ordering the Rows in a Report" on page 1311

HELP=*libref.catalog*
identifies the library and catalog containing user-defined help for the report. This help can be in CBT or HELP catalog entries. You can write a CBT or HELP entry for each item in the report with the BUILD procedure in SAS/AF software. Store all such entries for a report in the same catalog.

Specify the entry name for help for a particular report item in the DEFINITION window for that report item or in a DEFINE statement.

Restriction: This option only works in the Report Window.

LIST
writes to the SAS log the PROC REPORT code that creates the current report. This listing 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=
PROFILE=
REPORT=
WINDOWS|NOWINDOWS

- It omits SAS system options.
- It resolves automatic macro variables.

Restriction: This option has no effect in the interactive report window environment. In the interactive report window environment, you can write the report definition for the report that is currently in the REPORT window to the SOURCE window by selecting Tools, Report, Statements.

LS=*line-size*
specifies the length of a line of the report.

PROC REPORT honors the line size specifications that it finds in the following order of precedence:

- the LS= option in the PROC REPORT statement or LINESIZE= in the ROPTIONS window
- the LS= setting stored in the report definition loaded with REPORT= in the PROC REPORT statement

- the SAS system option LINESIZE=

Note: The PROC REPORT LS= option takes precedence over all other line size options.

Range: 64-256 (integer)

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Examples:

"Example 6: Displaying Multiple Statistics for One Variable" on page 1325

"Example 8: Condensing a Report into Multiple Panels" on page 1330

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"Missing Values" in Chapter 5 of *SAS Language Reference: Concepts* section in *SAS Language Reference: Concepts*.

Example: "Example 11: How PROC REPORT Handles Missing Values" on page 1342

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.

Example: "Example 7: Storing and Reusing a Report Definition" on page 1327

NOALIAS

lets you use a report that was created before compute blocks required aliases (prior to Release 6.11). If you use NOALIAS, then you cannot use aliases in compute blocks.

NOCENTER

See "CENTER | NOCENTER " on page 1249.

NOCOMPLETECOLS

See "COMPLETECOLS | NOCOMPLETECOLS" on page 1250.

NOCOMPLETEROWS

See "COMPLETEROWS | NOCOMPLETEROWS" on page 1250.

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.

When you suppress the display of column headings in the interactive report window environment, you cannot select any report items.

NOTHREADS
See "THREADS | NOTHREADS" on page 1260.

NOWINDOWS
See "WINDOWS | NOWINDOWS " on page 1261.

Alias: NOWD

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 to the output data set only; 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 12: Creating and Processing an Output Data Set" on page 1346
"Example 13: Storing Computed Variables as Part of a Data Set" on page 1349

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 omits these PROC REPORT statement options:

LIST	PROFILE=
NOALIAS	REPORT=
OUT=	WINDOWS|NOWINDOWS
OUTREPT=	

- It omits SAS system options.
- It resolves automatic macro variables.

Note: PROC REPORT version 7 and later cannot read entries created with SAS versions prior to Version 7.

Example: "Example 7: Storing and Reusing a Report Definition" on page 1327

PANELS=*number-of-panels*
specifies the number of panels on each page of the report. If the width of a report is less than half of the line size, then you can display the data in multiple sets of columns so that rows that would otherwise appear on multiple pages appear on the same page. Each set of columns is a *panel*. A familiar example of this type of report is a telephone book, which contains multiple panels of names and telephone numbers on a single page.

When PROC REPORT writes a multipanel report, it fills one panel before beginning the next.

The number of panels that fits on a page depends on the

- width of the panel
- space between panels
- line size

Default: 1

Note: This option has no effect on ODS destinations other than traditional SAS monospace output. However, the COLUMNS= option in the ODS PRINTER, ODS PDF, and ODS RTF statements produces similar results. For details, see the statements in *SAS Output Delivery System: User's Guide*.

Tip: If *number-of-panels* is larger than the number of panels that can fit on the page, then PROC REPORT creates as many panels as it can. Let PROC REPORT put your data in the maximum number of panels that can fit on the page by specifying a large number of panels (for example, 99).

See: For information about the space between panels and the line size, see the discussions of PSPACE= on page 1258 and the discussion of LS= on page 1253 .

Example: "Example 8: Condensing a Report into Multiple Panels" on page 1330

PCTLDEF=
See QNTLDEF= on page 1258.

PROFILE=*libref.catalog*
identifies the report profile to use. A profile

- specifies the location of menus that define alternative menu bars and menus for the REPORT and COMPUTE windows
- sets defaults for WINDOWS, PROMPT, and COMMAND

PROC REPORT uses the entry REPORT.PROFILE in the catalog that you specify as your profile. If no such entry exists, or if you do not specify a profile, then PROC REPORT uses the entry REPORT.PROFILE in SASUSER.PROFILE. If you have no profile, then PROC REPORT uses default menus and the default settings of the options.

You create a profile from the PROFILE window while using PROC REPORT in an interactive report window environment. To create a profile:

- Invoke PROC REPORT with the WINDOWS option.
- Select **Tools** ⇨ **Report Profile**
- Fill in the fields to meet your needs.
- Select **OK** to exit the PROFILE window. When you exit the window, PROC REPORT stores the profile in SASUSER.PROFILE.REPORT.PROFILE. Use the CATALOG procedure or the Explorer window to copy the profile to another location.

  Note: If, after opening the PROFILE window, you decide not to create a profile, then select **CANCEL** to close the window.

PROMPT
opens the REPORT window and starts the PROMPT facility. This facility guides you through creating a new report or adding more data set variables or statistics to an existing report.

If you start PROC REPORT with prompting, then the first window gives you a chance to limit the number of observations that are used during prompting. When you exit the prompter, PROC REPORT removes these limits:

- the PROMPT option in the PROC REPORT statement
- the setting in your report profile

If you omit PROMPT from the PROC REPORT statement, then the procedure uses the setting in your report profile, if you have one. If you do not have a report profile, then PROC REPORT does not use the prompt facility. For information about report profiles, see "PROFILE" on page 1389.

Restriction: When you use the PROMPT option, you open the REPORT window. When the REPORT window is open, you cannot send procedure output to any ODS destination.

Tip: You can store a setting of PROMPT in your report profile. PROC REPORT honors the first of these settings that it finds:

PS=*page-size*
specifies the number of lines in a page of the report.

PROC REPORT honors the first of these page size specifications that it finds:

- the PS= option in the PROC REPORT statement
- the PS= setting in the report definition specified with REPORT= in the PROC REPORT statement
- the SAS system option PAGESIZE=

Range: 15-32,767 (integer)

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Examples:

"Example 6: Displaying Multiple Statistics for One Variable" on page 1325

"Example 8: Condensing a Report into Multiple Panels" on page 1330

PSPACE=*space-between-panels*

specifies the number of blank characters between panels. PROC REPORT separates all panels in the report by the same number of blank characters. For each panel, the sum of its width and the number of blank characters separating it from the panel to its left cannot exceed the line size.

Default: 4

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Example: "Example 8: Condensing a Report into Multiple Panels" on page 1330

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 will 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 1671

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 1255

Example: "Example 7: Storing and Reusing a Report Definition" on page 1327

SHOWALL
overrides options in the DEFINE statement that suppress the display of a column.

See: NOPRINT and NOZERO in "DEFINE Statement" on page 1276

SPACING=*space-between-columns*
specifies the number of blank characters between columns. For each column, the sum of its width and the blank characters between it and the column to its left cannot exceed the line size.

Default: 2

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Interactions:
PROC REPORT separates all columns in the report by the number of blank characters specified by SPACING= in the PROC REPORT statement unless you use SPACING= in the DEFINE statement to change the spacing to the left of a specific item.

When CENTER is in effect, PROC REPORT ignores spacing that precedes the leftmost variable in the report.

Example: "Example 2: Ordering the Rows in a Report" on page 1311

SPANROWS
specifies that when the value of a GROUP or ORDER column is the same in multiple rows, the value will be 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.

Restriction: This option is supported only for the ODS MARKUP, RTF, PRINTER, and HTML destinations. It has no effect on the Report Window, the listing, or the data set destinations.

Tips:
The SPANROWS option also allows GROUP and ORDER variables values to repeat when the values break across pages in PDF, PS, and RTF destinations.

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.

In order to produce PROC REPORT output that is compliant with section 508, you need to specify the SPANROWS option in PROC REPORT and specify the HEADER_DATA_ASSOCIATIONS=yes OPTIONS option in the HTML statement. Section 508 is the accessibility standards for electronic information technology adopted by the U.S. Government under Section 508 of the U.S. Rehabilitation Act of 1973.

Here is example code: `ods html file="sec508.html" options(header_data_associations="yes"); proc report data=energy headline headskip nowd spanrows;`

SPLIT='*character*'
specifies the split character. PROC REPORT breaks a column heading when it reaches that character and continues the heading on the next line. The split character itself is not part of the column heading although each occurrence of the split character counts toward the 256-character maximum for a label.

Default: slash (/)

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output. However, in these ODS destinations, the SPLIT= option works in the column heading.

Interaction: The FLOW option in the DEFINE statement honors the split character.

Example: "Example 5: Creating a Column for Each Value of a Variable" on page 1321

STYLE<(*location(s)*)>=<*style-element-name*><[*style-attribute-specification(s)*]>
specifies the style element to use for the specified locations in the report. See "Using Style Elements in PROC REPORT " on page 1236 for details.

Restriction: This option affects only the HTML, RTF, and Printer output.

Tip: FONT names that contain characters other than letters or underscores must be enclosed in quotation marks.

Examples:
"Example 15: Specifying Style Elements for ODS Output in the PROC REPORT Statement" on page 1356

"Example 16: Specifying Style Elements for ODS Output in Multiple Statements" on page 1362

THREADS | NOTHREADS
enables or disables parallel processing of the input data set. This option overrides the SAS system option THREADS | NOTHREADS unless the system option is restricted. (See Restriction.) See "Support for Parallel Processing" in Chapter 13 of *SAS Language Reference: Concepts* for more information.

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 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 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 when a BY statement is specified.

VARDEF=*divisor*
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 46.8 *Possible Values for VARDEF=*

Value	Divisor	Formula for Divisor
DF	Degrees of freedom	$n - 1$

Value	Divisor	Formula for Divisor
N	Number of observations	n
WDF	Sum of weights minus one	$(\Sigma_i\ w_i) - 1$
WEIGHT | WGT	Sum of weights	$\Sigma_i\ w_i$

The procedure computes the variance as $CSS / divisor$, where CSS is the corrected sums of squares and equals $\Sigma\ (x_i - \bar{x})^2$. When you weight the analysis variables, CSS equals $\Sigma\ w_i(x_i - \bar{x}_w)^2$, 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 *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: "WEIGHT" on page 43

WINDOWS | NOWINDOWS

selects an interactive report window or nonwindowing environment.

When you use WINDOWS, SAS opens the REPORT window for the interactive report interface, which enables you to modify a report repeatedly and to see the modifications immediately. When you use NOWINDOWS, PROC REPORT runs without the REPORT window and sends its output to the open output destinations.

Alias: WD|NOWD

Restriction: When you use the WINDOWS option, you can send the output only to a SAS data set or to a Printer destination.

Tip: You can store a setting of WINDOWS in your report profile, if you have one. If you do not specify WINDOWS or NOWINDOWS in the PROC REPORT statement, then the procedure uses the setting in your report profile. If you do not have a report profile, then PROC REPORT looks at the setting of the SAS system option DMS. If DMS is ON, then PROC REPORT uses the interactive report window environment. If DMS is OFF, then it uses the nonwindowing environment.

See: For a discussion of the report profile see the discussion of PROFILE= on page 1256.

Example: "Example 1: Selecting Variables for a Report" on page 1307

WRAP

displays one value from each column of the report, on consecutive lines if necessary, before displaying another value from the first column. By default, PROC REPORT displays values for only as many columns as it can fit on one page. It fills a page

with values for these columns before starting to display values for the remaining columns on the next page.

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Interaction: When WRAP is in effect, PROC REPORT ignores PAGE in any item definitions.

Tip: Typically, you use WRAP in conjunction with the NAMED option in order to avoid wrapping column headings.

Example: "Example 7: Storing and Reusing a Report Definition" on page 1327

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 1311

"Example 4: Consolidating Multiple Observations into One Row of a Report" on page 1318

"Example 5: Creating a Column for Each Value of a Variable" on page 1321

"Example 8: Condensing a Report into Multiple Panels" on page 1330

"Example 9: Writing a Customized Summary on Each Page" on page 1333

Syntax

BREAK *location break-variable*</ *option(s)*>;

Summary of Optional Arguments

COLOR=*color*
: specifies the color of the break lines in the REPORT window.

CONTENTS='*link-text*'
: specifies the link text used in the table of contents.

DOL
: double overlines each value.

DUL
: double underlines each value.

OL
: overlines each value.

PAGE
: starts a new page after the last break line.

SKIP
: writes a blank line for the last break line.

STYLE<*location(s)*>=<*style-element-name*><[*style-attribute-specification(s)*]>
: specifies a style element 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.

UL
: underlines each value.

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

COLOR=*color*
specifies the color of the break lines in the REPORT window. The default color is the color of **Foreground** in the SASCOLOR window. You can use the following colors:

Table 46.9 *Colors Allowed for Break Lines*

BLACK	MAGENTA
BLUE	ORANGE
BROWN	PINK
CYAN	RED
GRAY	WHITE
GREEN	YELLOW

Default: The color of **Foreground** in the SASCOLOR window. (For more information, see the online Help for the SASCOLOR window.)

Restriction: This option affects output in the interactive report window environment only.

Note: Not all operating environments and devices support all colors, and on some operating systems and devices, one color might map to another color. For example, if the DEFINITION window displays the word BROWN in yellow characters, then selecting BROWN results in a yellow item.

CONTENTS='*link-text*'
specifies the text for the entries in the HTML contents file or PDF table of contents for the output that is produced by PROC REPORT. 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.

For information about HTML and PDF output, see "Output Delivery System" on page 34.

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.

Restrictions:
For HTML output, the CONTENTS= option has no effect in the HTML body file. It affects only the HTML contents file.

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 1276 .

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 1276 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 will then show 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 will be 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 will be 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.

DOL
(for double overlining) uses the 13th formatting character to overline each value

- that appears in the summary line
- that would appear in the summary line if you specified the SUMMARIZE option

Default: equal sign (=)

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Interaction: If you specify both the OL and DOL options, then PROC REPORT honors only OL.

See: The discussion of FORMCHAR= on page 1250.

DUL

(for double underlining) uses the 13th formatting character to underline each value

- that appears in the summary line
- that would appear in the summary line if you specified the SUMMARIZE option

Default: equal sign (=)

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Interaction: If you specify both the UL and DUL options, then PROC REPORT honors only UL.

See: The discussion of FORMCHAR= on page 1250.

OL

(for overlining) uses the second formatting character to overline each value

- that appears in the summary line
- that would appear in the summary line if you specified the SUMMARIZE option

Default: hyphen (-)

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Interaction: If you specify both the OL and DOL options, then PROC REPORT honors only OL.

See: The discussion of FORMCHAR= on page 1250.

Examples:

"Example 2: Ordering the Rows in a Report" on page 1311

"Example 9: Writing a Customized Summary on Each Page" on page 1333

PAGE

in monospace output, starts a new page. In HTML and PRINTER destinations, the PAGE option starts a new table.

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 9: Writing a Customized Summary on Each Page" on page 1333

SKIP

writes a blank line for the last break line.

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Examples:

"Example 2: Ordering the Rows in a Report" on page 1311

"Example 4: Consolidating Multiple Observations into One Row of a Report" on page 1318

"Example 5: Creating a Column for Each Value of a Variable" on page 1321
"Example 8: Condensing a Report into Multiple Panels" on page 1330

STYLE<*location(s)*>=<*style-element-name*><[*style-attribute-specification(s)*]>
specifies the style element to use for default summary lines that are created with the BREAK statement. See "Using Style Elements in PROC REPORT " on page 1236 for details.

Restriction: This option affects only the HTML, RTF, and Printer output.

Tip: FONT names that contain characters other than letters or underscores must be enclosed in quotation marks.

SUMMARIZE
writes a summary line in each group of break lines. A summary line for a set of observations contains values for

- 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
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 1273.)

* 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.

Examples:
"Example 2: Ordering the Rows in a Report" on page 1311
"Example 4: Consolidating Multiple Observations into One Row of a Report" on page 1318
"Example 9: Writing a Customized Summary on Each Page" on page 1333

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 1273.)

Example: "Example 4: Consolidating Multiple Observations into One Row of a Report" on page 1318

UL
(for underlining) uses the second formatting character to underline each value

- that appears in the summary line
- that would appear in the summary line if you specified the SUMMARIZE option

Default: hyphen (-)

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Interaction: If you specify both the UL and DUL options, then PROC REPORT honors only UL.

See: the discussion of FORMCHAR= on page 1250.

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. overlining or double overlining (OL or DOL)
2. summary line (SUMMARIZE)
3. underlining or double underlining (UL or DUL)
4. skipped line (SKIP)
5. 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 1273 and "LINE Statement" on page 1289.

BY Statement

Creates a separate report on a separate page for each BY group.

Restriction: If you use the BY statement, then you must use the NOWINDOWS option in the PROC REPORT statement.

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.

See: "BY" on page 36

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 the observations in the data set either must be sorted by all the variables that you specify or must be indexed appropriately. Variables in a BY statement are called *BY variables*.

Optional Arguments

DESCENDING
specifies that the data set is 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. For example, the data are grouped in 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.

CALL DEFINE Statement

Sets the value of an attribute for a particular column in the current row. The CALL DEFINE statement is often used to write report definitions that other people will use in an interactive report window environment. Only the FORMAT, URL, URLBP, and URLP attributes have an effect in the nonwindowing environment. In fact, URL, URLBP, and URLP are effective only in the nonwindowing environment. The STYLE= and URL

attributes are effective only when you are using the Output Delivery System to create HTML, RTF, or Printer output.

Restriction: Valid only in a compute block that is attached to a report item.

Examples: "Example 4: Consolidating Multiple Observations into One Row of a Report" on page 1318

"Example 16: Specifying Style Elements for ODS Output in Multiple Statements" on page 1362

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 '_C*n*_', 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

attribute-name
is the attribute to define. For attribute names, refer to Table 46.10 on page 1269.

Note: The attributes BLINK, HIGHLIGHT, and RVSVIDEO do not work on all devices.

Table 46.10 *Attribute Descriptions*

Attribute	Description	Values	Affects
BLINK	Controls blinking of current value	1 turns blinking on; 0 turns it off	Interactive report window environment
COLOR	Controls the color of the current value in the REPORT window	'blue', 'red', 'pink', 'green', 'cyan', 'yellow', 'white', 'orange', 'black', 'magenta', 'gray', 'brown'	Interactive report window environment
COMMAND	Specifies that a series of commands follows	A quoted string of SAS commands to submit to the command line	Interactive report window environment
FORMAT	Specifies a format for the column	A SAS format or a user-defined format	Interactive report window and nonwindowing environments
HIGHLIGHT	Controls highlighting of the current value	1 turns highlighting on; 0 turns it off	Interactive report window environment

Attribute	Description	Values	Affects
RVSVIDEO	Controls display of the current value	1 turns reverse video on; 0 turns it off	Interactive report window environment
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)	HTML, RTF, and PDF output
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)	HTML output
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)	HTML output

* For information about the BASE= and PATH= options, see the documentation for the ODS HTML Statement in *SAS Output Delivery System: User's Guide*.

ROW
is an automatic variable that indicates the entire current row.

value
sets the value for the attribute. For values for each attribute, refer to the following table.

Details

Using the STYLE Attribute

The STYLE attribute specifies the style element 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]");
```

In SAS 9.2, the STYLE and STYLE/REPLACE attributes specify the style element to be used for the Output Delivery System. If a style already exists for this cell or row, these STYLE attributes tell CALL DEFINE to replace the style specified by the STYLE= value. 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. See "Using Style Elements in PROC REPORT " on page 1236 for more details.

Restriction: This option affects only the HTML, RTF, Printer destinations.

Interaction: If you set a style element for the CALLDEF location in the PROC REPORT statement and you want to use that exact style element in a CALL DEFINE statement, then 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 16: Specifying Style Elements for ODS Output in Multiple Statements" on page 1362

COLUMN Statement

Describes the arrangement of all columns and of headings that span more than one column.

Restriction: You cannot use the COLUMN statement if you use REPORT= in the PROC REPORT statement.

Examples:

"Example 1: Selecting Variables for a Report" on page 1307
"Example 3: Using Aliases to Obtain Multiple Statistics for the Same Variable" on page 1314
"Example 5: Creating a Column for Each Value of a Variable" on page 1321
"Example 6: Displaying Multiple Statistics for One Variable" on page 1325
"Example 10: Calculating Percentages" on page 1338
"Example 11: How PROC REPORT Handles Missing Values" on page 1342

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 1230 for a list of available statistics.

report-item(s)
identifies items that each form a column in the report.

Examples:
"Example 1: Selecting Variables for a Report" on page 1307
"Example 11: How PROC REPORT Handles Missing Values" on page 1342

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 5: Creating a Column for Each Value of a Variable" on page 1321
"Example 6: Displaying Multiple Statistics for One Variable" on page 1325
"Example 10: Calculating Percentages" on page 1338

('*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 SPLIST= on page 1260.

In traditional (monospace) SAS output, if the first and last characters of a heading are one of the following characters, then PROC REPORT uses that character to expand the heading to fill the space over the column or columns. Note that the <> and the >< must be paired. **- = . _ * + <> ><**

Similarly, if the first character of a heading is < and the last character is >, or vice versa, then PROC REPORT expands the heading to fill the space over

the column by repeating the first character before the text of the heading and the last character after it.

Note: The use of expanding characters is supported only in monospace destinations. Therefore, PROC REPORT simply removes the expanding characters when the output is directed to an ODS MARKUP, HTML, RTF, or PRINTER destination. Refer to "Understanding ODS Destinations" in Chapter 3 of *SAS Output Delivery System: User's Guide* for more information.

report-item(s)
specifies the columns to span.

Example: "Example 10: Calculating Percentages" on page 1338

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 1233 .)

Example: "Example 3: Using Aliases to Obtain Multiple Statistics for the Same Variable" on page 1314

COMPUTE Statement

Starts a compute block containing one or more programming statements that PROC REPORT executes as it builds the 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). You create a compute block with the COMPUTE window or with the COMPUTE statement. 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 1232.

Examples: "Example 2: Ordering the Rows in a Report" on page 1311

"Example 3: Using Aliases to Obtain Multiple Statistics for the Same Variable" on page 1314

"Example 4: Consolidating Multiple Observations into One Row of a Report" on page 1318

"Example 5: Creating a Column for Each Value of a Variable" on page 1321

"Example 9: Writing a Customized Summary on Each Page" on page 1333

"Example 10: Calculating Percentages" on page 1338
"Example 13: Storing Computed Variables as Part of a Data Set" on page 1349
"Example 16: Specifying Style Elements for ODS Output in Multiple Statements" on page 1362

Syntax

COMPUTE *location* <*target*>
</ STYLE=<*style-element-name*><[*style-attribute-specification(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 "How PROC REPORT Builds a Report " on page 1297.)
- except in Printer and RTF output, near the bottom of each page, immediately before any footnotes, if you specify _PAGE_ as *target*.
- 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 "How PROC REPORT Builds a Report " on page 1297.)
- except in Printer and RTF output, near the top of each page, between any titles and the column headings, if you specify _PAGE_ as *target*.
- immediately before the first detail row if you omit a target.

Note: If a report contains more columns than will fit on a printed page, then 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 1314

"Example 9: Writing a Customized Summary on Each Page" on page 1333

report-item
specifies a data set variable, a computed variable, or a statistic to associate the compute block with. If you are working in the nonwindowing environment, then 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 4: Consolidating Multiple Observations into One Row of a Report" on page 1318

"Example 5: Creating a Column for Each Value of a Variable" on page 1321

Optional Arguments

STYLE<(*location(s)*)>=<*style-element-name*><[*style-attribute-specification(s)*]>
specifies the style to use for the text that is created by any LINE statements in this compute block. See "Using Style Elements in PROC REPORT " on page 1236 for details.

Restriction: This option affects only the HTML, RTF, and Printer destinations.

Tip: FONT names that contain characters other than letters or underscores must be enclosed in quotation marks.

Example: "Example 16: Specifying Style Elements for ODS Output in Multiple Statements" on page 1362

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>
in monospace output destinations, causes the compute block to execute once for each page, either immediately after printing any titles or immediately before printing any footnotes. *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 9: Writing a Customized Summary on Each Page" on page 1333

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 10: Calculating Percentages" on page 1338

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 10: Calculating Percentages" on page 1338

DEFINE Statement

Describes how to use and display a report item.

Restriction: A weight cannot be applied to a *report-item* alias without also applying it to the report-item. The WEIGHT= option must appear in the DEFINE statement for the *report-item*.

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 1311
"Example 3: Using Aliases to Obtain Multiple Statistics for the Same Variable" on page 1314
"Example 4: Consolidating Multiple Observations into One Row of a Report" on page 1318
"Example 5: Creating a Column for Each Value of a Variable" on page 1321
"Example 6: Displaying Multiple Statistics for One Variable" on page 1325
"Example 9: Writing a Customized Summary on Each Page" on page 1333
"Example 10: Calculating Percentages" on page 1338
"Example 12: Creating and Processing an Output Data Set" on page 1346
"Example 13: Storing Computed Variables as Part of a Data Set" on page 1349
"Example 14: Using a Format to Create Groups" on page 1353
"Example 16: Specifying Style Elements for ODS Output in Multiple Statements" on page 1362
"Example 17: Using Multilabel Formats" on page 1369
"Example 18: Using the WIDTH= and CELLWIDTH= Style Attributes with PROC REPORT" on page 1372

Syntax

DEFINE *report-item* / <*option(s)*>;

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.

COLOR=*color*
: specifies the color in the REPORT window of the column heading and of the values of the item that you define.

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.

ITEMHELP=*entry-name*
: references a HELP or CBT entry that contains Help information for the report 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.

SPACING=*horizontal-positions*
: for traditional SAS monospace output, defines the number of blank characters to leave between the column being defined and the column immediately to its left.

statistic
: associates a statistic with an analysis variable.

STYLE<(*location(s)*)>=<*style-element-name*><[*style-attribute-specification(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.

WIDTH=*column-width*
defines the width of the column in which PROC REPORT displays the report item.

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.

FLOW
wraps the value of a character variable in its column.

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. These 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: "Names in the SAS Language" in Chapter 3 of *SAS Language Reference: Concepts*, "SAS Variable Lists" in Chapter 4 of *SAS Language Reference: Concepts*, 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 1228.)

Example: "Example 5: Creating a Column for Each Value of a Variable" on page 1321

ANALYSIS

defines *report-item*, which must be a data set variable, as an analysis variable. (See "Analysis Variables" on page 1227.)

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 1311

"Example 3: Using Aliases to Obtain Multiple Statistics for the Same Variable" on page 1314

"Example 4: Consolidating Multiple Observations into One Row of a Report" on page 1318

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.

COLOR=*color*

specifies the color in the REPORT window of the column heading and of the values of the item that you are defining. You can use the following colors:

BLACK	MAGENTA
BLUE	ORANGE
BROWN	PINK

CYAN	RED
GRAY	WHITE
GREEN	YELLOW

Note: Not all operating environments and devices support all colors, and in some operating environments and devices, one color might map to another color. For example, if the DEFINITION window displays the word BROWN in yellow characters, then selecting BROWN results in a yellow item.

Default: The color of **Foreground** in the SASCOLOR window. (For more information, see the online Help for the SASCOLOR window.)

Restriction: This option affects output in the interactive report window environment only.

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.

In traditional (monospace) SAS output, if the first and last characters of a heading are one of the following characters, then PROC REPORT uses that character to expand the heading to fill the space over the column: `:- = \_ .* +`

Similarly, if the first character of a heading is < and the last character is >, or vice versa, then PROC REPORT expands the heading to fill the space over the column by repeating the first character before the text of the heading and the last character after it.

The following table shows the defaults variables and statistics:

Item	Header
Variable without a label	Variable name
Variable with a label	Variable label
Statistic	Statistic name

Tips:

If you want to use names when labels exist, then submit the following SAS statement before invoking PROC REPORT: `options nolabel;`

HEADLINE underlines all column headings and the spaces between them. In traditional (monospace) SAS output, you can underline column headings without underlining the spaces between them, by using the special characters `'--'` as the last line of each column heading instead of using HEADLINE. (See "Example 4: Consolidating Multiple Observations into One Row of a Report" on page 1318.)

See: SPLIT= on page 1260

Examples:

"Example 3: Using Aliases to Obtain Multiple Statistics for the Same Variable" on page 1314

"Example 4: Consolidating Multiple Observations into One Row of a Report" on page 1318

"Example 5: Creating a Column for Each Value of a Variable" on page 1321

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.

In the interactive report window environment, you add a computed variable to a report from the COMPUTED VAR window.

In the nonwindowing environment, 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

Examples:

"Example 5: Creating a Column for Each Value of a Variable" on page 1321

"Example 10: Calculating Percentages" on page 1338

CONTENTS='*link-text*'

specifies the text for the entries in the HTML contents file or PDF table of contents for the output that is produced by PROC REPORT. 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.

For information about HTML and PDF output, see "Output Delivery System" on page 34.

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.

Restrictions:

For HTML output, the CONTENTS= option has no effect in the HTML body file. It affects only the HTML contents file.

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.

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 will then show up in this separate Table of Contents page.

Tips:

If the DEFINE statement has the CONTENTS= option specified 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.

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 1226.)

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.

FLOW

wraps the value of a character variable in its column. The FLOW option honors the split character. If the text contains no split character, then PROC REPORT tries to split text at a blank.

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Example: "Example 10: Calculating Percentages" on page 1338

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 BEST*w*. 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. For numeric variables in the input data set and for computed variables (both numeric and character), the default column width is the value specified by COLWIDTH= in the PROC REPORT statement or in the ROPTIONS window.

In the interactive report window environment, if you are unsure what format to use, then type a question mark (?) in the format field in the DEFINITION window to access the FORMATS window.

Alias: F=

Examples:

"Example 2: Ordering the Rows in a Report" on page 1311

"Example 6: Displaying Multiple Statistics for One Variable" on page 1325

GROUP

defines *report-item*, which must be a data set variable, as a group variable. (See "Group Variables" on page 1226.)

Examples:

"Example 4: Consolidating Multiple Observations into One Row of a Report" on page 1318

"Example 6: Displaying Multiple Statistics for One Variable" on page 1325

"Example 14: Using a Format to Create Groups" on page 1353

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 will fit on one page.

Example: "Example 6: Displaying Multiple Statistics for One Variable" on page 1325

ITEMHELP=*entry-name*

references a HELP or CBT entry that contains help information for the report item. Use PROC BUILD in SAS/AF software to create a HELP or CBT entry for a report item. All HELP and CBT entries for a report must be in the same catalog, and you must specify that catalog with the HELP= option in the PROC REPORT statement or from the **User Help** fields in the ROPTIONS window.

Of course, you can access these entries only from an interactive report window environment. To access a Help entry from the report, select the item and issue the HELP command. PROC REPORT first searches for and displays an entry named *entry-name*.CBT. If no such entry exists, then PROC REPORT searches for *entry-name*.HELP. If neither a CBT nor a HELP entry for the selected item exists, then the opening frame of the Help for PROC REPORT is displayed.

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: This option only affects the LISTING output. It has no effect on other ODS output.

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, the LISTING destination, data sets, and the REPORT WINDOW.

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 the column does not have a MULTILABEL format associated with it, 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 the column already has a MULTILABEL format associated with it (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 658 in the VALUE statement of the FORMAT procedure.

Example: "Example 17: Using Multilabel Formats" on page 1369

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 9: Writing a Customized Summary on Each Page" on page 1333.)

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 1233.)

SHOWALL in the PROC REPORT statement or the ROPTIONS window overrides all occurrences of NOPRINT.

Examples:

"Example 3: Using Aliases to Obtain Multiple Statistics for the Same Variable" on page 1314

"Example 9: Writing a Customized Summary on Each Page" on page 1333

"Example 12: Creating and Processing an Output Data Set" on page 1346

NOZERO

suppresses the display of the report item if its values are all zero or missing.

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

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 1233 .)

SHOWALL in the PROC REPORT statement or in the ROPTIONS window 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 1226.)

Example: "Example 2: Ordering the Rows in a Report" on page 1311

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.

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 will continue to produce the reports that you expect even if the default changes.

Example: "Example 2: Ordering the Rows in a Report" on page 1311

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.

Interaction: PAGE is ignored if you use WRAP in the PROC REPORT statement or in the ROPTIONS window.

Tip: In listing destinations, a PAGE option in the DEFINE statement causes PROC REPORT to print this column and all columns to its right on a new page. However, for ODS MARKUP, HTML, PRINTER , and RTF destinations, 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.

Note: 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.

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: This option only affects the LISTING output. It has no affect on other ODS output.

SPACING=*horizontal-positions*
defines the number of blank characters to leave between the column being defined and the column immediately to its left. For each column, the sum of its width and the blank characters between it and the column to its left cannot exceed the line size.

Default: 2

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Interactions:
When PROC REPORT's CENTER option is in effect, PROC REPORT ignores spacing that precedes the leftmost variable in the report.

SPACING= in an item's definition overrides the value of SPACING= in the PROC REPORT statement or in the ROPTIONS window.

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 1230 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 1311

"Example 3: Using Aliases to Obtain Multiple Statistics for the Same Variable" on page 1314

"Example 4: Consolidating Multiple Observations into One Row of a Report" on page 1318

STYLE<(*location(s)*)>=<*style-element-name*><[*style-attribute-specification(s)*]>
specifies the style element to use for column headings and for text inside cells for this report item. See "Using Style Elements in PROC REPORT " on page 1236 for details.

Restriction: This option affects only the HTML, RTF, and Printer destinations.

Tip: FONT names that contain characters other than letters or underscores must be enclosed in quotation marks.

Example: "Example 16: Specifying Style Elements for ODS Output in Multiple Statements" on page 1362

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 *report-item* alias without also applying it to the *report-item*. The WEIGHT= option must appear in the DEFINE statement for the *report-item*.

Note: Prior to Version 7 of SAS, the REPORT 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 in the PROC REPORT statement is appropriate.

Use the WEIGHT= option in separate variable definitions in order to specify different weights for the variables.

WIDTH=*column-width*
defines the width of the column in which PROC REPORT displays *report-item*. This option only affects traditional SAS monospace output.

Default: A column width that is just large enough to handle the format. If there is no format, then PROC REPORT uses the value of the COLWIDTH= option in the PROC REPORT statement.

Range: 1 to the value of the SAS system option LINESIZE=

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output. For ODS destinations, use the STYLE= option with the WIDTH= style attribute or the CELLWIDTH= style attribute. Refer to "Style Attributes Tables " in Chapter 13 of *SAS Output Delivery System: User's Guide* for details. See how style attributes WIDTH= and CELLWIDTH= can be used with PROC REPORT in "Example 18: Using the WIDTH= and CELLWIDTH= Style Attributes with PROC REPORT" on page 1372.

Interaction: WIDTH= in an item definition overrides the value of COLWIDTH= in the PROC REPORT statement or the ROPTIONS window.

Tip: When you stack items in the same column in a report, the width of the item that is at the bottom of the stack determines the width of the column.

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 1311

Syntax

ENDCOMP;

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.

See: For an example that uses the FREQ statement, see "Example" on page 41

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.

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.

Examples: "Example 2: Ordering the Rows in a Report" on page 1311

"Example 3: Using Aliases to Obtain Multiple Statistics for the Same Variable" on page 1314

"Example 4: Consolidating Multiple Observations into One Row of a Report" on page 1318

"Example 5: Creating a Column for Each Value of a Variable" on page 1321

"Example 9: Writing a Customized Summary on Each Page" on page 1333

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 1233.

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 1311

'*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 **'DF'x**) that is specified within *character-string* will not resolve 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 will resolve.

Example: "Example 2: Ordering the Rows in a Report" on page 1311

*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 1314

pointer-control
specifies the column in which PROC REPORT displays the next specification. You can use either of the following forms for pointer controls:

@*column-number*
specifies the number of the column in which to begin displaying the next item in the specification list.

+*column-increment*
specifies the number of columns to skip before beginning to display the next item in the specification list.

Both *column-number* and *column-increment* can be either a variable or a literal value.

Restriction: The pointer controls are designed for monospace output. They have no effect in the HTML, RTF, or Printer output.

Examples:
"Example 3: Using Aliases to Obtain Multiple Statistics for the Same Variable" on page 1314
"Example 5: Creating a Column for Each Value of a Variable" on page 1321

Details

Differences between the LINE and PUT Statements

The LINE statement does not support the following features of the PUT statement:

- automatic labeling signaled by an equal sign (=), also known as named output
- the _ALL_, _INFILE_, and _PAGE_ arguments and the OVERPRINT option
- grouping items and formats to apply one format to a list of items
- pointer control using expressions
- line pointer controls (# and /)
- trailing at signs (@ and @@)
- format modifiers
- array elements

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 for a Report" on page 1307
"Example 10: Calculating Percentages" on page 1338

Syntax

RBREAK *location* </ *option(s)*>;

Summary of Optional Arguments

COLOR=*color*
specifies the color of the break lines in the REPORT window.

CONTENTS='*link-text*'
specifies the link text used in the table of contents.

DOL
double overlines each value.

DUL
double underlines each value.

OL
overlines each value.

PAGE
starts a new page after the last break line of a break located at the beginning of the report.

SKIP
writes a blank line for the last break line of a break located at the beginning of the report.

STYLE<(*location(s)*)>=<*style-element-name*><[*style-attribute-specification(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.

UL
underlines each value.

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

COLOR=*color*
specifies the color of the break lines in the REPORT window. You can use the following colors:

BLACK	MAGENTA
BLUE	ORANGE
BROWN	PINK

CYAN	RED
GRAY	WHITE
GREEN	YELLOW

Default: The color of **Foreground** in the SASCOLOR window. (For more information, see the online Help for the SASCOLOR window.)

Restriction: This option affects output in the interactive report window environment only.

Note: Not all operating environments and devices support all colors, and in some operating environments and devices, one color might map to another color. For example, if the DEFINITION window displays the word BROWN in yellow characters, then selecting BROWN results in a yellow item.

CONTENTS='*link-text*'
specifies the text for the entries in the HTML contents file or PDF table of contents for the output that is produced by PROC REPORT. 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.

For information about HTML and PDF output, see "Output Delivery System" on page 34.

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**.

Restrictions:
For HTML output, the CONTENTS= option has no effect in the HTML body file. It affects only the HTML contents file.

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 a table in the table of contents.

Interaction: 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 will then show up in this separate Table of Contents page.

Tip: HTML output can now have additional anchor tags.

DOL
(for double overlining) uses the 13th formatting character to overline each value

- that appears in the summary line
- that would appear in the summary line if you specified the SUMMARIZE option

Default: equal sign (=)

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Interaction: If you specify both the OL and DOL options, then PROC REPORT honors only OL.

See: The discussion of FORMCHAR= on page 1250.

Example: "Example 1: Selecting Variables for a Report" on page 1307

DUL
(for double underlining) uses the 13th formatting character to underline each value

- that appears in the summary line
- that would appear in the summary line if you specified the SUMMARIZE option

Default: equal sign (=)

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Interaction: If you specify both the UL and DUL options, then PROC REPORT honors only UL.

See: The discussion of FORMCHAR= on page 1250.

OL
(for overlining) uses the second formatting character to overline each value

- that appears in the summary line
- that would appear in the summary line if you specified the SUMMARIZE option

Default: hyphen (-)

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Interaction: If you specify both the OL and DOL options, then PROC REPORT honors only OL.

See: the discussion of FORMCHAR= on page 1250.

Example: "Example 10: Calculating Percentages" on page 1338

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.

SKIP
writes a blank line after the last break line of a break located at the beginning of the report.

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

STYLE<(*location(s)*)>=<*style-element-name*><[*style-attribute-specification(s)*]>
specifies the style element to use for default summary lines that are created with the RBREAK statement. See "Using Style Elements in PROC REPORT " on page 1236 for details.

Restriction: This option affects only the HTML, RTF, and Printer destinations.

Tip: FONT names that contain characters other than letters or underscores must be enclosed in quotation marks.

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

- 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
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 1273.)

Examples:
"Example 1: Selecting Variables for a Report" on page 1307
"Example 10: Calculating Percentages" on page 1338

UL
(for underlining) uses the second formatting character to underline each value

- that appears in the summary line
- that would appear in the summary line if you specified the SUMMARIZE option.

Default: hyphen (-)

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Interaction: If you specify both the UL and DUL options, then PROC REPORT honors only UL.

See: The discussion of FORMCHAR= on page 1250.

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. overlining or double overlining (OL or DOL, traditional SAS monospace output only)
2. summary line (SUMMARIZE)
3. underlining or double underlining (UL or DUL, traditional SAS monospace output only)
4. skipped line (SKIP, traditional SAS monospace output only)
5. 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 the COMPUTE statement on page 1273 and the LINE statement "LINE Statement" on page 1289.

WEIGHT Statement

Specifies weights for analysis variables in the statistical calculations.

See: For information about calculating weighted statistics see "Calculating Weighted Statistics" on page 44 . For an example that uses the WEIGHT statement, see "Weighted Statistics Example" on page 44.

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. If the value of variable is

Note: Prior to Version 7 of SAS, the procedure did not exclude the observations with missing weights from the count of observations.

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 1260 and the calculation of weighted statistics in "Keywords and Formulas" on page 1666 for more information.

Details

Weight Information Is Not Saved

When you store a report definition, PROC REPORT does not store the WEIGHT statement.

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:

- DB2
- Oracle
- Teradata
- Netezza

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 will 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*.

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 1298. The sequence of events is the same whether you use programming statements or the interactive report window environment.

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 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. If there is a compute block attached to the break, then PROC REPORT then executes the statements in the compute block. See "Construction of Summary Lines" on page 1298 for details.

Note: Because of the way PROC REPORT builds a report, you can

- 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 1233.

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 1231 and "COMPUTE Statement" on page 1273 .

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 Log 46.2 on page 1299 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 Log 46.2 on page 1299. The user-defined formats that are used are created by a PROC FORMAT step on page 1308.

```
libname proclib
'SAS-library';

options nodate pageno=1 linesize=64
        pagesize=60 fmtsearch=(proclib);


ods html close;
ods listing;
proc report data=grocery headline headskip nowd;
    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 / dol dul summarize;
    compute after;
       sector='TOTALS:';
    endcomp;

    where sector contains 'n';
    title 'Report for Northeast and Northwest Sectors';
  run;
ods listing close;
```

Log 46.2 *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 46.9 *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 46.10 *First Detail Row with Values Filled in from Left to Right*

Sector	Department	Sales	Profit	N
Northeast		.	.	.

Sector	Department	Sales	Profit	N
Northeast	Canned	.	.	.

Sector	Department	Sales	Profit	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 46.11 *A Computed Variable Added to the First Detail Row*

Sector	Department	Sales	Profit	N
Northeast	Canned	$840.00	$336.00	.

5. 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 46.12 *First Complete Detail Row*

Sector	Department	Sales	Profit	N
Northeast	Canned	$840.00	$336.00	2

6. The procedure writes the completed row to the report.
7. PROC REPORT repeats steps 2, 3, 4, 5, and 6 for each detail row in the report.
8. At the break at the end of the report, PROC REPORT constructs the break lines described by the RBREAK statement. These lines include double underlining, double overlining, and 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. (See the following figure.)

Figure 46.13 *Preliminary Summary Line*

Sector	Department	Sales	Profit	N
		$4,285.00	$1,071.25	20

9. 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 46.14 *Final Summary Line*

Sector	Department	Sales	Profit	N
TOTALS:		$4,285.00	$1,071.25	20

10. Finally, PROC REPORT writes all the break lines, with underlining, overlining, and the final summary line, to the report.

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 Log 46.3 on page 1303 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 Log 46.3 on page 1303. The user-defined formats that are used are created by a PROC FORMAT step on page 1308.

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';

options nodate pageno=1 linesize=64
        pagesize=60 fmtsearch=(proclib);
ods html close;
ods listing;
proc report data=grocery noheader nowindows;
  column sector department sales
         Sctrpct sales=Salespct;

  define sector     / 'Sector' group
                      format=$sctrfmt.;
  define department / group format=$deptfmt.;
  define sales      / analysis sum
                      format=dollar9.2 ;
  define sctrpct    / computed
                      format=percent9.2 ;
  define salespct   / pctsum format=percent9.2;

  compute before;
```

```
      line ' ';
      line @16 'Total for all stores is '
           sales.sum dollar9.2;
      line ' ';
      line @29 'Sum of' @40 'Percent'
           @51 'Percent of';
      line @6 'Sector' @17 'Department'
           @29 'Sales'
           @40 'of Sector' @51 'All Stores';
      line @6 55*'=';
      line ' ';
   endcomp;

   break before sector / summarize ul;
   compute before sector;
      sctrtot=sales.sum;
      sctrpct=sales.sum/sctrtot;
   endcomp;

   compute sctrpct;
      sctrpct=sales.sum/sctrtot;
   endcomp;

   break after sector/skip;
   where sector contains 'n';
   title 'Report for Northeast and Northwest Sectors';
run;
ods listing close;
```

Log 46.3 *Report with Temporary Variables*

```
        Report for Northeast and Northwest Sectors          1


            Total for all stores is $4,285.00

                         Sum of     Percent    Percent of
Sector     Department    Sales      of Sector  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. (See the following figure.)

Figure 46.15 *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. (See the following figure.)

   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 46.16 *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%	.

4. 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. (The NOHEADER option in the PROC REPORT statement suppresses the default column headings, which would have appeared before the customized summary.)

5. 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 and has 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. (See the following figure.)

Figure 46.17 *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 will be missing because the value of Sctrtot is missing at the time that the COMPUTE Sctrpct block executes.

Figure 46.18 *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. The UL option in the BREAK statement underlines the summary line.
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 46.19 *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 46.20 *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 46.21 *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. (See the following figure.)

Figure 46.22 *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. The only action at this break is that the SKIP option in the BREAK AFTER statement writes a blank line after the last detail row of the group.
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. (See the following figure.) The statements in the compute block for this break calculate the correct value. (See the following step.)

Figure 46.23 *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 will not be synchronized with the other values in the summary line, and the report will be 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 46.24 *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. The UL option in the BREAK statement underlines the summary line.

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 for a Report

Features: PROC REPORT statement options

NOWD

COLUMN statement
default variable usage

RBREAK statement options
DOL
SUMMARIZE

Other features: FORMAT statement

FORMAT procedure
LIBRARY=

SAS system options
FMTSEARCH=

Automatic macro variables
SYSDATE

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);

proc report data=grocery nowd;

   column manager department sales;

   rbreak after / dol summarize;

   where sector='se';

   format manager $mgrfmt. department $deptfmt.
          sales dollar11.2;

   title 'Sales for the Southeast Sector';
   title2 "for &sysdate";
run;
```

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 report options. The NOWD option runs the REPORT procedure without the REPORT window and sends its output to the open output destination.

```
proc report data=grocery nowd;
```

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. DOL writes a line of equal signs (=) above the summary information. SUMMARIZE sums the value of Sales for all observations in the report.

```
    rbreak after / dol summarize;
```

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 when 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;
```

Output

Sales for the Southeast Sector for 10DEC10

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
- COLWIDTH=
- HEADLINE
- HEADSKIP
- SPACING=

BREAK statement options
- OL
- SKIP
- SUMMARIZE

COMPUTE statement arguments
- AFTER

DEFINE statement options
- ANALYSIS
- FORMAT=
- ORDER
- ORDER=
- SUM

ENDCOMP statement

LINE statement
- with quoted text
- with variable values

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
- underlines the column headings and writes a blank line beneath the underlining
- 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);

proc report data=grocery nowd
            colwidth=10
            spacing=5
            headline headskip;

   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 / ol
                         summarize
                         skip;

   compute after;
      line 'Total sales for these stores were: '
           sales.sum dollar9.2;
   endcomp;

   where sector='se';

   title 'Sales for the Southeast Sector';
 run;
```

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 report options. The NOWD option runs PROC REPORT without the REPORT window and sends its output to the open output destinations. COLWIDTH=10 sets the default column width to 10 characters. SPACING= puts five blank characters between columns. HEADLINE underlines all column headings and the spaces between them at the top of each page of the report. HEADSKIP writes a blank line beneath the underlining that HEADLINE writes.

```
proc report data=grocery nowd
            colwidth=10
            spacing=5
            headline headskip;
```

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. OL writes a row of hyphens above the summary line. SUMMARIZE writes the value of Sales (the only analysis or computed variable) 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. SKIP writes a blank line after the summary line.

```
break after manager / ol
                      summarize
                      skip;
```

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. An ENDCOMP statement must end the compute block.

```
compute after;
   line 'Total sales for these stores were: '
        sales.sum dollar9.2;
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;
```

Output

Sales for the Southeast Sector

Manager	Department	Sales
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

Example 3: Using Aliases to Obtain Multiple Statistics for the Same Variable

Features:

COLUMN statement
- with aliases

COMPUTE statement arguments
- AFTER

DEFINE statement options
- ANALYSIS
- MAX
- MIN
- NOPRINT
- customizing column headings

LINE statement

pointer controls
quoted text
repeating a character string
variable values and formats
writing a blank line

Other features: 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);

proc report data=grocery nowd headline headskip;

   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 ' ';
      line @7 53*'-';

      line @7 '| Departmental sales ranged from'
           salesmin dollar7.2  +1 'to' +1 salesmax dollar7.2
           '. |';
      line @7 53*'-';
   endcomp;

   where sector='se';
```

```
   title 'Sales for the Southeast Sector';
   title2 "for &sysdate";
run;
```

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 report options. The NOWD option runs PROC REPORT without the REPORT window and sends its output to the open output destinations. HEADLINE underlines all column headings and the spaces between them at the top of each page of the report. HEADSKIP writes a blank line beneath the underlining that HEADLINE writes.

```
proc report data=grocery nowd headline headskip;
```

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=SALESMAX 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;
```

Print a horizontal line at the end of the report. This COMPUTE statement begins a compute block that executes at the end of the report. The first LINE statement writes a blank line. The second LINE statement writes 53 hyphens (-), beginning in column 7. Note that the pointer control (@) has no effect on ODS destinations other than traditional SAS monospace output.

```
compute after;
   line ' ';
   line @7 53*'-';
```

Produce a customized summary. The first line of this LINE statement writes the text in quotation marks, beginning in column 7. The second line writes the value of Salesmin with the DOLLAR7.2 format, beginning in the next column. The cursor then moves one column to the right (+1), where PROC REPORT writes the text in quotation marks. Again, the cursor moves one column to the right, and PROC REPORT writes the value of Salesmax with the DOLLAR7.2 format. (Note that the program must reference the variables by their aliases.) The third line writes the text in quotation marks, beginning in the next column. Note that the pointer control (@) is designed for the Listing destination (traditional SAS output). It has no effect on ODS destinations other than traditional SAS monospace output. The ENDCOMP statement ends the compute block.

```
   line @7 '| Departmental sales ranged from'
        salesmin dollar7.2  +1 'to' +1 salesmax dollar7.2
        '. |';
   line @7 53*'-';
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 when 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;
```

Output

Sales for the Southeast Sector
for 10DEC10

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: Consolidating Multiple Observations into One Row of a Report

Features:

BREAK statement options
- OL
- SKIP
- SUMMARIZE
- SUPPRESS

CALL DEFINE statement

Compute block
- associated with a data set variable

COMPUTE statement arguments
- AFTER
- a data set variable as *report-item*

DEFINE statement options
- ANALYSIS
- GROUP
- SUM
- customizing column headings

LINE statement
- quoted text
- variable values

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);

proc report data=grocery nowd headline headskip;

   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 / ol
                        summarize
                        suppress
                        skip;

   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;
```

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 report options. The NOWD option runs PROC REPORT without the REPORT window and sends its output to the open output destinations. HEADLINE underlines all column headings and the spaces between them at the top of each page of the report. HEADSKIP writes a blank line beneath the underlining that HEADLINE writes.

```
proc report data=grocery nowd headline headskip;
```

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. OL writes a row of hyphens above the summary line. SUMMARIZE writes the value of Sales in the summary line. PROC REPORT sums the values of Sales for each manager because Sales is an analysis variable used to calculate the Sum statistic. SUPPRESS prevents PROC REPORT from displaying the value of Sector in the summary line. SKIP writes a blank line after the summary line.

```
break after sector / ol
                     summarize
                     suppress
                     skip;
```

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;
```

Output

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 5: Creating a Column for Each Value of a Variable

Features: PROC REPORT statement options
SPLIT=

BREAK statement options
SKIP
COLUMN statement
stacking variables
COMPUTE statement arguments
with a computed variable as *report-item*
AFTER
DEFINE statement options
ACROSS
ANALYSIS
COMPUTED
SUM
LINE statement
pointer controls

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 blank lines
- double-spaces between detail rows
- uses pointer controls to control the placement of text and variable values in a customized summary

Program

```
libname proclib
'SAS-library';

options fmtsearch=(proclib);

proc report data=grocery nowd
            headline
            headskip
            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=sum(_c3_, _c4_);
   endcomp;

   compute after;
      line @4 57*'-';
      line @4 '|   Combined sales for meat and dairy : '
           @46 _c3_ dollar11.2 '   |';
      line @4 '|   Combined sales for produce : '
           @46 _c4_ dollar11.2 '   |';
      line @4 '|' @60 '|';
      line @4 '|   Combined sales for all perishables: '
           @46 _c5_ dollar11.2 '   |';
      line @4 57*'-';
   endcomp;

   where sector contains 'n'
         and (department='p1' or department='p2');

   title 'Sales Figures for Perishables in Northern Sectors';
run;
```

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 report options. The NOWD option runs PROC REPORT without the REPORT window and sends its output to the open output destinations. HEADLINE underlines the column headings. HEADSKIP writes a blank line beneath the underlining that HEADLINE writes. 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 nowd
            headline
            headskip
            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. In traditional (monospace) SAS output, PROC REPORT expands the heading with underscores to fill all columns that are created by the across variable.

```
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';
```

Produce a report summary. This BREAK statement creates a default summary after the last row for each value of Manager. The only option that is in use is SKIP, which writes a blank line. You can use this technique to double-space in many reports that contains a group or order variable.

```
break after manager / skip;
```

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. The first LINE statement writes 57 hyphens (-) starting in column 4. Subsequent 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. Note that the pointer control (@) is designed for the Listing destination. It has no effect on ODS destinations other than traditional SAS monospace output.

```
compute after;
   line @4 57*'-';
   line @4 '|   Combined sales for meat and dairy : '
        @46 _c3_ dollar11.2 '   |';
   line @4 '|   Combined sales for produce : '
        @46 _c4_ dollar11.2 '   |';
   line @4 '|' @60 '|';
   line @4 '|   Combined sales for all perishables: '
```

```
            @46 _c5_ dollar11.2 '   |';
         line @4 57*'-';
      endcomp;

      where sector contains 'n'
            and (department='p1' or department='p2');
```

Specify the title.

```
      title 'Sales Figures for Perishables in Northern Sectors';
   run;
```

Output

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 6: Displaying Multiple Statistics for One Variable

Features: PROC REPORT statement options
- LS=
- PS=

COLUMN statement
- specifying statistics for stacked variables

DEFINE statement options
- FORMAT=
- GROUP
- ID

Data set: GROCERY

Format: $MGRFMT

Details

The report in this example displays six statistics for the sales for each manager's store. The output is too wide to fit all the columns on one page, so three of the statistics appear on the second page of the report. In order to make it easy to associate the statistics on the second page with their group, the report repeats the values of Manager and Sector on every page of the report.

Program

```
libname proclib
'SAS-library';

options fmtsearch=(proclib);

proc report data=grocery nowd headline headskip
            ls=66 ps=18;

column sector manager (Sum Min Max Range Mean Std),sales;

   define manager / group format=$mgrfmt. id;
   define sector / group format=$sctrfmt.;
   define sales / format=dollar11.2 ;

   title 'Sales Statistics for All Sectors';
run;
```

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 report options. The NOWD option runs PROC REPORT without the REPORT window and sends its output to the open output destinations. HEADLINE underlines all column headings and the spaces between them at the top of each page of the report. HEADSKIP writes a blank line beneath the underlining that HEADLINE writes. LS= sets the line size for the report to 66, and PS= sets the page size to 18.

```
proc report data=grocery nowd headline headskip
            ls=66 ps=18;
```

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. ID specifies that Manager is an ID variable. An ID variable and all columns to its left appear at the left of every page of a report. 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=$mgrfmt. id;
define sector / group format=$sctrfmt.;
define sales / format=dollar11.2 ;
```

Specify the title.

```
title 'Sales Statistics for All Sectors';
run;
```

Output

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 7: Storing and Reusing a Report Definition

Features: PROC REPORT statement options
- NAMED
- OUTREPT=
- REPORT=
- WRAP

Other features: TITLE statement
WHERE statement

Data set: GROCERY

Format: $MGRFMT

Format: $DEPTFMT

Format: $SCTRFMT

Details

The first PROC REPORT step in this example creates a report that displays one value from each column of the report, using two rows to do so, before displaying another value from the first column. (By default, PROC REPORT displays values for only as many columns as it can fit on one page. It fills a page with values for these columns before starting to display values for the remaining columns on the next page.)

Each item in the report is identified in the body of the report rather than in a column heading.

The report definition created by the first PROC REPORT step is stored in a catalog entry. The second PROC REPORT step uses it to create a similar report for a different sector of the city.

Program to Store a Report Definition

```
libname proclib
'SAS-library';

options fmtsearch=(proclib);

proc report data=grocery nowd
            named
            wrap
            ls=64 ps=36
            outrept=proclib.reports.namewrap;

   column sector manager department sales;

   define sector / format=$sctrfmt.;
   define manager / format=$mgrfmt.;
   define department / format=$deptfmt.;
   define sales / format=dollar11.2;

   where manager='1';

   title "Sales Figures for Smith on &sysdate";
run;
```

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 report options. The NOWD option runs PROC REPORT without the REPORT window and sends its output to the open output destinations. NAMED writes *name*= in front of each value in the report, where *name*= is the column heading for the value. When you use NAMED, PROC REPORT suppresses the display of column headings at the top of each page.

```
proc report data=grocery nowd
            named
            wrap
            ls=64 ps=36
            outrept=proclib.reports.namewrap;
```

Specify the report columns. The report contains a column for Sector, Manager, Department, and Sales.

```
    column sector manager department sales;
```

Define the display and analysis variables. Because no usage is specified in the DEFINE statements, PROC REPORT uses the defaults. The character variables (Sector, Manager, and Department) are display variables. Sales is an analysis variable that is used to calculate the sum statistic. FORMAT= specifies the formats to use in the report.

```
    define sector / format=$sctrfmt.;
    define manager / format=$mgrfmt.;
    define department / format=$deptfmt.;
    define sales / format=dollar11.2;
```

Select the observations to process. A report definition might differ from the SAS program that creates the report. In particular, PROC REPORT stores neither WHERE statements nor TITLE statements.

```
    where manager='1';
```

Specify the title. SYSDATE is an automatic macro variable that returns the date when the SAS job or SAS session began. The TITLE statement uses double rather than single quotation marks so that the macro variable resolves.

```
    title "Sales Figures for Smith on &sysdate";
run;
```

Output

The following output is the output from the first PROC REPORT step, which creates the report definition.

Sales Figures for Smith on 10DEC10

Sector=Southeast	Manager=Smith	Department=Paper	Sales= $50.00
Sector=Southeast	Manager=Smith	Department=Meat/Dairy	Sales= $100.00
Sector=Southeast	Manager=Smith	Department=Canned	Sales= $120.00
Sector=Southeast	Manager=Smith	Department=Produce	Sales= $80.00

Program to Use a Report Definition

```
options fmtsearch=(proclib);

 proc report data=grocery report=proclib.reports.namewrap
             nowd;
   where sector='sw';
```

```
   title "Sales Figures for the Southwest Sector on &sysdate";
run;
```

Program Description

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 report options, load the report definition, and select the observations to process. REPORT= uses the report definition that is stored in PROCLIB.REPORTS.NAMEWRAP to produce the report. The second report differs from the first one because it uses different WHERE and TITLE statements.

```
 proc report data=grocery report=proclib.reports.namewrap
             nowd;
   where sector='sw';
   title "Sales Figures for the Southwest Sector on &sysdate";
run;
```

Output

Sales Figures for the Southwest Sector on 10DEC10

Sector=Southwest	Manager=Taylor	Department=Paper	Sales= $53.00
Sector=Southwest	Manager=Taylor	Department=Meat/Dairy	Sales= $130.00
Sector=Southwest	Manager=Taylor	Department=Canned	Sales= $120.00
Sector=Southwest	Manager=Taylor	Department=Produce	Sales= $50.00
Sector=Southwest	Manager=Adams	Department=Paper	Sales= $40.00
Sector=Southwest	Manager=Adams	Department=Meat/Dairy	Sales= $350.00
Sector=Southwest	Manager=Adams	Department=Canned	Sales= $225.00
Sector=Southwest	Manager=Adams	Department=Produce	Sales= $80.00

Example 8: Condensing a Report into Multiple Panels

Features: PROC REPORT statement options
- FORMCHAR=
- HEADLINE
- LS=
- PANELS=
- PS=
- PSPACE=

BREAK statement options
- SKIP

Other features: SAS system option FORMCHAR=

Data set: GROCERY

Format: $MGRFMT

Format: $DEPTFMT

Details

The report in this example does the following:

- uses panels to condense a two-page report to one page. Panels compactly present information for long, narrow reports by placing multiple rows of information side by side.
- uses a default summary to place a blank line after the last row for each manager.
- changes the default underlining character for the duration of this PROC REPORT step.

Program

```
libname proclib
'SAS-library';

options fmtsearch=(proclib);

proc report data=grocery nowd headline
            formchar(2)='~'
            panels=99 pspace=6
            ls=64 ps=18;

   column manager department sales;

   define manager / order
                    order=formatted
                    format=$mgrfmt.;
   define department / order
                order=internal
                format=$deptfmt.;
   define sales / format=dollar7.2;


   break after manager / skip;

   where sector='nw' or sector='sw';

   title 'Sales for the Western Sectors';
run;
```

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 report options. The NOWD option runs PROC REPORT without the REPORT window and sends its output to the open output destinations. HEADLINE underlines all column headings and the spaces between them at the top of each panel of the report. FORMCHAR= sets the value of the second formatting character (the one that HEADLINE uses) to the tilde (~). Therefore, the tilde underlines the column headings in the output. HEADSKIP writes a blank line beneath the underlining that HEADLINE writes. LS= sets the line size for the report to 64, and PS= sets the page size to 18. PANELS= creates a multipanel report. Specifying PANELS=99 ensures that PROC REPORT fits as many panels as possible on one page. PSPACE=6 places six spaces between panels.

```
proc report data=grocery nowd headline
            formchar(2)='~'
            panels=99 pspace=6
            ls=64 ps=18;
```

Specify the report columns. The report contains a column for Manager, Department, and Sales.

```
   column manager department sales;
```

Define the sort order and analysis columns. 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, within each value of Manager, 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.

```
   define manager / order
                    order=formatted
                    format=$mgrfmt.;
   define department / order
                order=internal
                format=$deptfmt.;
   define sales / format=dollar7.2;
```

Produce a report summary. This BREAK statement produces a default summary after the last row for each manager. Because SKIP is the only option in the BREAK statement, each break consists of only a blank line.

```
   break after manager / skip;
```

Select the observations to process. The WHERE statement selects for the report only the observations for stores in the northwest or southwest sector.

```
   where sector='nw' or sector='sw';
```

Specify the title.

```
   title 'Sales for the Western Sectors';
run;
```

Output

Sales for the Western Sectors

Manager	Department	Sales
Adams	Paper	$40.00
	Canned	$225.00
	Meat/Dairy	$350.00
	Produce	$80.00
Brown	Paper	$45.00
	Canned	$230.00
	Meat/Dairy	$250.00
	Produce	$73.00
Pelfrey	Paper	$45.00
	Canned	$420.00
	Meat/Dairy	$205.00
	Produce	$76.00
Reveiz	Paper	$60.00
	Canned	$420.00
	Meat/Dairy	$600.00
	Produce	$30.00
Taylor	Paper	$53.00
	Canned	$120.00
	Meat/Dairy	$130.00
	Produce	$50.00

Example 9: Writing a Customized Summary on Each Page

Features: BREAK statement options
- OL
- PAGE
- SUMMARIZE

COMPUTE statement arguments
- with a computed variable as *report-item*
- BEFORE *break-variable*
- AFTER *break-variable* with conditional logic
- BEFORE _PAGE_

DEFINE statement options
NOPRINT
LINE statement
pointer controls
quoted text
repeating a character string
variable values and formats

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);

proc report data=grocery nowd
            headline headskip;

   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_ / left;
      line sector $sctrfmt. ' Sector';
      line 'Store managed by ' manager $mgrfmt.;
      line ' ';
      line ' ';
```

```
      line ' ';
   endcomp;

   break after manager / ol summarize 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 ' ';
      line text $35.;
   endcomp;
   run;
```

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 report options. The NOWD option runs PROC REPORT without the REPORT window and sends its output to the open output destinations. NOHEADER in the PROC REPORT statement suppresses the default column headings.

```
proc report data=grocery nowd
            headline headskip;
```

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 LEFT option left-justifies the text in the LINE statements. Each LINE statement writes the text in quotation marks just as it appears in the statement. The first two LINE statements write a variable value with the format specified immediately after the variable's name.

```
 compute before _page_ / left;
   line sector $sctrfmt. ' Sector';
   line 'Store managed by ' manager $mgrfmt.;
   line ' ';
   line ' ';
   line ' ';
endcomp;
```

Produce a report summary. This BREAK statement creates a default summary after the last row for each manager. OL writes a row of hyphens above the summary line. 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.

```
break after manager / ol summarize 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 will be long enough to hold the longest version.

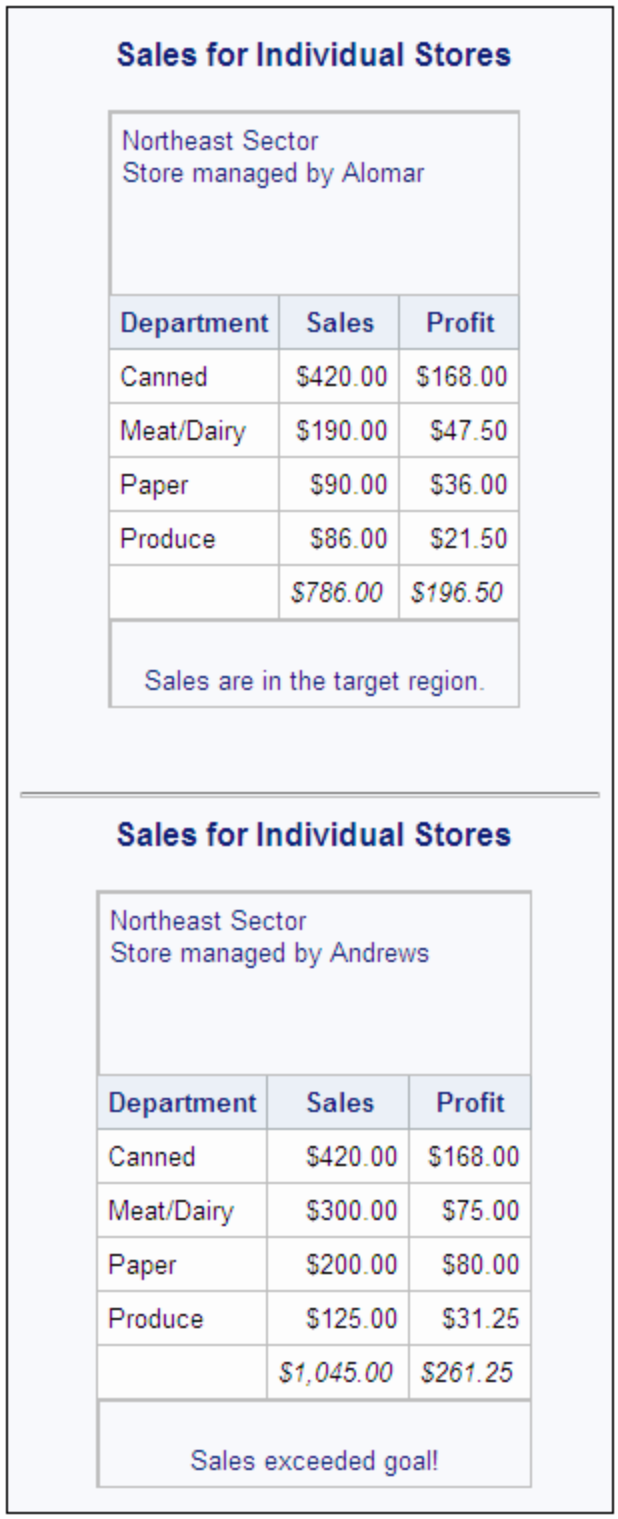
```
   length text $ 35;
```

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 ' ';
      line text $35.;
   endcomp;
   run;
```

Output



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*

Sales are in the target region.

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*

Sales exceeded goal!

Example 10: Calculating Percentages

Features: COLUMN statement arguments
PCTSUM

SUM
spanning headings

COMPUTE statement options
CHAR
LENGTH=

DEFINE statement options
COMPUTED
FLOW
WIDTH=

RBREAK statement options
OL
SUMMARIZE

Other features: 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. This heading looks like a title, but it differs from a title because it would be stored in a report definition. You must submit a null TITLE statement whenever you use the report definition, or the report will contain both a title and the spanning heading.

The report includes a computed character variable, COMMENT, that flags stores with an unusually high percentage of sales. The text of COMMENT wraps across multiple rows. It makes sense to compute COMMENT only for individual stores. Therefore, the compute block that does the calculation includes conditional code that prevents PROC REPORT from calculating COMMENT on the summary line.

Program

```
libname proclib
'SAS-library';

options fmtsearch=(proclib);

proc report data=grocery nowd headline;
   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. width=8;
   define comment / computed width=20 '' flow;

   compute comment / char length=40;

      if sales.pctsum gt .15 and _break_ = ' '
      then comment='Sales substantially above expectations.';
      else comment=' ';
   endcomp;

   rbreak after / ol summarize;
run;
```

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 format search library. The SAS system option FMTSEARCH= adds the SAS library PROCLIB to the search path that is used to locate formats.

```
proc report data=grocery nowd headline;
   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 that 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, a format, and a column width of 8. The PERCENT. format presents the value of Pctsum as a percentage rather than a decimal. The DEFINE statement for COMMENT defines it as a computed variable and assigns it a column width of 20 and a blank column heading. The FLOW option wraps the text for COMMENT onto multiple lines if it exceeds the column width.

```
define pctsum / 'Percent of Sales' format=percent6. width=8;
define comment / computed width=20 '' flow;
```

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 exceeded 15% of the sales for all stores, this 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. OL writes a row of hyphens above the summary line. SUMMARIZE writes the values of Sales.sum and Sales.pctsum in the summary line.

```
   rbreak after / ol summarize;
run;
```

Output

Individual Store Sales as a Percent of All Sales				
Sector	Manager	Total Sales	Percent of Sales	
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 11: How PROC REPORT Handles Missing Values

Features: PROC REPORT statement options
MISSING
COLUMN statement
with the N statistic

Other features: 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 with 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
;

proc report data=grocmiss nowd headline;

   column sector manager N sales;

   define sector / group format=$sctrfmt.;
   define manager / group format=$mgrfmt.;
   define sales / format=dollar9.2;

   rbreak after / dol summarize;

   title 'Summary Report for All Sectors and Managers';
run;
```

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 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
;
```

Specify the report options. The NOWD option runs PROC REPORT without the REPORT window and sends its output to the open output destinations. HEADLINE underlines all column headings and the spaces between them.

```
proc report data=grocmiss nowd headline;
```

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. DOL writes a row of equal signs above the summary line. SUMMARIZE writes the values of N and Sales.sum in the summary line.

```
rbreak after / dol summarize;
```

Specify the title.

```
title 'Summary Report for All Sectors and Managers';
run;
```

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
		31	*$5,443.00*

Program with Data Set with Missing Values

```
proc report data=grocmiss nowd headline missing;
   column sector manager N sales;
   define sector / group format=$sctrfmt.;
   define manager / group format=$mgrfmt.;
   define sales / format=dollar9.2;
   rbreak after / dol summarize;
run;
```

Program Description

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 nowd headline missing;
   column sector manager N sales;
   define sector / group format=$sctrfmt.;
   define manager / group format=$mgrfmt.;
   define sales / format=dollar9.2;
   rbreak after / dol summarize;
run;
```

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 12: Creating and Processing an Output Data Set

Features: PROC REPORT statement options
- BOX
- OUT=

DEFINE statement options
- ANALYSIS
- GROUP
- NOPRINT
- SUM

Other features: Data set options
- WHERE=

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.

The first PROC REPORT step creates a report (which it does not display) 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. This report uses line-drawing characters to separate the rows and columns.

Program to Create Output Data Set

```
libname proclib
'SAS-library';

options fmtsearch=(proclib);

proc report data=grocery nowd
            out=temp( where=(sales gt 1000) );
   column manager sales;

   define manager / group noprint;
   define sales / analysis sum noprint;
run;
```

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 report options and columns. The NOWD option runs PROC REPORT without the REPORT window and sends its output to the open output destinations. OUT= creates the output data set TEMP. The output data set contains a variable for each column in the report (Manager and Sales) 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.

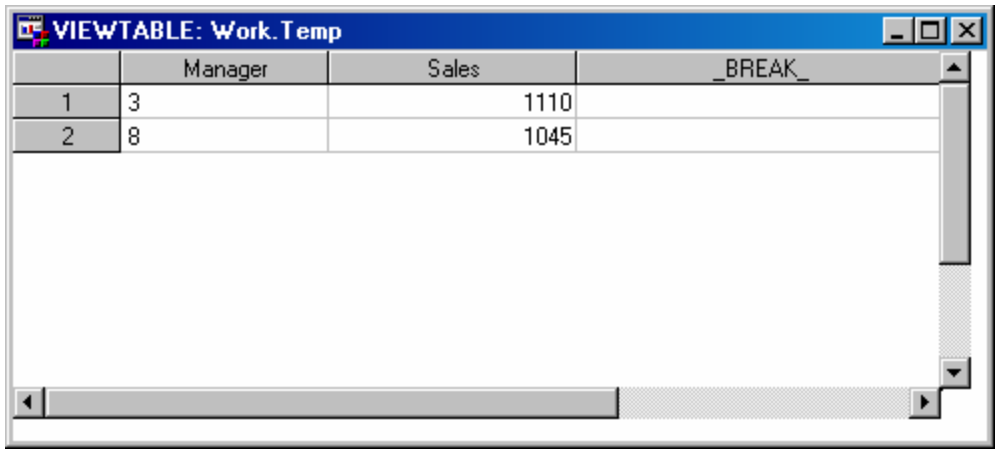
```
proc report data=grocery nowd
            out=temp( where=(sales gt 1000) );
   column manager sales;
```

Define the group and analysis variables. Because the definitions of all report items in this report include the NOPRINT option, PROC REPORT does not print a report. However, the PROC REPORT step does execute and create an output data set.

```
   define manager / group noprint;
   define sales / analysis sum noprint;
run;
```

Output Showing the Output Data Set

The following output is the output data set that PROC REPORT creates. It is used as the input set in the second PROC REPORT step.



Program That Uses the Output Data Set

```
proc report data=temp box nowd;
   column manager sales;
   define manager / group format=$mgrfmt.;
   define sales / analysis sum format=dollar11.2;
   title 'Managers with Daily Sales';
   title2 'of over';
   title3 'One Thousand Dollars';
run;
```

Program Description

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 BOX option draws an outline around the output, separates the column headings from the body of the report, and separates rows and columns of data. The TITLE statements specify a title for the report.

```
proc report data=temp box nowd;
   column manager sales;
   define manager / group format=$mgrfmt.;
   define sales / analysis sum format=dollar11.2;
   title 'Managers with Daily Sales';
   title2 'of over';
   title3 'One Thousand Dollars';
run;
```

Report Based on the Output Data Set

Managers with Daily Sales of over One Thousand Dollars

Manager	Sales
Andrews	$1,045.00
Reveiz	$1,110.00

Example 13: Storing Computed Variables as Part of a Data Set

Features: PROC REPORT statement options
OUT=
COMPUTE statement
with a computed variable as *report-item*
DEFINE statement options
COMPUTED

Other features: CHART procedure

Data set: GROCERY

Format: $SCTRFMT

Details

The report in this example does the following:

- creates a computed variable
- stores it in an output data set
- uses that data set to create a chart based on the computed variable

Program That Creates the Output Data Set

```
libname proclib
'SAS-library';

options fmtsearch=(proclib);

title;

proc report data=grocery nowd out=profit;

   column sector manager department sales Profit;

   define profit / computed;

      /* Compute values for Profit. */
   compute profit;
```

```
      if department='np1' or department='np2' then profit=0.4*sales.sum;
      else profit=0.25*sales.sum;
   endcomp;
run;
```

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);
```

Delete any existing titles.

```
title;
```

Specify the report options. The NOWD option runs PROC REPORT without the REPORT window and sends its output to the open output destinations. OUT= creates the output data set PROFIT.

```
proc report data=grocery nowd out=profit;
```

Specify the report columns. The report contains columns for Manager, Department, Sales, and Profit, which is not in the input data set. Because the purpose of this report is to generate an output data set to use in another procedure, the report layout simply uses the default usage for all the data set variables to list all the observations. DEFINE statements for the data set variables are unnecessary.

```
   column sector manager department sales Profit;
```

Define the computed column. The COMPUTED option tells PROC REPORT that Profit is defined in a compute block somewhere in the PROC REPORT step.

```
   define profit / computed;
```

Calculate the computed column. 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 values for Profit. */
   compute profit;
      if department='np1' or department='np2' then profit=0.4*sales.sum;
      else profit=0.25*sales.sum;
   endcomp;
run;
```

The Output Data Set

The following output is the output data set that is created by PROC REPORT. It is used as input for PROC CHART.

Sector	Manager	Department	Sales	Profit
se	1	np1	50	20
se	1	p1	100	25
se	1	np2	120	48
se	1	p2	80	20
se	2	np1	40	16
se	2	p1	300	75
se	2	np2	220	88
se	2	p2	70	17.5
nw	3	np1	60	24
nw	3	p1	600	150
nw	3	np2	420	168
nw	3	p2	30	7.5
nw	4	np1	45	18
nw	4	p1	250	62.5
nw	4	np2	230	92
nw	4	p2	73	18.25
nw	9	np1	45	18
nw	9	p1	205	51.25
nw	9	np2	420	168
nw	9	p2	76	19
sw	5	np1	53	21.2
sw	5	p1	130	32.5
sw	5	np2	120	48
sw	5	p2	50	12.5
sw	6	np1	40	16

Program That Uses the Output Data Set

```
options fmtsearch=(proclib);
```

```
proc chart data=profit;
   block sector / sumvar=profit;
   format sector $sctrfmt.;
   format profit dollar7.2;
   title 'Sum of Profit by Sector';
run;
```

Program Description

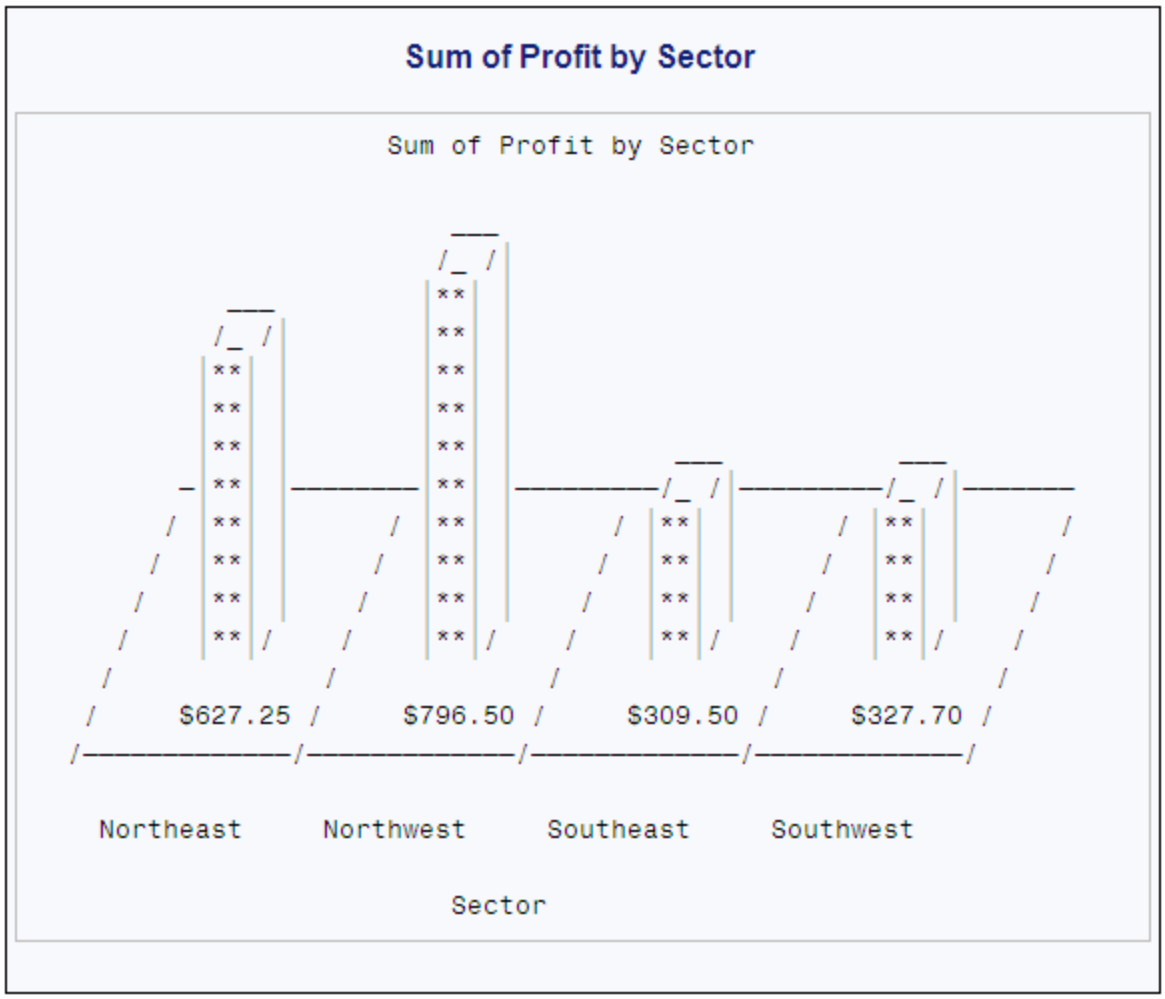
Specify the format search library. The SAS system option FMTSEARCH= adds the SAS library PROCLIB to the search path that is used to locate formats.

```
options fmtsearch=(proclib);
```

Chart the data in the output data set. PROC CHART uses the output data set from the previous PROC REPORT step to chart the sum of Profit for each sector.

```
proc chart data=profit;
   block sector / sumvar=profit;
   format sector $sctrfmt.;
   format profit dollar7.2;
   title 'Sum of Profit by Sector';
run;
```

Output from Processing the Output Data Set



Example 14: Using a Format to Create Groups

Features: DEFINE statement options
GROUP

Other features: FORMAT procedure

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;

proc report data=grocery nowd
     headline
     headskip;

   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 width=13;

   compute after;
      line ' ';
      line 'Total sales for these stores were: '
           sales.sum dollar9.2;
   endcomp;

title 'Sales Summary for All Stores';
run;
```

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 report options. The NOWD option runs the REPORT procedure without the REPORT window and sends its output to the open output destinations. HEADLINE underlines all column headings and the spaces between them at the top of each page of the report. HEADSKIP writes a blank line beneath the underlining that HEADLINE writes.

```
proc report data=grocery nowd
     headline
     headskip;
```

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. WIDTH= 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 width=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. An ENDCOMP statement must end the compute block.

```
compute after;
   line ' ';
   line 'Total sales for these stores were: '
         sales.sum dollar9.2;
endcomp;
```

Specify the title.

```
title 'Sales Summary for All Stores';
run;
```

Output

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

Total sales for these stores were: $6,313.00

Example 15: Specifying Style Elements for ODS Output in the PROC REPORT Statement

Features: STYLE= option in the PROC REPORT statement

Other features: 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.

Program

```
libname proclib
'SAS-library';

options fmtsearch=(proclib);
```

```
ods pdf file='external-PDF-file';
ods rtf file='external-RTF-file';

proc report data=grocery nowd headline headskip

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=$mgrfmt.
                    'Manager';
   define department / order
                       order=internal
                       format=$deptfmt.
                       'Department';

   break after manager / summarize;

   compute after manager;
      line 'Subtotal for ' manager $mgrfmt. 'is '
           sales.sum dollar7.2 '.';
   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;
```

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 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. The NOWD option runs PROC REPORT without the REPORT window. 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 nowd headline headskip
```

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 CELLSPACING=, BORDERWIDTH=, and BORDERCOLOR=.

```
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 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 sort order variables. In this report Manager and Department are order variables. 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. For Manager, ORDER= specifies that values of Manager are arranged according to their formatted values; similarly, for Department, ORDER= specifies that values of Department are arranged according to their internal values. FORMAT= specifies the format to use for each variable. Text in quotation marks specifies the column headings.

```
define manager / order
                 order=formatted
                 format=$mgrfmt.
                 'Manager';
define department / order
                    order=internal
                    format=$deptfmt.
                    'Department';
```

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 after each value of Manager. 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.

```
compute after manager;
   line 'Subtotal for ' manager $mgrfmt. 'is '
        sales.sum dollar7.2 '.';
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';
```

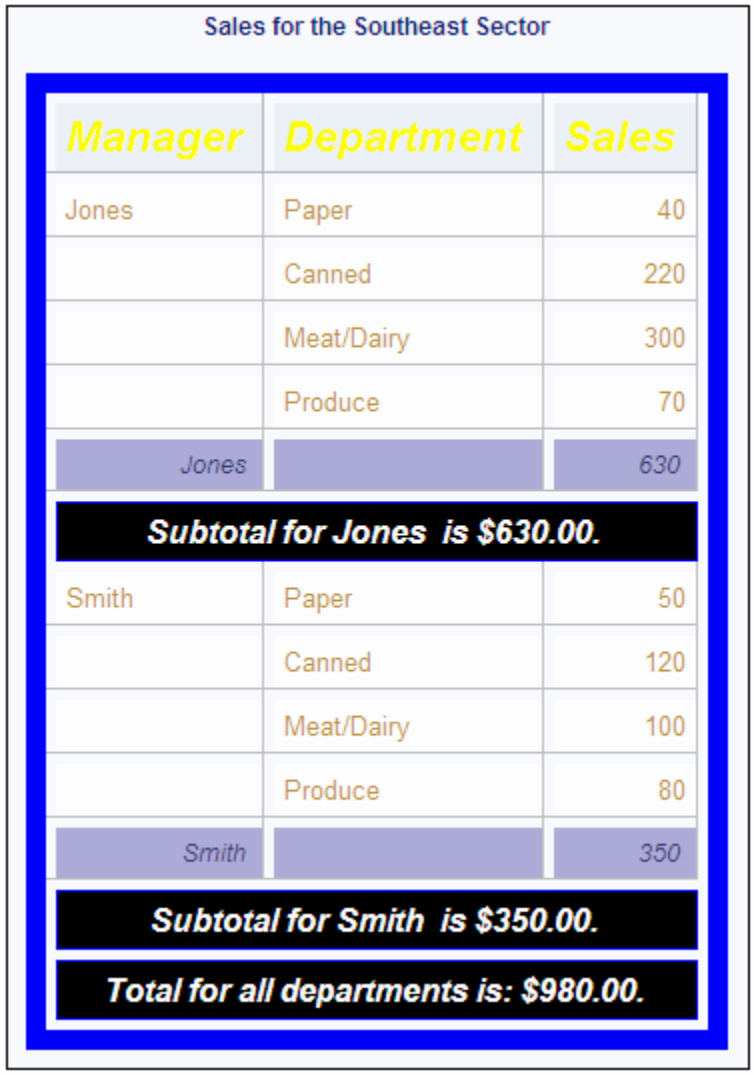
Specify the title.

```
    title 'Sales for the Southeast Sector';
run;
```

Close the ODS destinations.

```
ods pdf close;
ods rtf close;
```

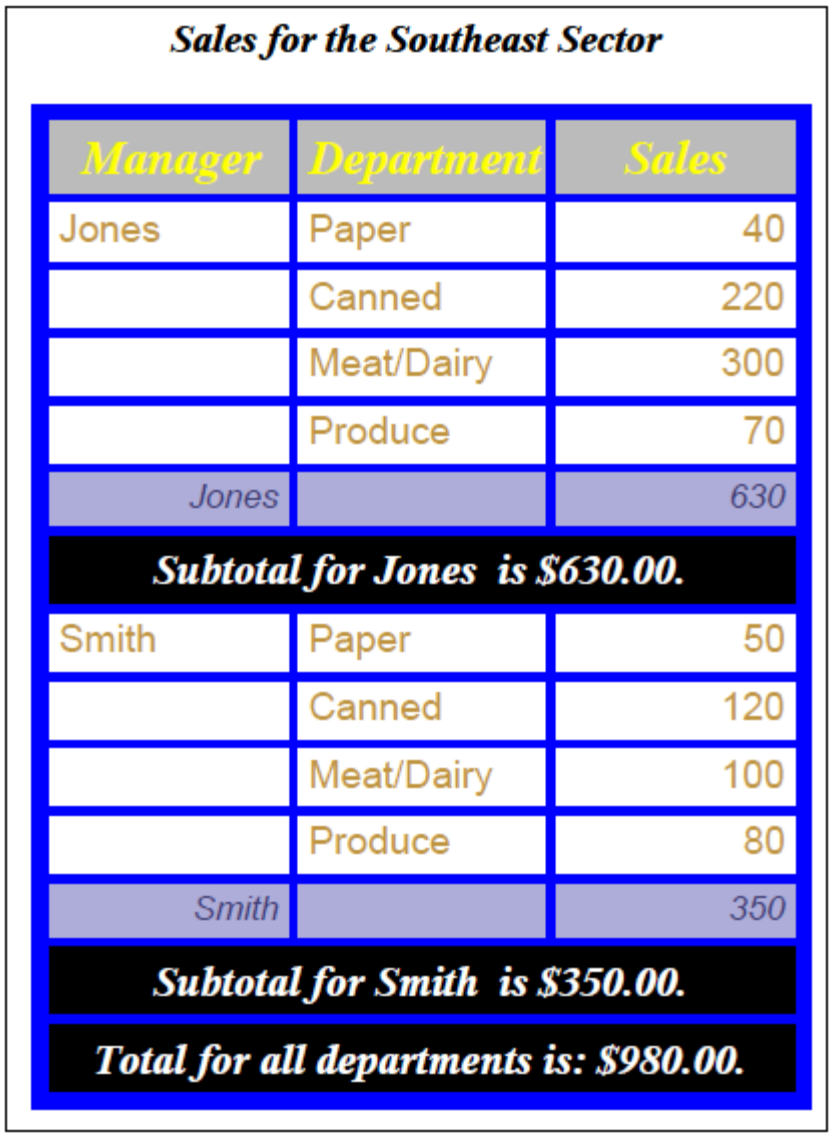
HTML Output



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.		

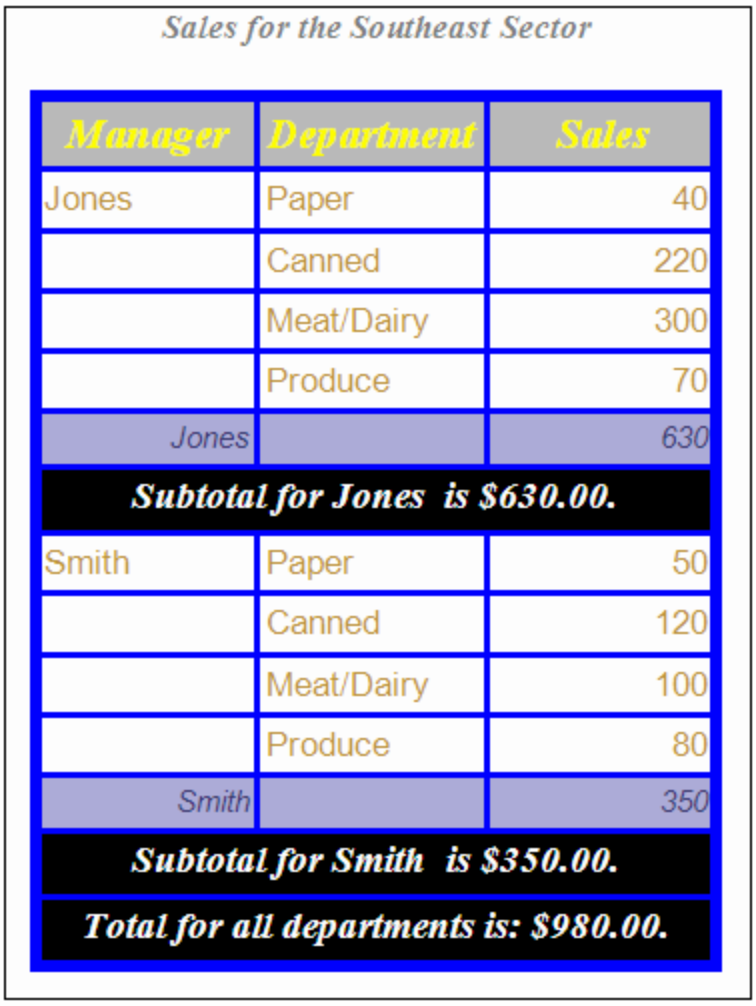
PDF Output



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.		

RTF 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.		

Example 16: Specifying Style Elements for ODS Output in Multiple Statements

Features: STYLE= option in
PROC REPORT statement
CALL DEFINE statement
COMPUTE statement
DEFINE statement

Other features: 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.

Program

```
libname proclib
'SAS-library';

options fmtsearch=(proclib);

ods pdf file='external-PDF-file';
ods rtf file='external-RTF-file';

proc report data=grocery nowd headline headskip

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;
```

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 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. The NOWD option runs PROC REPORT without the REPORT window. 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 nowd headline headskip
```

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 Manager. The other style attributes for the column heading will match the ones that were established for the HEADER location in the PROC REPORT statement.

```
                 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 other style attributes for the cells will 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';
```

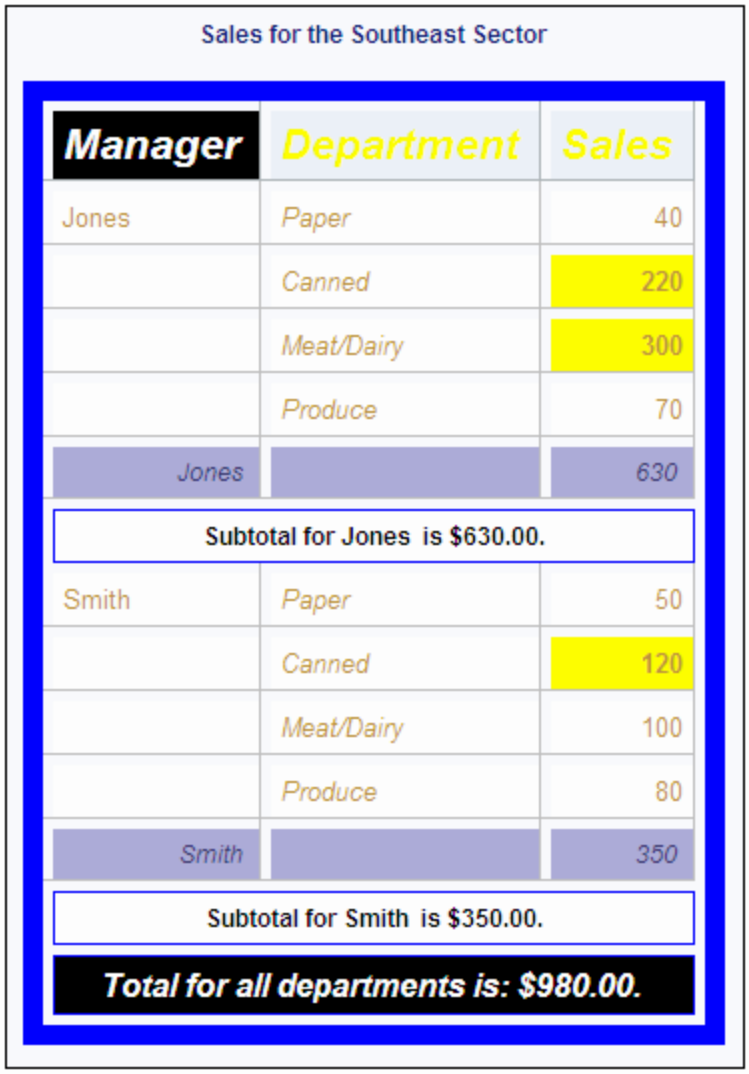
Specify the title.

```
title 'Sales for the Southeast Sector';
run;
```

Close the ODS destinations.

```
ods pdf close;
ods rtf close;
```

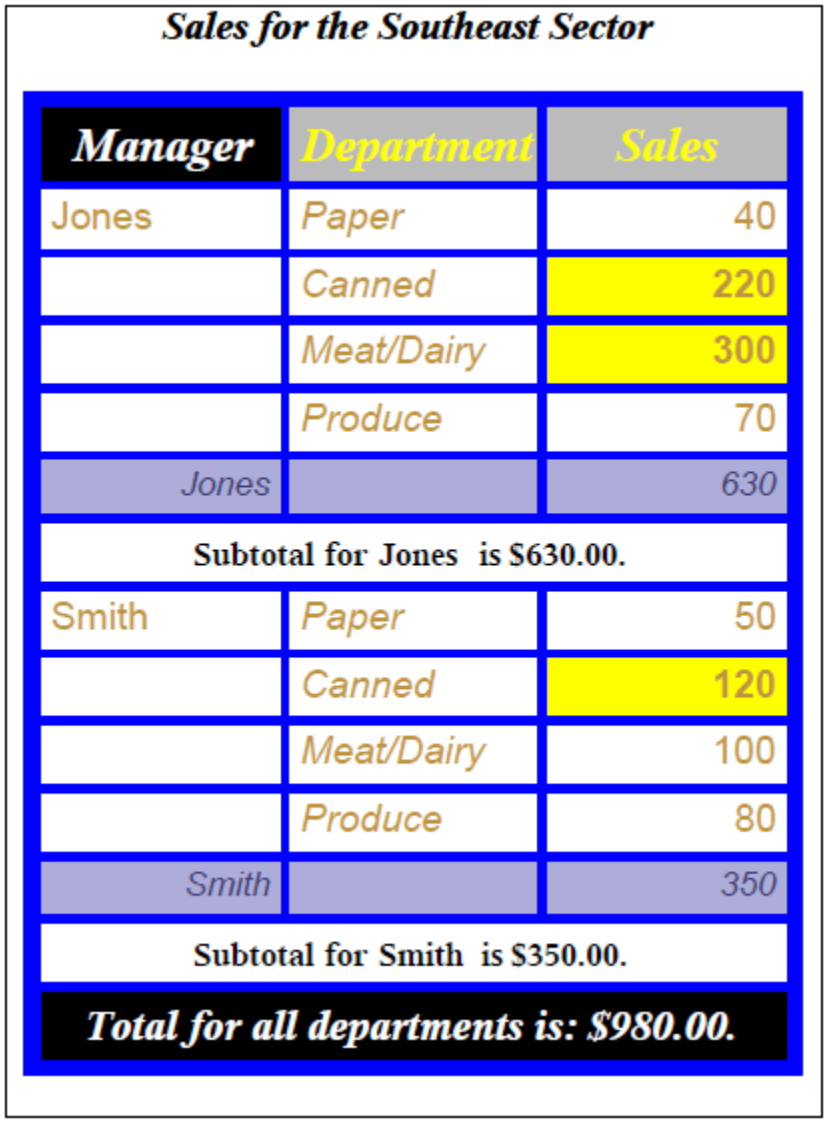
HTML Output



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.		

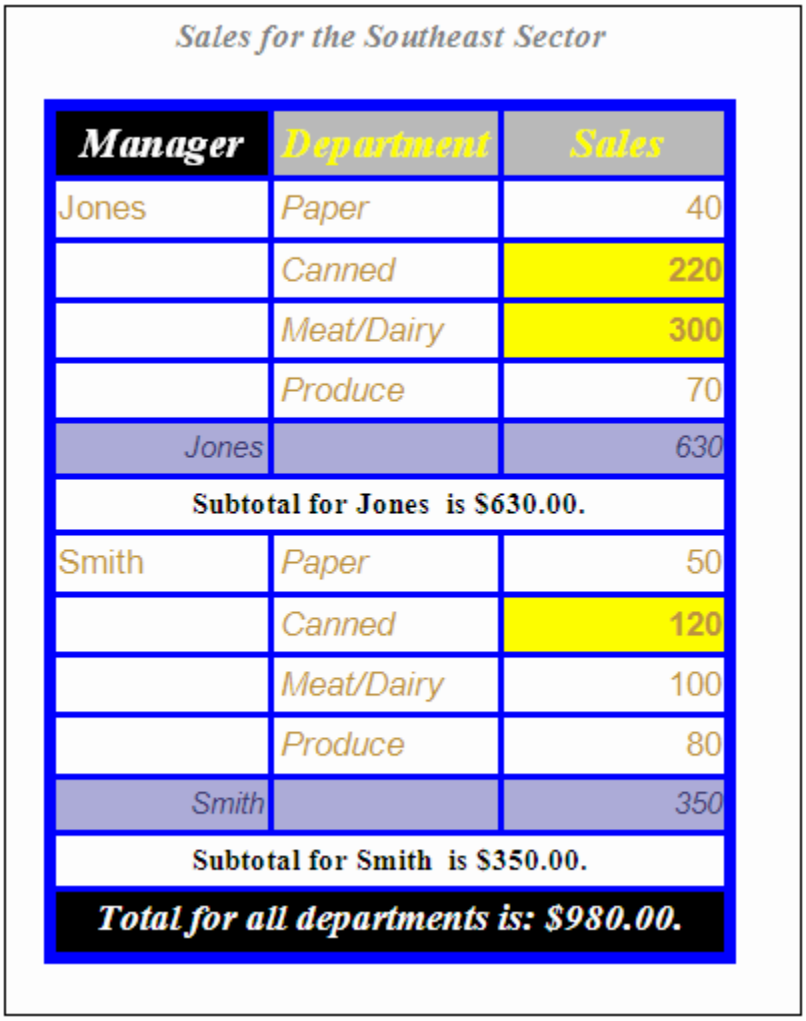
PDF Output



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.		

RTF 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.		

Example 17: Using Multilabel Formats

Features: PROC REPORT statement options
- NOWD

COLUMN statement

DEFINE statement options
- FORMAT=
- GROUP
- MLF
- MEAN

Other features: FORMAT procedure option
- multilabel

ODS HTML 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

Program

```
proc format;
  value agelfmt (multilabel)
     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 nowd;

   col age ('Mean' height weight);

   define age / group mlf format=agelfmt. 'Age Group';
   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.

```
proc format;
  value agelfmt (multilabel)
     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 HTML output filename.

```
ods html file="example.html";
```

Specify a title.

```
title "GROUP Variable with MLF Option";
```

Specify the report options. The NOWD option runs the REPORT procedure without the REPORT window and sends its output to the open output destination.

```
proc report data=sashelp.class nowd;
```

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 group variables. AGE is a group variable. The AGE variable uses the MLF option to activate multilabel format processing. FORMAT= specifies that the multilabel format, AGE1FMT. will be used. 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.

```
   define age / group mlf format=agelfmt. 'Age Group';
   define height / mean format=6.2 'Height (in.)';
   define weight / mean format=6.2 'Weight (lbs.)';
run;
```

Close the ODS HTML output.

```
ods html close;
```

Output

GROUP variable with MLF option

	Mean	
Age Group	Height (in.)	Weight (lbs.)
11	54.40	67.75
11 or 12	58.00	86.79
12	59.44	94.40
13	61.43	88.67
13 and below	59.03	87.35
13 or 14	63.41	96.21
14	64.90	101.88
14 and above	66.01	114.11
15	65.63	117.38
15 or 16	66.90	123.90
16	72.00	150.00

Example 18: Using the WIDTH= and CELLWIDTH= Style Attributes with PROC REPORT

Features: PROC REPORT statement options
- NOWD
- STYLE= using WIDTH= attribute

COLUMN statement

DEFINE statement options
- STYLE= option with CELLWIDTH= attribute
- STYLE= option with WIDTH= attribute

Details

This example uses PROC REPORT to create a table using ODS style attributes that does the following:

- sets the cell width of the total report
- defines the cell width for that column in the ODS output

Refer to “Style Attributes Tables ” in Chapter 13 of *SAS Output Delivery System: User's Guide* for details.

Note: The DEFINE statement WIDTH= option changes the width only for tables output in traditional monospace output.

Program

```
proc report nowd data=sashelp.class style(report)={width=50%};
   col name age sex;
   define age / style(column)=[cellwidth=1in];
   define sex / style(column)=[width=10%];
title "Using the WIDTH= and CELLWIDTH= Styles with PROC REPORT";
run;
```

Program Description

Specify the cell width for all the columns in the ODS output. The NOWD option runs the REPORT procedure without the REPORT window and sends its output to the open output destination. ODS HTML is the output destination used as the default in this example. The STYLE= option used with the WIDTH= attribute sets the cell width for all of the columns in the ODS output.

```
proc report nowd data=sashelp.class style(report)={width=50%};
```

Specify the Columns to be used.

```
   col name age sex;
```

Define a column width using the CELLWIDTH= style attribute. Define the dimensions of the AGE column using the STYLE= option. The style attribute CELLWIDTH= is used to define the column size as 1 inch.

```
   define age / style(column)=[cellwidth=1in];
```

Define a column width using the WIDTH= style attribute. Define the dimensions of the SEX column using the STYLE= option. The style attribute WIDTH= is used to define the column size as a percentage of the table size.

```
   define sex / style(column)=[width=10%];
```

Specify a table title. Provide a table name and run the SAS program.

```
title "Using the WIDTH= and CELLWIDTH= Styles with PROC REPORT";
run;
```

Output

Using the WIDTH= and CELLWIDTH= Style Attributes 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

Chapter 47

REPORT Procedure Windows

Overview of REPORT Procedure Windows

The interactive report window environment in PROC REPORT provides essentially the same functionality as the statements, with one major exception: you cannot use the Output Delivery System from the interactive report window environment.

Dictionary

BREAK

Controls PROC REPORT's actions at a change in the value of a group or order variable or at the top or bottom of a report.

Details

Path

Edit ⇨ Summarize information

After you select **Summarize information**, PROC REPORT offers you four choices for the location of the break:

- **Before Item**
- **After Item**
- **At the top**
- **At the bottom**

After you select a location, the BREAK window appears.

Note: To create a break before or after detail lines (when the value of a group or order variable changes), you must select a variable before you open the BREAK window.

Description

Note: For information about changing the formatting characters that are used by the line drawing options in this window, see the discussion of "FORMCHAR <(position(s))>='formatting-character(s)' " on page 1250

Options

Overline summary
uses the second formatting character to overline each value

- that appears in the summary line
- that would appear in the summary line if you specified the SUMMARIZE option

Default: hyphen (-)

Interaction: If you specify options to overline and to double overline, then PROC REPORT overlines.

Double overline summary
uses the 13th formatting character to overline each value

- that appears in the summary line
- that would appear in the summary line if you specified the SUMMARIZE option

Default: equal sign (=)

Interaction: If you specify options to overline and to double overline, then PROC REPORT overlines.

Underline summary
uses the second formatting character to underline each value

- that appears in the summary line
- that would appear in the summary line if you specified the SUMMARIZE option

Default: hyphen (-)

Interaction: If you specify options to underline and to double underline, then PROC REPORT underlines.

Double underline summary
uses the 13th formatting character to underline each value

- that appears in the summary line
- that would appear in the summary line if you specified the SUMMARIZE option

Default: equal sign (=)

Interaction: If you specify options to underline and to double underline, then PROC REPORT underlines.

Skip line after break
writes a blank line for the last break line.

This option has no effect if you use it in a break at the end of a report.

Page after break
starts a new page after the last break line. This option has no effect in a break at the end of a report.

Interaction: If you use this option in a break on a variable and you create a break at the end of the report, then the summary for the whole report is on a separate page.

Summarize analysis columns
writes a summary line in each group of break lines. A summary line contains values for

- statistics
- analysis variables
- computed variables

A summary line between sets of observations also contains

- the break variable (which you can suppress with **Suppress break value**)
- other group or order variables to the left of the break variable.

The following table shows how PROC REPORT calculates the value for each type of report item in a summary line created by the BREAK window:

Report Item	Resulting Value
The break variable	The current value of the variable (or a missing value if you select **`suppress break value`**)
A group or order variable to the left of the break variable	The current value of the variable
A group or order variable to the right of the break variable, or a display variable anywhere in the report	Missing*
A statistic	The value of the statistic over all observations in the set
An analysis variable	The 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	The results of the calculations based on the code in the corresponding compute block. (See the "COMPUTE Statement" on page 1273 statement.)

*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).

Suppress break value
suppresses printing of

- the value of the break variable in the summary line
- any underlining and overlining in the break lines in the column containing the break variable

If you select **Suppress break value**, then the value of the break variable is unavailable for use in customized break lines unless you assign it a value in the compute block that is associated with the break.

Color

From the list of colors, select the one to use in the REPORT window for the column heading and the values of the item that you are defining. The default is the color of **Foreground** in the SASCOLOR window. (For more information, see the online Help for the **SASCOLOR** window.)

Note: Not all operating environments and devices support all colors, and in some operating environments and devices, one color might map to another color. For example, if the DEFINITION window displays the word BROWN in yellow characters, then selecting BROWN results in a yellow item.

Buttons

Edit Program
opens the COMPUTE window and enables you to associate a compute block with a location in the report.

OK
applies the information in the BREAK window to the report and closes the window.

Cancel
closes the BREAK window without applying information to the report.

COMPUTE

Attaches a compute block to a report item or to a location in the report. Use the SAS Text Editor commands to manipulate text in this window.

Details

Path

From **Edit Program** in the COMPUTED VAR, DEFINITION, or BREAK window.

Description

For information about the SAS language features that you can use in the COMPUTE window, see "The Contents of Compute Blocks" on page 1232.

COMPUTED VAR

Adds a variable that is not in the input data set to the report.

Details

Path

Select a column. Then select **Edit** ⇨ **Add Item** ⇨ **Computed Column**.

After you select **Computed Column**, PROC REPORT prompts you for the location of the computed column relative to the column that you have selected. After you select a location, the COMPUTED VAR window appears.

Description

Enter the name of the variable at the prompt. If it is a character variable, then select the **Character data** check box and, if you want, enter a value in the **Length** field. The length can be any integer between 1 and 200. If you leave the field blank, then PROC REPORT assigns a length of 8 to the variable.

After you enter the name of the variable, select **Edit Program** to open the COMPUTE window. Use programming statements in the COMPUTE window to define the computed variable. After closing the COMPUTE and COMPUTED VAR windows, open the DEFINITION window to describe how to display the computed variable.

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.

DATA COLUMNS

Lists all variables in the input data set so that you can add one or more data set variables to the report.

Details

Path

Select a report item. Then select **Edit** ⇨ **Add Item** ⇨ **Data Column**.

After you select **Data column**, PROC REPORT prompts you for the location of the computed column relative to the column that you have selected. After you select a location, the DATA COLUMNS window appears.

Description

Select one or more variables to add to the report. When you select the first variable, it moves to the top of the list in the window. If you select multiple variables, then subsequent selections move to the bottom of the list of selected variables. An asterisk (*) identifies each selected variable. The order of selected variables from top to bottom determines their order in the report from left to right.

DATA SELECTION

Loads a data set into the current report definition.

Details

Path

File ⇨ **Open Data Set**

Description

The first list box in the DATA SELECTION window lists all the librefs defined for your SAS session. The second one lists all the SAS data sets in the selected library.

Note: You must use data that is compatible with the current report definition. The data set that you load must contain variables whose names are the same as the variable names in the current report definition.

Buttons

OK
loads the selected data set into the current report definition.

Cancel
closes the DATA SELECTION window without loading new data.

DEFINITION

Displays the characteristics associated with an item in the report and lets you change them.

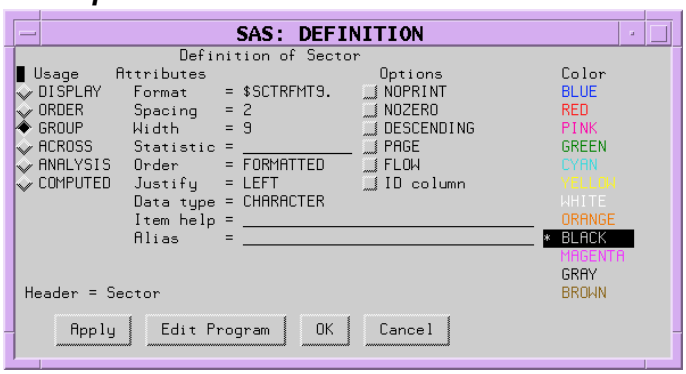
Details

Path

Select a report item. Then select **Edit** ⇨ **Define**.

Note: Alternatively, double-click on the selected item. (Not all operating environments support this method of opening the DEFINITION window.)

Description



Usage

For an explanation of each type of usage, see “Laying Out a Report ” on page 1225.

DISPLAY
defines the selected item as a display variable. DISPLAY is the default for character variables.

ORDER
defines the selected item as an order variable.

GROUP
defines the selected item as a group variable.

ACROSS
defines the selected item as an across variable.

ANALYSIS
defines the selected item as an analysis variable. You must specify a statistic (see the discussion of the Statistic= attribute on page 1383) for an analysis variable. ANALYSIS is the default for numeric variables.

COMPUTED
defines the selected 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. However, computed variables are included in an output data set if you create one.

In the interactive report window environment, you add a computed variable to a report from the COMPUTED VAR window.

Attributes

Format=
assigns a SAS or user-defined format to the item. This format applies to the selected item as PROC REPORT displays it; the format does not alter the format that is 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 DEFINITION window
- the format that is assigned in a FORMAT statement when you start 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 BEST*w*. 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. For numeric variables in the input data set and for computed variables (both numeric and character), the default column width is the value of the COLWIDTH= attribute in the ROPTIONS window.

If you are unsure what format to use, then type a question mark (?) in the format field in the DEFINITION window to access the FORMATS window.

Spacing=
defines the number of blank characters to leave between the column being defined and the column immediately to its left. For each column, the sum of its width and the blank characters between it and the column to its left cannot exceed the line size.

Default: 2

Interactions:
When PROC REPORT's CENTER option is in effect, PROC REPORT ignores spacing that precedes the leftmost variable in the report.

SPACING= in an item definition overrides the value of SPACING= in the PROC REPORT statement or the ROPTIONS window.

Width=
defines the width of the column in which PROC REPORT displays the selected item.

Default: A column width that is just large enough to handle the format. If there is no format, then PROC REPORT uses the value of COLWIDTH=.

Range: 1 to the value of the SAS system option LINESIZE=

Note: When you stack items in the same column in a report, the width of the item that is at the bottom of the stack determines the width of the column.

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 represented by each cell of the report. You cannot use *statistic* in the definition of any other type of variable.

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 **Header** field of the DEFINITION window.

You can use the following values for *statistic*:

Descriptive statistic keywords	
CSS	PCTSUM
CV	RANGE
MAX	STD
MEAN	STDERR
MIN	SUM
N	SUMWGT
NMISS	USS
PCTN	VAR
Quantile statistic keywords	
MEDIAN | P50	Q3 | P75
P1	P90
P5	P95
P10	P99
Q1 | P25	QRANGE
Hypothesis testing keyword	
PRT|PROBT	T

Explanations of the keywords, the formulas that are used to calculate them, and the data requirements are discussed in "SAS Elementary Statistics Procedures" on page 1665.

Default: SUM

Requirement: To compute standard error and the Student's *t*-test you must use the default value of VARDEF=, which is DF.

See: For definitions of these statistics, see "Keywords and Formulas" on page 1666.

Order=
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.

FORMATTED
orders values by their formatted (external) values. By default, the order is ascending.

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.

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 will continue to produce the reports that you expect even if the default changes.

Justify=
You can justify the placement of the column heading and of the values of the item that you are defining within a column in one of three ways:

LEFT
left-justifies the formatted values of the item that you are defining within the column width and left-justifies the column heading over the values. If the format width is the same as the width of the column, then LEFT has no effect on the placement of values.

RIGHT
right-justifies the formatted values of the item that you are defining within the column width and right-justifies the column heading 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.

CENTER
centers the formatted values of the item that you are defining within the column width and centers the column heading over the values. This option has no effect on the setting of the SAS system option CENTER.

When justifying values, PROC REPORT justifies the field width defined by the format of the item within the column. Thus, numbers are always aligned.

Data type=
shows you if the report item is numeric or character. You cannot change this field.

Item Help=
references a HELP or CBT entry that contains help information for the selected item. Use PROC BUILD in SAS/AF software to create a HELP or CBT entry for a report item. All HELP and CBT entries for a report must be in the same catalog, and you must specify that catalog with the HELP= option in the PROC REPORT statement or from the **User Help** fields in the ROPTIONS window.

To access a help entry from the report, select the item and issue the HELP command. PROC REPORT first searches for and displays an entry named *entry-name*.CBT. If no such entry exists, then PROC REPORT searches for *entry-name*.HELP. If neither a CBT nor a HELP entry for the selected item exists, then the opening frame of the help for PROC REPORT is displayed.

Alias=
By entering a name in the **Alias** field, you create an alias for the report item that you are defining. Aliases let you distinguish between different uses of the same report item. When you refer in a compute block to a report item that has an alias, you must use the alias. (See "Example 3: Using Aliases to Obtain Multiple Statistics for the Same Variable" on page 1314.)

Options

NOPRINT
suppresses the display of the item that you are defining. Use this option

- if you do not want to show the item in the report but you need to use the values in it 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 9: Writing a Customized Summary on Each Page" on page 1333.)

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 1233 .)

SHOWALL in the PROC REPORT statement or the ROPTIONS window overrides all occurrences of NOPRINT.

NOZERO
suppresses the display of the item that you are defining 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 1233.)

SHOWALL in the PROC REPORT statement or the ROPTIONS window overrides all occurrences of NOZERO.

DESCENDING
reverses the order in which PROC REPORT displays rows or values of a group, order, or across variable.

PAGE
inserts a page break just before printing the first column containing values of the selected item.

Interaction: PAGE is ignored if you use WRAP in the PROC REPORT statement or in the ROPTIONS window.

FLOW
wraps the value of a character variable in its column. The FLOW option honors the split character. If the text contains no split character, then PROC REPORT tries to split text at a blank.

ID column
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 will fit on one page.

Color

From the list of colors, select the one to use in the REPORT window for the column heading and the values of the item that you are defining. The default is the color of **Foreground** in the SASCOLOR window. (For more information, see the online Help for the SASCOLOR window.)

Note: Not all operating environments and devices support all colors, and in some operating environments and devices, one color might map to another color. For example, if the DEFINITION window displays the word BROWN in yellow characters, then selecting BROWN results in a yellow item.

Buttons

Apply
applies the information in the open window to the report and keeps the window open.

Edit Program
opens the COMPUTE window and enables you to associate a compute block with the variable that you are defining.

OK
applies the information in the DEFINITION window to the report and closes the window.

Cancel
closes the DEFINITION window without applying changes made with **APPLY**.

DISPLAY PAGE

Displays a particular page of the report.

Details

Path

View ⇨ Display Page

Description

You can access the last page of the report by entering a large number for the page number. When you are on the last page of the report, PROC REPORT sends a note to the message line of the REPORT window.

EXPLORE

Lets you experiment with your data.

Restriction: You cannot open the EXPLORE window unless your report contains at least one group or order variable.

Details

Path

Edit ⇨ Explore Data

Description

In the EXPLORE window, you can

- subset the data with list boxes
- suppress the display of a column with the **Remove Column** check box
- change the order of the columns with **Rotate columns**

Note: The results of your manipulations in the EXPLORE window appear in the REPORT window but are not saved in report definitions.

Window Features

list boxes

The EXPLORE window contains three list boxes. These boxes contain the value **All levels** as well as actual values for the first three group or order variables in your report. The values reflect any WHERE clause processing that is in effect. For example, if you use a WHERE clause to subset the data so that it includes only the northeast and northwest sectors, then the only values that appear in the list box for Sector are **All levels**, **Northeast**, and **Northwest**. Selecting **All levels** in this case displays rows of the report for only the northeast and northwest sectors. To see data for all the sectors, you must clear the WHERE clause before you open the EXPLORE window.

Selecting values in the list boxes restricts the display in the REPORT window to the values that you select. If you select incompatible values, then PROC REPORT returns an error.

Remove Column

Above each list box in the EXPLORE window is a check box labeled **Remove Column**. Selecting this check box and applying the change removes the column from the REPORT window. You can easily restore the column by clearing the check box and applying that change.

Buttons

OK

applies the information in the EXPLORE window to the report and closes the window.

Apply

applies the information in the EXPLORE window to the report and keeps the window open.

Rotate columns
changes the order of the variables displayed in the list boxes. Each variable that can move one column to the left does; the leftmost variable moves to the third column.

Cancel
closes the EXPLORE window without applying changes made with **APPLY**.

FORMATS

Displays a list of formats and provides a sample of each one.

Details

Path

From the DEFINE window, type a question mark (?) in the **Format** field and select any of the Buttons except **Cancel**, or press RETURN.

Description

When you select a format in the FORMATS window, a sample of that format appears in the **Sample:** field. Select the format that you want to use for the variable that you are defining.

Buttons

OK
writes the format that you have selected into the **Format** field in the DEFINITION window and closes the FORMATS window. To see the format in the report, select **Apply** in the DEFINITION window.

Cancel
closes the FORMATS window without writing a format into the **Format** field.

LOAD REPORT

Loads a stored report definition.

Details

Path

File ⇨ Open Report

Description

The first list box in the LOAD REPORT window lists all the librefs that are defined for your SAS session. The second list box lists all the catalogs that are in the selected library. The third list box lists descriptions of all the stored report definitions (entry types of REPT) that are in the selected catalog. If there is no description for an entry, then the list box contains the entry's name.

Buttons

OK
loads the current data into the selected report definition.

Cancel
closes the LOAD REPORT window without loading a new report definition.

Note: Issuing the END command in the REPORT window returns you to the previous report definition (with the current data).

MESSAGES

Automatically opens to display notes, warnings, and errors returned by PROC REPORT.

Details

You must close the MESSAGES window by selecting **OK** before you can continue to use PROC REPORT.

PROFILE

Customizes some features of the PROC REPORT environment by creating a report profile.

Details

Path

Tools ⇨ Report Profile

Description

The PROFILE window creates a report profile that

- specifies the SAS library, catalog, and entry that define alternative menus to use in the REPORT and COMPUTE windows. Use PROC PMENU to create catalog entries of type PMENU that define these menus. PMENU entries for both windows must be in the same catalog.
- sets defaults for WINDOWS, PROMPT, and COMMAND. PROC REPORT uses the default option whenever you start the procedure unless you specifically override the option in the PROC REPORT statement.

Specify the catalog that contains the profile to use with the PROFILE= option in the PROC REPORT statement. (See the discussion of "PROFILE=libref.catalog" on page 1256.)

Buttons

OK
stores your profile in a file that is called SASUSER.PROFILE.REPORT.PROFILE.

Note: Use PROC CATALOG or the EXPLORER window to copy the profile to another location.

Cancel
closes the window without storing the profile.

PROMPTER

Prompts you for information as you add items to a report.

Details

Path

Specify the PROMPT option when you start PROC REPORT or select PROMPT from the ROPTIONS window. The PROMPTER window appears the next time you add an item to the report.

Description

The prompter guides you through parts of the windows that are most commonly used to build a report. As the content of the PROMPTER window changes, the title of the window changes to the name of the window that you would use to perform a task if you were not using the prompter. The title change is to help you begin to associate the windows with their functions and to learn what window to use if you later decide to change something.

If you start PROC REPORT with prompting, then the first window gives you a chance to limit the number of observations that are used during prompting. When you exit the prompter, PROC REPORT removes the limit.

Buttons

OK
applies the information in the open window to the report and continues the prompting process.

Note: When you select **OK** from the last prompt window, PROC REPORT removes any limit on the number of observations that it is working with.

Apply
applies the information in the open window to the report and keeps the window open.

Backup
returns you to the previous PROMPTER window.

Exit Prompter
closes the PROMPTER window without applying any more changes to the report. If you have limited the number of observations to use during prompting, then PROC REPORT removes the limit.

REPORT

Is the surface on which the report appears.

Details

Path

Use WINDOWS or PROMPT in the PROC REPORT statement.

Description

You cannot write directly in any part of the REPORT window except column headings. To change other aspects of the report, you select a report item (for example, a column heading) as the target of the next command and issue the command. To select an item, use a mouse or cursor keys to position the cursor over it. Then click the mouse button or press ENTER. To execute a command, make a selection from the menu bar at the top of the REPORT window. PROC REPORT displays the effect of a command immediately unless the DEFER option is on.

Note: Issuing the END command in the REPORT window returns you to the previous report definition with the current data. If there is no previous report definition, then END closes the REPORT window.

Note: In the REPORT window, there is no Save As option from the File menu to save your report to a file. Instead:

1. From the REPORT window, select Save Data Set. In the dialog box, enter a SAS library and filename in which to save this data set.
2. From the program editor window, execute a PROC PRINT.
3. In the File menu, select Save As to save the generated output to a file.

ROPTIONS

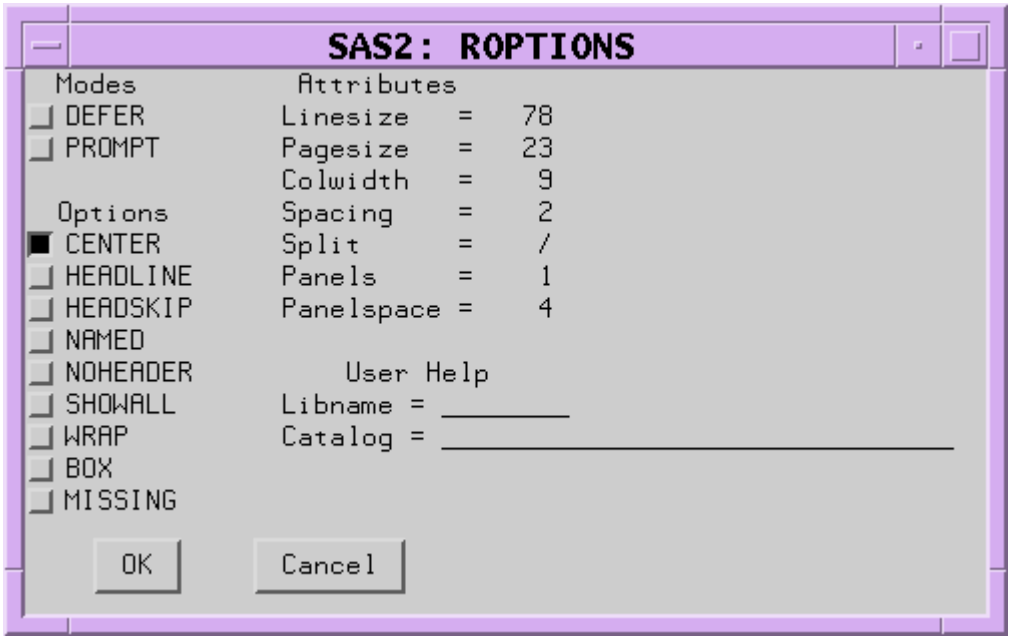
Displays choices that control the layout and display of the entire report and identifies the SAS library and catalog containing CBT or HELP entries for items in the report.

Details

Path

Tools ⇨ Options ⇨ Report

Description



Modes

DEFER

stores the information for changes and makes the changes all at once when you turn DEFER mode off or select **View** ⇨ **Refresh**.

DEFER is particularly useful when you know that you need to make several changes to the report but do not want to see the intermediate reports.

By default, PROC REPORT redisplays the report in the REPORT window each time you redefine the report by adding or deleting an item, by changing information in the DEFINITION window, or by changing information in the BREAK window.

PROMPT

opens the PROMPTER window the next time you add an item to the report.

Options

CENTER

centers the report and summary text (customized break lines). If CENTER is not selected, then the report is left-justified.

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 loaded with REPORT= in the PROC REPORT statement
- the SAS system option CENTER or NOCENTER

When PROC REPORT's CENTER option is in effect, PROC REPORT ignores spacing that precedes the leftmost variable in the report.

HEADLINE

underlines all column headings and the spaces between them at the top of each page of the report.

HEADLINE underlines with the second formatting character. (See the discussion of "FORMCHAR <(position(s))>='formatting-character(s)' " on page 1250.)

Default: hyphen (-)

Tip: In traditional (monospace) SAS output, you can underline column headings without underlining the spaces between them, by using `'--'` as the last line of each column heading instead of using HEADLINE.

HEADSKIP
writes a blank line beneath all column headings (or beneath the underlining that the HEADLINE option writes) at the top of each page of the report.

NAMED
writes *name*= in front of each value in the report, where *name* is the column heading for the value.

Interaction: When you use NAMED, PROC REPORT automatically uses NOHEADER.

Tip: Use NAMED in conjunction with WRAP 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.

NOHEADER
suppresses column headings, including headings that span multiple columns.

Once you suppress the display of column headings in the interactive report window environment, you cannot select any report items.

SHOWALL
overrides the parts of a definition that suppress the display of a column (NOPRINT and NOZERO). You define a report item with a DEFINE statement or in the DEFINITION window.

WRAP
displays one value from each column of the report, on consecutive lines if necessary, before displaying another value from the first column. By default, PROC REPORT displays values for only as many columns as it can fit on one page. It fills a page with values for these columns before starting to display values for the remaining columns on the next page.

Interaction: When WRAP is in effect, PROC REPORT ignores PAGE in any item definitions.

Tip: Typically, you use WRAP in conjunction with NAMED to avoid wrapping column headings.

BOX
uses formatting characters to add line-drawing characters to the report. These characters

- surround each page of the report
- separate column headings from the body of the report
- separate rows and columns from each other

Interaction: You cannot use BOX if you use WRAP in the PROC REPORT statement or ROPTIONS window or if you use FLOW in any item's definition.

See: For information about formatting characters, see the discussion of "FORMCHAR <(position(s))>='formatting-character(s)' " on page 1250.

MISSING
considers missing values as valid values for group, order, or across 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 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 one or more group, order, or across variables in the report.

Attributes

LINESIZE=
specifies the line size for a report. PROC REPORT honors the first of these line-size specifications that it finds:

- LS= in the PROC REPORT statement or LINESIZE= in the ROPTIONS window
- the LS= setting stored in the report definition loaded with REPORT= in the PROC REPORT statement
- the SAS system option LINESIZE=

Range: 64-256 (integer)

Tip: If the line size is greater than the width of the REPORT window, then use SAS interactive report window environment commands RIGHT and LEFT to display portions of the report that are not currently in the display.

PAGESIZE=
specifies the page size for a report. PROC REPORT honors the first of these page size specifications that it finds:

- PS= in the PROC REPORT statement or PAGESIZE= in the ROPTIONS window
- the PS= setting stored in the report definition loaded with REPORT= in the PROC REPORT statement
- the SAS system option PAGESIZE=

Range: 15-32,767 (integer)

COLWIDTH=
specifies the default number of characters for columns containing computed variables or numeric data set variables.

When setting the width for a column, PROC REPORT first looks at WIDTH= in the definition for that column. If WIDTH= is not present, then PROC REPORT uses a column width large enough to accommodate the format for the item. (For information about formats, see the discussion of "Format=" on page 1382 .) If no format is associated with the item, then the column width depends on variable type:

Variable	Resulting Column Width
Character variable in the input data set	Length of the variable
Numeric variable in the input data set	Value of the COLWIDTH= option
Computed variable (numeric or character)	Value of the COLWIDTH= option

Default: 9

Range: 1 to the line size

SPACING=*space-between-columns*
specifies the number of blank characters between columns. For each column, the sum of its width and the blank characters between it and the column to its left cannot exceed the line size.

Default: 2

Interactions:
PROC REPORT separates all columns in the report by the number of blank characters specified by SPACING= in the PROC REPORT statement or the ROPTIONS window unless you use SPACING= in the definition of a particular item to change the spacing to the left of that item.

When CENTER is in effect, PROC REPORT ignores spacing that precedes the leftmost variable in the report.

SPLIT='*character*'
specifies the split character. PROC REPORT breaks a column heading when it reaches that character and continues the heading on the next line. The split character itself is not part of the column heading although each occurrence of the split character counts toward the 40-character maximum for a label.

Default: slash (/)

Interaction: The FLOW option in the DEFINE statement honors the split character.

Tip: If you are typing over a heading (rather than entering one from the PROMPTER or DEFINITION window), then you do not see the effect of the split character until you refresh the screen by adding or deleting an item, by changing the contents of a DEFINITIONor a BREAKwindow, or by selecting**View** ⇨ **Refresh**.

PANELS=*number-of-panels*
specifies the number of panels on each page of the report. If the width of a report is less than half of the line size, then you can display the data in multiple sets of columns so that rows that would otherwise appear on multiple pages appear on the same page. Each set of columns is a *panel*. A familiar example of this type of report is a telephone book, which contains multiple panels of names and telephone numbers on a single page.

When PROC REPORT writes a multipanel report, it fills one panel before beginning the next.

The number of panels that fits on a page depends on the

- width of the panel
- space between panels
- line size

Default: 1

Tip: If *number-of-panels* is larger than the number of panels that can fit on the page, then PROC REPORT creates as many panels as it can. Let PROC REPORT put your data in the maximum number of panels that can fit on the page by specifying a large number of panels (for example, 99).

See: For information about specifying the space between panels see the discussion of PSPACE=. For information about setting the line size, see the discussion of "LINESIZE=" on page 1394.

PSPACE=*space-between-panels*
specifies the number of blank characters between panels. PROC REPORT separates all panels in the report by the same number of blank characters. For each panel, the

sum of its width and the number of blank characters separating it from the panel to its left cannot exceed the line size.

Default: 4

User Help
identifies the library and catalog containing user-defined help for the report. This help can be in CBT or HELP catalog entries. You can write a CBT or HELP entry for each item in the report with the BUILD procedure in SAS/AF software. You must store all such entries for a report in the same catalog.

Specify the entry name for help for a particular report item in the DEFINITION window for that report item or in a DEFINE statement.

SAVE DATA SET

Lets you specify an output data set in which to store the data from the current report.

Details

Path
File ⇨ Save Data Set

Description
To specify an output data set, enter the name of the SAS library and the name of the data set (called **`member`** in the window) that you want to create in the Save Data Set window.

Buttons
OK
creates the output data set and closes the Save Data Set window.

Cancel
closes the Save Data Set window without creating an output data set.

SAVE DEFINITION

Saves a report definition for subsequent use with the same data set or with a similar data set.

Details

Path
File ⇨ Save Report

Description
The SAVE DEFINITION window prompts you for the complete name of the catalog entry in which to store the definition of the current report and for an optional description of the report. This description shows up in the LOAD REPORT window and helps you to select the appropriate report.

SAS stores the report definition as a catalog entry of type REPT. 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 those variables that are used in the report definition.

Buttons

OK
creates the report definition and closes the SAVE DEFINITION window.

Cancel
closes the SAVE DEFINITION window without creating a report definition.

SOURCE

Lists the PROC REPORT statements that build the current report.

Details

Path

Tools ⇨ Report Statements

STATISTICS

Displays statistics that are available in PROC REPORT.

Details

Path

Edit ⇨ Add item ⇨ Statistic

After you select **Statistic**, PROC REPORT prompts you for the location of the statistic relative to the column that you have selected. After you select a location, the STATISTICS window appears.

Description

Select the statistics that you want to include in your report and close the window. When you select the first statistic, it moves to the top of the list in the window. If you select multiple statistics, then subsequent selections move to the bottom of the list of selected statistics. An asterisk (*) indicates each selected statistic. The order of selected statistics from top to bottom determines their order in the report from left to right.

Note: If you double-click on a statistic, then PROC REPORT immediately adds it to the report. The STATISTICS window remains open.

To compute standard error and the Student's *t* test you must use the default value of VARDEF=, which is DF.

To add all selected statistics to the report, select **File ⇨ Accept Selection**. Selecting **File ⇨ Close** closes the STATISTICS window without adding the selected statistics to the report.

WHERE

Selects observations from the data set that meet the conditions that you specify.

Details

Path

Subset ⇨ Where

Description

Enter a *where-expression* in the **Enter WHERE clause** field. A *where-expression* is an arithmetic or logical expression that generally consists of a sequence of operands and operators. For information about constructing a *where-expression*, see the documentation of the "WHERE Statement" in *SAS Statements: Reference* .

Note: You can clear all *where-expressions* by leaving the **Enter WHERE clause** field empty and by selecting **OK**.

Buttons

OK
applies the *where-expression* to the report and closes the WHERE window.

Cancel
closes the WHERE window without altering the report.

WHERE ALSO

Selects observations from the data set that meet the conditions that you specify and any other conditions that are already in effect.

Details

Path

Subset ⇨ Where Also

Description

Enter a *where-expression* in the **Enter where also clause** field. A *where-expression* is an arithmetic or logical expression that generally consists of a sequence of operands and operators. For information about constructing a *where-expression*, see the documentation of the "WHERE Statement" in *SAS Statements: Reference*.

Buttons

OK
adds the *where-expression* to any other *where-expressions* that are already in effect and applies them all to the report. It also closes the WHERE ALSO window.

Cancel
closes the WHERE ALSO window without altering the report.

Chapter 48

SCAPROC Procedure

Overview: SCAPROC Procedure

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.

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.

Note: 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.

Syntax: SCAPROC Procedure

PROC SCAPROC;
  RECORD *filespec* <ATTR> <OPENTIMES>
    <GRID *filespec*<RESOURCE "*resource name*"><NOOPTIMIZE>> ;
  WRITE;

Statement	Task	Example
"PROC SCAPROC Statement"	Implement the SAS Code Analyzer	Ex. 1, Ex. 2
"RECORD Statement"	Specify a filename or a fileref to contain the output of the SAS Code Analyzer	Ex. 1, Ex. 2
"WRITE Statement"	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 1407
"Example 2: Specifying the Grid Job Generator" on page 1408

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 1407
"Example 2: Specifying the Grid Job Generator" on page 1408

Syntax

RECORD *filespec* <ATTR> <OPENTIMES> <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 1404.

Optional Arguments

ATTR
outputs additional information about the variables in the input data sets and views.

OPENTIMES
outputs the open time, size, and physical filename of the input data sets.

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 SASMain.

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 1407

Syntax

WRITE;

Without Arguments

The WRITE statement specifies that the SAS Code Analyzer outputs 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 output information to the specified record file.

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 outputs 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 outputs 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 on 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.

Examples: SCAPROC Procedure

Example 1: Specifying a Record File

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

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

Details

This example writes information from the SAS Code Analyzer to the file named **1.txt**. The example code also runs the Grid Job Generator, and writes that information to the file 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

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 */
```

Chapter 49

SOAP Procedure

Overview: SOAP Procedure

The Simple Object Access Protocol (SOAP) procedure reads XML input from a file that has a fileref and writes XML output to another file that has a fileref. The message component, an XML document that corresponds to a service request, is part of the content of the fileref. It is defined in the IN option of PROC SOAP. The input XML is either a SOAPEnvelope element, or an element inside the SOAPEnvelope that is required to invoke the Web service.

Concepts: SOAP Procedure

With PROC SOAP, you can include an optional SOAPEnvelope element in your XML file. Do this if you want to include custom information in the SOAPHeader element. A SOAP envelope wraps the message, which has an application-specific message vocabulary. The SOAPHeader content will be added to the actual SAS registered Web service request. This addition occurs because there could be additional SOAPHeader elements including elements that support WS-Addressing or WS-Security that were not included in the XML file that was passed to PROC SOAP. The XML code that is transmitted might not exactly match the XML code provided in this case.

A request does not need to be contained in a SOAP envelope. An envelope will be added if you do not specify an envelope. If an envelope is specified, it will be incorporated into the envelope that is sent. A response is returned within an envelope only if the envelope property is set. The default behavior is to return only the contents of the envelope.

You can set the amount of time to wait for a response from the Web service by using the CONFIGFILE option. The default time to wait is 60 seconds.

Requests must not include an encoding declaration even if the envelope is included. If the request is being read from a file, the file must be encoded in the same encoding as the session encoding. Requests are encoded as UTF-8 before being sent to the Web service.

z/OS Specifics

Calling SAS registered services is not available in the z/OS operating environment. SAS registered Web services require WS-Security with Password Digest, and Password Digest is not supported on z/OS.

Syntax: SOAP Procedure

PROC SOAP *options* <*properties*>;

Statement	Task	Example
"PROC SOAP Statement"	Invoke a Web service with a SOAPEnvelope element	Ex. 1
	Invoke a Web service without a SOAPEnvelope element	Ex. 2
	Invoke a Web service by using a proxy	Ex. 3
	Invoke a SAS registered Web service by using the Service Registry Service	Ex. 4
	Invoke a SAS registered Web service by using the SAS environments file	Ex. 5
	Change the default timeout for Web service calls	Ex. 6

PROC SOAP Statement

Invokes a Web service through Java Native Interface (JNI).

Syntax

PROC SOAP *options* <*properties*>;

Summary of Optional Arguments

CONFIGFILE
: sets the timeout limit for Web service calls.

DEBUG
: specifies an output log file.

ENVFILE
: specifies the location of the SAS environments file.

ENVIRONMENT
: specifies that you use the environment that is defined in the SAS environments file.

MUSTUNDERSTAND
: specifies the setting for the mustUnderstand attribute.

OUT=*fileref* '*your-output-file*'
: specifies a fileref for response output.

PROXYDOMAIN
: specifies an HTTP proxy server domain.

PROXYHOST
: specifies an HTTP proxy server host name.

PROXYPASSWORD
: specifies an HTTP proxy server password.

PROXYPORT
: specifies an HTTP proxy server port.

PROXYUSERNAME
: specifies an HTTP proxy server user name.

SOAPACTION
: specifies a SOAPAction element.

SRSURL
: specifies the URL of the System Registry Service.

WEBAUTHDOMAIN
: specifies user name and password retrieval from metadata.

WEBDOMAIN
: specifies the domain or realm for a user name and password.

WEBPASSWORD
: specifies a password for Web service authentication.

WEBUSERNAME
: specifies a user name for Web service authentication.

WSSAUTHDOMAIN
: specifies that the active connection to the SAS Metadata Server will be used to retrieve credentials.

WSSPASSWORD
specifies a WS-Security password.

WSSUSERNAME
specifies a WS-Security user name.

Required Arguments

IN=*fileref* '*your-input-file*'
specifies the fileref that is used to input XML data that contains a PROC SOAP request.

The fileref might have SOAPEnvelope and SOAPHeader elements as part of its content, but they are not required unless you have specific header information to provide.

SERVICE
specifies the SAS registered Web service that you want to call.

Note: If you use the SERVICE option, then do not use the URL option.

URL
specifies the URL of the Web service endpoint.

Note: If you use the URL option, then do not use the SERVICE option.

Optional Arguments

CONFIGFILE
enables you to set the timeout limit for Web service calls. The default timeout is 60 seconds.

DEBUG
enables you to specify an output log file. The debug option turns on wire logging for httpclient and writes the output to the specified file. The value of this option is the path or filename to the desired output.

ENVFILE
specifies the location of the SAS environments file.

ENVIRONMENT
specifies to use the environment that is defined in the SAS environments file.

MUSTUNDERSTAND
specifies the setting for the mustUnderstand attribute in the PROC SOAP header.

OUT=*fileref* '*your-output-file*'
specifies the fileref where the PROC SOAP XML response output is written.

PROXYDOMAIN
specifies an HTTP proxy server domain.

Note: This option is required only if your proxy server requires domain- or realm-qualified credentials.

PROXYHOST
specifies an HTTP proxy server host name.

PROXYPASSWORD
specifies an HTTP proxy server password. Encodings that are produced by PROC PWENCODE are supported.

Note: This option is required only if your proxy server requires credentials.

PROXYPORT
specifies an HTTP proxy server port.

PROXYUSERNAME
specifies an HTTP proxy server user name.

Note: This option is required only if your proxy server requires credentials.

SOAPACTION
specifies a SOAPAction element to invoke on the Web service.

SRSURL
specifies the URL of the System Registry Service.

WEBAUTHDOMAIN
specifies that a user name and password be retrieved from metadata for the specified authentication domain.

WEBDOMAIN
specifies the domain or realm for the user name and password for NTLM authentication.

WEBPASSWORD
specifies a password for either basic or NTLM Web service authentication. Encodings that are produced by PROC PWENCODE are supported.

WEBUSERNAME
specifies a user name for either basic or NTLM Web service authentication.

WSSAUTHDOMAIN
specifies that the active connection to the SAS Metadata Server will be used to retrieve credentials in the specified authentication domain.

If credentials are found, they will be used as the credentials for a WS-Security UsernameToken.

WSSUSERNAME
specifies a WS-Security user name. If a value is set, then WS-Security is used and a UsernameToken is sent with the Web service request for user authentication, security, and encryption.

WSSPASSWORD
specifies a WS-Security password that is the password for WSSUSERNAME. Encodings that are produced by PROC PWENCODE are supported.

Properties

ENVELOPE
specifies that a SOAP envelope is to be included in the response.

Using PROC SOAP with Secure Socket Layer (SSL)

SSL and Data Encryption

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: SSL now has a renegotiation feature that prevents unauthorized text to be added to the beginning or end of an encrypted data stream. This feature is specifically used by certificate-based client authentication. This feature disables SSL renegotiation in the Java Secure Sockets Extension (JSSE) by default. As a result, when you attempt to access a Web resource that requires certificate-based client authentication through the interception proxy, the following Java SSL error message is generated:

```
(javax.net.ssl.SSLException): HelloRequest followed by an unexpected handshake message
```

However, it is still possible to re-enable the SSL renegotiation in Java by setting the new system property

```
sun.security.ssl.allowUnsafeRenegotiation
```

to true before the JSSE library is initialized.

Making PROC SOAP Calls by Using the HTTPS Protocol

In order to make PROC SOAP calls by using the HTTPS protocol, you must configure a trust source that contains the certificate of the service to be trusted. This trust store and its password must be provided to the SAS session by setting Java system options using jreoptions. You can provide this information on the SAS command line or in a SAS configuration file. Use the following syntax. Be sure to enter the following entry on one line:

```
-jreoptions (-Djavax.net.ssl.trustStore=full-path-to-the-trust-store
-Djavax.net.ssl.trustStorePassword=trustStorePassword)
```

The following example shows how to use the entry on the SAS command line. The example uses the Windows operating environment. Be sure to enter the following entry on one line:

```
"C:\Program Files\SAS\SASFoundation\9.3\sas.exe" -CONFIG "C:\Program
Files\SAS\SASFoundation\9.3\nls\en\SASV9.CFG" -jreoptions
(- Djavax.net.ssl.trustStore=C:\Documents and Settings\mydir\.keystore
-Djavax.net.ssl.trustStorePassword=trustpassword)
```

Methods of Calling SAS Registered Web Services

You can use two methods to call SAS registered Web services. The first method requires that you know the URL of the Service Registry Service and the URL of the endpoint of the service that you are calling. You must set the URL of the Service Registry Service on the SRSURL option. The URL option indicates the endpoint of the service that you are calling. See "Example 4: Calling a SAS Registered Web Service Using the Service Registry Service" on page 1420 for an example.

The second method used to call SAS registered Web services uses the SAS environments file to specify the endpoint of the service that you are calling. Using this method, you can indicate the location of the SAS environments file in one of two ways:

- use the ENVFILE option in PROC SOAP
- define the Java property **`env.definition.location`** in JREOPTIONS on the SAS command line or in the SAS configuration file

Use the following –JREOPTIONS syntax:

```
-jreoptions (-Denv.definition.location=http://your-SAS-environment.xml)
```

You must also specify the desired environment within that file using the ENVIRONMENT option, and specify the name of the service that you are calling using the SERVICE option. See "Example 5: Calling a SAS Registered Web Service Using the SAS Environments File" on page 1421 for an example.

In both cases, the WSUSERNAME and WSPASSWORD options will be set to the user name and password that are required to contact the Security Token Service.

Calling a SAS Secured Service without Providing Credentials

You can use the SRSURL, ENVFILE, or ENVIRONMENT options to call a SAS secured service without having to supply a set of credentials. In this case, you do not need to use the WSSUSERNAME, WSSPASSWORD, or WSSAUTHDOMAIN options. This functionality eliminates the need to store user credentials in metadata and reduces the flow of credentials across the network. A connection to the metadata server is required. Credentials that are valid for a single use will be generated for the user that is connected to the metadata server.

Specifying an Output Log File

The log file contains HTTP headers and data that are transmitted to and from servers when transmitting the HTTP request in PROC SOAP. You use the log file that is created with the DEBUG option to log debug output.

To turn on logging to see the SOAP request and response for an entire SAS session, you must restart SAS with the –jreoptions command line option.

PROC SOAP uses log4j for logging requests and responses so that you can trace them. To create a log file that contains the request issued and the response received, create a file that has the following contents:

```
log4j.appender.FILE=org.apache.log4j.FileAppender
log4j.appender.FILE.File=wire.log
log4j.appender.FILE.layout=org.apache.log4j.PatternLayout
log4j.appender.FILE.layout.ConversionPattern =%d %5p [%c] %m%n

log4j.logger.httpclient.wire=DEBUG, FILE
```

Enable logging by setting a Java system option using –jreoptions on the SAS command line or in a SAS configuration file. The following syntax shows how to set the system option:

```
-jreoptions (-Dlog4j.configuration=path-to-log4j-config-file)
```

The following example shows how to use the entry on the SAS command line. The example uses the Windows operating environment. Be sure to enter the entry on one line:

```
"C:\Program Files\SAS\SASFoundation\9.2\sas.exe" -CONFIG "C:\Program
Files\SAS\SASFoundation\9.2\nls\en\SASV9.CFG" -jreoptions
(- Dlog4j.configuration=file:/c:/public/log4j.properties)
```

Using the configuration file and jreoptions method described above turns on logging for the entire SAS session. To turn on httpclient.wire for an individual PROC SOAP call, use the DEBUG option.

You can use the DEBUG option to turn on wire logging for the duration of a PROC SOAP call. The value of the DEBUG option is the path or filename to the output file.

Examples: SOAP Procedure

Example 1: Using PROC SOAP with a SOAPEnvelope Element

Details

This example calls a service that requires WS-Security. It provides an envelope.

Program

```
filename request 'c:\temp\simpleTest_REQUEST.xml';
filename response 'c:\temp\simpleTest_RESPONSE.xml';

data _null_;
   file request;
   input;
   put _infile_;
   datalines4;

<soapenv:Envelope xmlns:add="http://tempuri.org/addintegersWS"
                  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Header/>
   <soapenv:Body>
      <add:addintegers>
         <add:parameters>
            <add:int1>20</add:int1>
            <add:int2>30</add:int2>
         </add:parameters>
      </add:addintegers>
   </soapenv:Body>
   </soapenv:Envelope>
;;;;
run;

%let response=response;
proc soap in=request
          out=&response
          url="http://localhost:8080/SASBIWS/services/addintegersWS"
          wssusername="user-name"
          wsspassword="password";
run;
```

Example 2: Using PROC SOAP without a SOAPEnvelope Element

Details

This example calls the same service as is called in the process described in "Example 1: Using PROC SOAP with a SOAPEnvelope Element" on page 1418. Here the service is called without an envelope.

Program

```
filename request 'c:\temp\simpleTest_REQUEST.xml';
filename response 'c:\temp\simpleTest_RESPONSE.xml';

data _null_;
   file request;
   input;
   put _infile_;
   datalines4;

<add:addintegers xmlns:add="http://tempuri.org/addintegersWS">
   <add:parameters>
      <add:int1>20</add:int1>
         <add:int2>30</add:int2>
   </add:parameters>
</add:addintegers
;;;;
run;

%let response=response;
proc soap in=request
          out=&response
          url="http://localhost:8080/SASBIWS/services/addintegersWS"
          wssusername="user-name"
          wsspassword="password";
run;
```

Example 3: Calling a Web Service by Using a Proxy

Details

This example calls an external Web service and therefore uses a proxy.

Program

```
filename request temp;
filename response "c:\temp\Output.xml";

data _null_;
   file request;
   input;
   put _infile_;
```

```
   datalines4;

<soapenv:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
                  xmlns:ndf="http://www.weather.gov/forecasts/xml/DWMLgen/wsdl/ndfdXML.wsdl">

   <soapenv:Header/>
     <soapenv:Body>
           <ndf:NDFDgenByDay soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
           <latitude xsi:type="xsd:decimal">35.79</latitude>
                       <longitude xsi:type="xsd:decimal">-78.82</longitude>
           <startDate xsi:type="xsd:date">2006-10-03</startDate>
               <numDays xsi:type="xsd:integer">3</numDays>
           <format xsi:type="dwml:formatType"
                   xmlns:dwml="http://www.weather.gov/forecasts/xml/SWMLgen/schema/DWML.xsd">24 hourly</format>
         </ndf:NDFDgenByDay>
   </soapenv:Body>
</soapenv:Envelope>
;;;;

proc soap in=request
          out=response
          url="http://www.weather.gov/forecasts/xml/SOAP_server/ndfdXMLserver.php"
          soapaction="http://www.weather.gov/forecasts/xml/DWMLgen/wsdl/ndfdXML.wsdl#NDFDgenByDay"
          proxyhost="proxygw.abc.sas.com"
          proxyport=80;
run;
```

Example 4: Calling a SAS Registered Web Service Using the Service Registry Service

Details

This example calls a SAS registered Web service by using the service URL and the Service Registry Service.

Program

```
filename request temp;
filename response "c:\temp\output.xml";

data _null_;
   file request;
   input;
   put _infile_;
   datalines4;

<soapenv:Envelope xmlns:rep="http://www.sas.com/xml/schema/sas-svcs/reportrepository-9.2"
                  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header>
      <Action
xmlns="http://schemas.xmlsoap.org/ws/2004/08/addressing">
```

```
      http://www.sas.com/xml/schema/sas-svcs/reportrepository-9.2/DirectoryServiceInterface/isDirectory</Action>
  </soapenv:Header>
  <soapenv:Body>
     <rep:isDirectoryDirectoryServiceInterfaceRequest>
        <rep:dirPathUrl>SBIP:path-name</rep:dirPathUrl>
     </rep:isDirectoryDirectoryServiceInterfaceRequest>
  </soapenv:Body>
</soapenv:Envelope>
;;;;
run;

proc soap in=request out=response
   url="http://machine-name:port-number/SASWIPSoapServices/services/ReportRepositoryService"
   soapaction="http://www.sas.com/xml/schema/sas-svcs/reportrepository-9.2/DirectoryServiceInterface/isDirectory"
   srsurl="http://machine-name:port-number/SASWIPSoapServices/services/ServiceRegistry"
   wssusername="user-name"
   wsspassword="password";
run;
```

Example 5: Calling a SAS Registered Web Service Using the SAS Environments File

Details

This example uses the SAS environments file and the test environment to call the SAS registered Web service.

Program

```
filename request temp;
filename response "c:\temp\output.xml";

data _null_;
   file request;
   input;
   put _infile_;
   datalines4;

<soapenv:Envelope xmlns:rep="http://www.sas.com/xml/schema/sas-svcs/reportrepository-9.2"
                  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header>
      <Action
xmlns="http://schemas.xmlsoap.org/ws/2004/08/addressing">
       http://www.sas.com/xml/schema/sas-svcs/reportrepository-9.2/DirectoryServiceInterface/isDirectory</Action>
  </soapenv:Header>
  <soapenv:Body>
     <rep:isDirectoryDirectoryServiceInterfaceRequest>
        <rep:dirPathUrl>SBIP:path-name</rep:dirPathUrl>
     </rep:isDirectoryDirectoryServiceInterfaceRequest>
  </soapenv:Body>
</soapenv:Envelope>
;;;;
run;
```

```
proc soap in=request out=response service="ReportRepositoryService"
soapaction="http://machine-name:port-number/SASWIPSoapServices/services/ReportRepositoryService"
                  envfile="http://file-server-name.abc.xyz.com/sas-environment.xml"
                  environment="test";
                  wssusername="user-name"
                  wsspassword="password";
run;
```

Example 6: Changing the Default Timeout for Web Service Calls

Details

This example uses the CONFIGFILE option to set the amount of time, in milliseconds, to wait for a SAS registered Web service response. In this example, the soTimeout value is 20000 milliseconds. You can create a config file with the following contents and change the value to set a different timeout. The following example uses the soap-client-config.xml file that is located in the C:Public directory.

Program

```
   /* This section of code is not part of the example. The code  */
   /* is the content of the file that is pointed to by the CONFIGFILE  */
   /* option in the PROC SOAP command. The soTimeout property that is  */
   /* defined in this file is what changes the timeout.                */

<?xml version="1.0" encoding="UTF-8" ?>
-<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:aop="http://www.springframework.org/schema/aop"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
   http://www.springframework.org/schema/aop
          http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">
-<bean id="httpClientParams"
       class="org.apache.commons.httpclient.params.HttpClientParams">
       <property name="soTimeout" value="20000" />
  </bean>
</beans>



   /* This is the beginning of the example. */

filename request "c:\temp\AddInts_request.xml" ;
filename response "c:\temp\AddInts_response.xml" ;

data _null_;
   file request;
   input;
   put _infile_;
   datalines4;
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
```

```
                xmlns:add="http://tempuri.org/AddInts">
   <soapenv:Header/>
   <soapenv:Body>
      <add:addInts>
         <add:parameters>
            <add:int1>2</add:int1>
            <add:int2>3</add:int2>
         </add:parameters>
      </add:addInts>
   </soapenv:Body>
</soapenv:Envelope>
;;;;

proc soap;
   in=request;
   out=response;
   url="http://somehost.abc.xyz.com:8080/SASBIWS/services/AddInts"
   soapaction="http://tempuri.org/AddInts/addInts"
   configfile="c:\public\soap-client-config.xml";
run;
```

Chapter 50

SORT Procedure

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 1448.

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 only 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 50.1 on page 1426 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;
```

Log 50.1 *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 50.1 *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 1451.

Output 50.2 *Observations Sorted by the Values of Three Variables*

Customers with Past-Due Accounts 1
Listed by Town, Amount, Account Number

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

Multi-threaded Sorting

The SAS system option THREADS permits multi-threaded sorting, which is new with SAS System 9. Multi-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 “Support for Parallel Processing” in Chapter 13 of *SAS Language Reference: Concepts*.

Note: The TAGSORT option on page 1442 does not support multi-threaded sorting.

The value of the SAS system option CPUCOUNT= affects the performance of the multi-threaded sort. CPUCOUNT= suggests how many system CPUs are available for use by the multi-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*.

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 either the EBCDIC or the ASCII collating sequence when it compares character values, depending on the environment under which the procedure is running.

For more information about the various collating sequences and when they are used, see“Collating Sequence” in Chapter 3 of *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.

EBCDIC Order

The z/OS operating environment uses the EBCDIC collating sequence.

The sorting order of the English-language EBCDIC sequence is consistent with the following sort order example.

Table 50.1 *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.

ASCII Order

The operating environments that use the ASCII collating sequence include the following:

- UNIX and its derivatives
- Windows
- OpenVMS

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 50.2 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.

Specifying Sorting Orders for Character Variables

The options EBCDIC, ASCII, NATIONAL, DANISH, 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 Chapter 17, "TRANTAB Procedure" in *SAS National Language Support (NLS): Reference Guide*.

Linguistic Collation sorts data according to rules of language. For detailed information about Linguistic Collation, see "Collating Sequence" in Chapter 3 of *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.

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 how 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 429.

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 399.

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, then the data set will not be recognized as being sorted, because the additional variables (beyond 127) are not found within the sort indicator. For a detailed explanation, refer to"What Is a Sort Indicator?" in Chapter 25 of *SAS Language Reference: Concepts*.

Presorted Input Data Sets

Specifying the "PRESORTED" on page 1441 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, if the observation sequence has been verified and an output data set is specified, the observations from the input data set are copied to the output data set, and the metadata for the output data set indicates that the data is validly sorted according to the BY variables.

If observations within the data set are not in sequence, then the data set will be sorted.

If the "NODUPKEY" on page 1439 option has been specified, then the sequence checking determines whether observations with duplicate keys are present in the data set. Otherwise, the input data set is deemed not to be sorted if the NODUPKEY option is specified and observations with duplicate keys are detected.

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 will be performed.

See "Sorted Data Sets" in Chapter 25 of *SAS Language Reference: Concepts* and interactions with the "SORTVALIDATE System Option" in *SAS System Options: Reference*.

Syntax: SORT Procedure

Requirement: BY statement

Tips: You can use the ATTRIB, FORMAT, LABEL, and WHERE statements with the PROC SORT procedure. For more information, see "Statements with the Same Function in Multiple Procedures" on page 35. You can also use any global statement. For a list, see "Global Statements" in Chapter 1 of *SAS Statements: Reference*.

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 1446.

See: "SORT Procedure: Windows" in *SAS Companion for Windows*, "SORT Procedure: z/OS" in *SAS Companion for z/OS*, "SORT Procedure: UNIX" in *SAS Companion for UNIX Environments*.

PROC SORT <*collating-sequence-option*> <*other option(s)*>;
BY <DESCENDING> *variable-1* <…<DESCENDING>*variable-n*>;
KEY *variable(s)* </ option>;

Statement	Task	Example
"PROC SORT Statement"	Order SAS data set observations by the values of one or more character or numeric variables	Ex. 1, Ex. 3, Ex. 4
"BY Statement"	Specify the sorting variables	Ex. 1, Ex. 2, Ex. 4
"KEY Statement"	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 1449
"Example 3: Maintaining the Relative Order of Observations in Each BY Group" on page 1453
"Example 4: Retaining the First Observation of Each BY Group" on page 1456

Syntax

PROC SORT <*collating-sequence-option*> <*other option(s)*>;

Summary of Optional Arguments

ASCII
: specifies ASCII.

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.

REVERSE
reverses the collation order for character variables.

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

NOTHREADS
prevents multi-threaded sorting.

THREADS | NOTHREADS
enables or prevents the activation of multi-threaded sorting.

Specify the collating sequence

DANISH
specifies Danish.

EBCDIC
specifies EBCDIC.

FINNISH
specifies Finnish.

NATIONAL
specifies a customized sequence.

NORWEGIAN
specifies Norwegian.

POLISH
specifies Polish.

SORTSEQ= *collating-sequence*
specifies the collating sequence.

SWEDISH
specifies Swedish.

Specify the output order

EQUALS | NOEQUALS
specifies the relative order within BY groups.

NOEQUALS

does not maintain relative order within BY groups.

Collating-Sequence-Options

Operating Environment Information

For information about behavior specific to your operating environment for the DANISH, FINNISH, NORWEGIAN, or SWEDISH *collating-sequence-option*, see the SAS documentation for your operating environment.

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.

DANISH

sorts characters according to the Danish and Norwegian convention.

The Danish and Norwegian collating sequence is shown in Figure 50.1 on page 1434.

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: "Sorting Orders for Character Variables " on page 1428

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: "Sorting Orders for Character Variables " on page 1428

POLISH

sorts characters according to the Polish convention.

FINNISH

sorts characters according to the Finnish and Swedish convention. The Finnish and Swedish collating sequence is shown in Figure 50.1 on page 1434.

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 have a customized national sort sequence defined. Check with the SAS Installation Representative at your site to determine whether a customized national sort sequence is available.

NORWEGIAN

See "DANISH" on page 1433.

SWEDISH

See "FINNISH" on page 1433.

SORTSEQ= *collating-sequence*

specifies any of the listed collating sequences (ASCII, EBCDIC, DANISH, FINNISH, ITALIAN, NORWEGIAN, POLISH, SPANISH, SWEDISH, or NATIONAL), the name of any other system provided translation table (POLISH, SPANISH), and the name of a user-created translation table. You can specify an encoding. You can also specify either the keyword LINGUISTIC or UCA to achieve a locale appropriate collating sequence. The *collating-sequence* can be a collating-sequence-option, a translation table, an encoding, or the keyword LINGUISTIC. Only one collating sequence can be specified. For detailed information, refer to

"Collating Sequence" in Chapter 3 of *SAS National Language Support (NLS): Reference Guide*.

Here are descriptions of the collating sequences:

collating-sequence-option | *translation_table*
specifies either a translation table, which can be one that SAS provides or any user-defined translation table, or one of the PROC SORT statement *collating-sequence-options*. Refer to the PROC SORT statement "Collating-Sequence-Options" on page 1433. For an example of using PROC TRANTAB and PROC SORT with SORTSEQ=, see "Example 6: Using Different Translation Tables for Sorting" in Chapter 17 of *SAS National Language Support (NLS): Reference Guide*.

These are the available translation tables:

- ASCII
- DANISH
- EBCDIC
- FINNISH
- ITALIAN
- NORWEGIAN
- POLISH
- REVERSE
- SPANISH
- SWEDISH

The following figure shows how the alphanumeric characters in each language will sort.

Figure 50.1 *National Collating Sequences of Alphanumeric Characters*

```
Danish:     0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZÆØÅabcdefghijklmnopqrstuvwxyzæøå
Finnish:    0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZÅÄÖabcdefghijklmnopqrstuvwxyzåäö
Italian:    0123456789AÀBCÇDEÉÈFGHIÌJKLMNOÒPQRSTUÙVWXYZaàbcçdeéèfghiìjklmnoòpqrstuùvwxyz
Norwegian:  0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZÆØÅabcdefghijklmnopqrstuvwxyzæøå
Spanish:    0123456789AÁaáBbCcDdEÉeéFfGgHhIÍiíJjKkLlMmNnÑñOÓoóPpQqRrSsTtUÚuúÜüVvWwXxYyZz
Swedish:    0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZÅÄÖabcdefghijklmnopqrstuvwxyzåäö
```

Restriction: You can specify only one *collating-sequence-option* in a PROC SORT step.

Interaction: In-database processing will not occur when the SORTSEQ= option is specified.

Tips:
The SORTSEQ= *collating-sequence* options are specified without parenthesis and have no arguments associated with them. An example of how to specify a collating sequence follows:

```
proc sort data=mydata SORTSEQ=ASCII;
```

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-rules)*>
specifies linguistic collation, which sorts characters according to rules of the specified language. The rules and default *collating-sequence* options 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).

The following are the *collation-rules* that can be specified for the LINGUISTIC option. These rules 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.

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 50.3 *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 50.4 *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.
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	is the 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= Values and Default Settings for ENCODING, PAPERSIZE, DFLANG, and DATESTYLE Options” in Chapter 18 of *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 50.5 *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 50.6 *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").	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.
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, "ab" < "a-b" < "aB").	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

Restrictions:

The SORTSEQ=LINGUISTIC option is available only on the PROC SORT SORTSEQ= option and is not available for the SAS System SORTSEQ= option.

Linguistic collation is not supported on platforms VMS on Itanium (VMI) or 64-bit Windows on Itanium (W64).

Tips:

The *collating-rules* must be enclosed in parentheses. More than one collating rule 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:

The "ICU License - ICU 1.8.1 and later" on page 1777 agreement.

The section on "Specifying Linguistic Collation" in Chapter 3 of *SAS National Language Support (NLS): Reference Guide* for detailed information.

The *http://www.unicode.org* Web site for the Unicode Collation Algorithm (UCA) specification.

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.

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.

Restriction: For in-database processing to occur, it is necessary that the data set refer to a table residing on the DBMS.

See: "Input Data Sets" on page 20

DATECOPY

copies the SAS internal date and time when the SAS data set was created and the date and time when 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 429.

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.

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 will not sort or overwrite the INPUT data set. Instead, SAS will generate 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.

EQUALS | NOEQUALS

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 multi-threaded sort. However, I/O performance might be reduced when using the EQUALS option with the multi-threaded sort because partitioned data sets are processed as if they consist of a single partition.

The NOEQUALS option is supported by the multi-threaded sort. The order of observations within BY groups that are returned by the multi-threaded sort might not be consistent between runs.

Tip: Using NOEQUALS can save CPU time and memory.

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 Version 5 compatibility engine or with a sequential engine such as a tape format engine, you must also 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, then 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, then the observation is not written to the output data set.

Operating Environment Information

If you use the VMS operating environment and are using the VMS host sort, the observation that is written to the output data set is not always the first observation of the BY group.

Interactions:

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.

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.

Options NODUPKEY and NOUNIQUEKEY are not compatible. If these options are specified together, an error will be printed to the SAS log.

Tips:

Use the EQUALS option with the NODUPKEY option for consistent results in your output data sets.

The DUPOUT= option can be used with the NODUPKEY option. It cannot be combined with the NOUNIQUEKEY option.

Example: "Example 4: Retaining the First Observation of Each BY Group" on page 1456

NOEQUALS

See "EQUALS | NOEQUALS" on page 1439.

NOTHREADS

See "THREADS | NOTHREADS" on page 1443.

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. An observation has a unique sort key when it 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.

Alias: NOUNIKEY | NOUNIKEYS | NOUNIQUEKEYS

Interaction: Options NODUPKEY and NOUNIQUEKEY are not compatible. If NODUPKEY and NOUNIQUEKEY are specified together, an error will be printed to the SAS log.

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 overwrites the original data set.

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=.

Example: "Example 1: Sorting by the Values of Multiple Variables" on page 1449

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 will be lost if a failure occurs while the output data set is being written.

Restrictions:
If the OVERWRITE and the OUT= options are specified and the OUT= data set name is not the same as the INPUT data set name, SAS will 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 multi-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 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 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 1439 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 | NOEQUALS" on page 1439 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*.

REVERSE
sorts character variables using a collating sequence that is reversed from the normal collating sequence.

Operating Environment Information

For information about the normal collating sequence for your operating environment, see "EBCDIC Order" on page 1428, "ASCII Order" on page 1428, and the SAS documentation for your operating environment.

Restriction: The REVERSE option cannot be used with a *collating-sequence-option*. You can specify either a *collating-sequence-option* or the REVERSE option in PROC SORT, but you cannot specify both.

Interaction: Using REVERSE with the DESCENDING option in the BY statement restores the sequence to the normal order.

See: The "DESCENDING" on page 1444 option in the BY statement. The difference is that the DESCENDING option can be used with both character and numeric variables.

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.

*n*K
specifies the amount of memory in kilobytes, where *n* is a real number.

*n*M
specifies the amount of memory in megabytes, where *n* is a real number.

*n*G
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 multi-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 | NOTHREADS

enables or prevents the activation of multi-threaded sorting.

Default: the value of the THREADS | NOTHREADS SAS system option. Note that the default can be overridden using the procedure THREADS | NOTHREADS 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 | 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.

If a failure occurs when adding the THREADS | NOTHREADS procedure option using the SPD engine, PROC SORT stops processing and writes a message to the SAS log.

Interactions:

The PROC SORT THREADS | NOTHREADS options override the SAS system THREADS | NOTHREADS 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 multi-threaded processing is deemed to be beneficial. If the value of the SAS system option CPUCOUNT=1, then multi-threaded processing is not beneficial. However, you can specify the PROC SORT THREADS option to force multi-threaded processing when the system option is set to NOTHREADS or when the system option is THREADS and the procedure option is NOTHREADS. This option combination prevents multi-threaded processing and overrides the actions taken that are based on the system options. Note that when multi-threaded sorting is in effect and NOEQUALS is specified, observations within BY groups might be returned in an unpredictable order.

If multi-threaded SAS sort is being used, the UTILLOC= system option will affect 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= .

The TAGSORT option is not supported by the multi-threaded sort. Specifying the TAGSORT option will prevent multi-threaded processing.

See: "Multi-threaded Sorting" on page 1427 and "Support for Parallel Processing" in Chapter 13 of *SAS Language Reference: Concepts*.

UNIQUEOUT= ***SAS-data-set***

specifies the output data set for observations eliminated by the NOUNIQUEKEY option.

Alias: UNIOUT=

Interaction: The DUPOUT= and UNIOUT= options are not compatible and cannot be specified simultaneously.

Tip: The UNIQUEOUT= option can be used with the NOUNIQUEKEY option. It cannot be combined with the NODUPKEY option.

See: "NOUNIQUEKEY" on page 1440

BY Statement

Specifies the sorting variables.

Examples: "Example 1: Sorting by the Values of Multiple Variables" on page 1449
"Example 2: Sorting in Descending Order" on page 1451
"Example 4: Retaining the First Observation of Each BY Group" on page 1456

Syntax

BY <DESCENDING> *variable-1* <…<DESCENDING>*variable-n*>;

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 | NOTHREADS option is unspecified.

Example: "Example 2: Sorting in Descending Order" on page 1451

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;
```

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 now supports the following database management systems:

- DB2
- Netezza
- Oracle
- Teradata

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 the third maintenance release for SAS 9.2, 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 will perform the sort. The observation ordering that is produced by PROC SORT will depend 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 will affect 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.

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 will 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 1448 . For more information about integrity constraints, see the chapter on “SAS Data Files” in Chapter 26 of *SAS Language Reference: Concepts*.

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 50.7 *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= .

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

Other features: PROC PRINT

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

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

Other features: PROC PRINT

Data set: Account

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

Note that sorting last by AccountNumber puts the businesses in Apex with a debt of $37.95 in order of account number.

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

Other features: 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 will not maintain the order of the observations relative to each other.

```
proc sort data=insurance out=byyears2 noequals;
```

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

Note that sorting with the EQUALS option versus sorting with the NOEQUALS option causes a different sort order for the observations where YearsWorked=3.

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

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

Other features: PROC PRINT

Data set: Account

Note: The EQUALS option, which is the default, must be in effect to ensure that the first observation for each BY group is the one that is retained by the NODUPKEY option. If the NOEQUALS option has been specified, then one observation for each BY group will still be retained by the NODUPKEY option, but not necessarily the first observation.

Details

In this example, PROC SORT creates an output data set that contains only the first observation of each BY group. The NODUPKEY option prevents an observation from being written to the output data set when its BY value is identical to the BY value of the last observation written to the output data set. The resulting report 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. If you use the VMS operating environment sort, then the observation that is written to the output data set is not always the first observation of the BY group.

```
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

The output data set contains only four observations, one for each town in the input data set.

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

Chapter 51

STANDARD Procedure

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 51.1 *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 1471.

Output 51.2 *Z Scores for Each BY Group Using PROC STANDARD*

```
               Life Expectancies by Birth Rate                2

-------------------- PopulationRate=Stable ---------------------

                    The STANDARD Procedure

                                  Standard
Name              Mean       Deviation               N
Label

Life50       67.400000        1.854724               5
1950 life expectancy
Life93       74.500000        4.888763               6
1993 life expectancy


-------------------- PopulationRate=Rapid ----------------------

                                  Standard
Name              Mean       Deviation               N
Label

Life50       42.000000        5.033223               8
1950 life expectancy
Life93       59.100000        8.225300              10
1993 life expectancy
```

```
          Standardized Life Expectancies at Birth            3
                 by a Country's Birth Rate

     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

Tips: Supports the Output Delivery System. For details, see Chapter 3, “Output Delivery System: Basic Concepts,” in *SAS Output Delivery System: User's Guide*.

You can use the ATTRIB, FORMAT, LABEL, and WHERE statements.

PROC STANDARD <*option(s)*>;
BY <DESCENDING> *variable-1* <…<DESCENDING>*variable-n*> <NOTSORTED>;
FREQ *variable*;
VAR *variable(s)*;
WEIGHT *variable*;

Statement	Task	Example
"PROC STANDARD Statement"	Standardize variables to a given mean and standard deviation	Ex. 1, Ex. 2
"BY Statement"	Calculate separate standardized values for each BY group	Ex. 2
"FREQ Statement"	Identify a variable whose values represent the frequency of each observation	
"VAR Statement"	Select the variables to standardize and determine the order in which they appear in the printed output	Ex. 1
"WEIGHT Statement"	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 1471
"Example 1: Standardizing to a Given Mean and Standard Deviation" on page 1468

Syntax

PROC STANDARD <*option(s)*>;

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.

Control printed output

NOPRINT
suppresses all printed output.

PRINT
prints statistics for each variable to standardize.

Preserve values

PRESERVERAWBYVALUES
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 20

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*.

Default: mean of the input values

Example: "Example 1: Standardizing to a Given Mean and Standard Deviation" on page 1468

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 DATA*n*, where *n* is the smallest integer that makes the name unique.

Default: DATA*n*

Example: "Example 1: Standardizing to a Given Mean and Standard Deviation" on page 1468

PRESERVERAWBYVALUES
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 1471

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 1471

STD=*std-value*
standardizes variables to a standard deviation of *std-value*.

Default: standard deviation of the input values

Example: "Example 1: Standardizing to a Given Mean and Standard Deviation" on page 1468

VARDEF=*divisor*
specifies the divisor to use in the calculation of variances and standard deviation. The following table shows the possible values for *divisor* and the associated divisors.

Table 51.1 *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	$(\Sigma_i w_i) - 1$
WEIGHT|WGT	Sum of weights	$\Sigma_i w_i$

The procedure computes the variance as $CSS / divisor$, where CSS is the corrected sums of squares and equals $\Sigma (x_i - \bar{x})^2$. When you weight the analysis variables, CSS equals $\Sigma w_i (x_i - \bar{x}_w)^2$ 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:
"WEIGHT" on page 43
"Keywords and Formulas" on page 1666

BY Statement

Calculates standardized values separately for each BY group.

See: "BY" on page 36

Example: "Example 2: Standardizing BY Groups and Replacing Missing Values" on page 1471

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 the observations in the data set must either be sorted by all the variables that you specify, or they must be indexed appropriately. These variables are called *BY variables*.

Optional Arguments

DESCENDING
specifies that the data set is 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 data 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.

FREQ Statement

Specifies a numeric variable whose values represent the frequency of the observation.

Tip: The effects of the FREQ and WEIGHT statements are similar except when calculating degrees of freedom.

See: For an example that uses the FREQ statement, see "FREQ" on page 40

Syntax

FREQ *variable*;

Required Argument

variable
specifies a numeric variable whose value represents the frequency of the observation. If you use the FREQ statement, the procedure assumes that each observation represents *n* observations, where *n* is the value of *variable*. If *n* is not an integer, the SAS System truncates it. If *n* is less than 1 or is missing, the procedure does not use that observation to calculate statistics but the observation is still standardized.

The sum of the frequency variable represents the total number of observations.

VAR Statement

Specifies the variables to standardize and their order in the printed output.

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 1468

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.

See: For information about calculating weighted statistics and for an example that uses the WEIGHT statement, see "WEIGHT" on page 43.

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.

Weight Value	PROC STANDARD Action
0	Counts the observation in the total number of observations

Weight Value	PROC STANDARD Action
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. See "VARDEF=divisor " on page 1464 and the calculation of weighted statistics in "Keywords and Formulas" on page 1666 for more information.

Details

Note: Before Version 7 of SAS, the procedure did not exclude the observations with missing weights from the count of observations.

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

Other features: 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=stndtest;

   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, stndtest 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 when 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 0s 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 output data set STNDTEST. PROC STANDARD uses a mean of 75 and a standard deviation of 5 to standardize the values. OUT= identifies STNDTEST as the data set to contain the standardized values.

```
proc standard data=score mean=75 std=5 out=stndtest;
```

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 STNDTEST 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, stndtest 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;
```

The following data set contains variables with both standardized and original values. StdTest1 and StdTest2 store the standardized test scores that PROC STANDARD computes.

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

Other features: 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 when 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 LIFEEXP 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.

```
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 output data set ZSCORE. 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 print.

```
proc print data=zscore noobs;
   title 'Standardized Life Expectancies at Birth';
   title2 'by a Country''s Birth Rate';
run;
```

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).

Life Expectancies by Birth Rate 1

PopulationRate=Stable

Name	Mean	Standard Deviation	N	Label
Life50	67.400000	1.854724	5	1950 life expectancy
Life93	74.500000	4.888763	6	1993 life expectancy

PopulationRate=Rapid

Name	Mean	Standard Deviation	N	Label
Life50	42.000000	5.033223	8	1950 life expectancy
Life93	59.100000	8.225300	10	1993 life expectancy

Standardized Life Expectancies at Birth 2
by a Country's Birth Rate

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

Chapter 52

SUMMARY Procedure

Overview: SUMMARY Procedure

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 30, "MEANS Procedure," on page 762. Except for the differences that are discussed here, all the PROC MEANS information also applies to PROC SUMMARY.

Syntax: SUMMARY Procedure

Tips: Supports the Output Delivery System. For details, see Chapter 3, "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 768.

PROC SUMMARY <*option(s)*> <*statistic-keyword(s)*>;
BY <DESCENDING> *variable-1*<…<DESCENDING>*variable-n*> <NOTSORTED>;
CLASS *variable(s)* </ *option(s)*>;
FREQ *variable*;
ID *variable(s)*;
OUTPUT <OUT=*SAS-data-set*><*output-statistic-specification(s)*> <*id-group-specification(s)*> <*maximum-id-specification(s)*> <*minimum-id-specification(s)*></ *option(s)*> ;
TYPES *request(s)*;
VAR *variable(s)*</ WEIGHT=*weight-variable*>;
WAYS *list*;
WEIGHT *variable*;

Statement	Task
"PROC SUMMARY Statement"	Compute descriptive statistics for variables across all observations or within groups of observations
"BY Statement"	Calculate separate statistics for each BY group
"CLASS Statement"	Identify variables whose values define subgroups for the analysis
"FREQ Statement"	Identify a variable whose values represent the frequency of each observation
"ID Statement"	Include additional identification variables in the output data set
"OUTPUT Statement"	Create an output data set that contains specified statistics and identification variables
"TYPES Statement"	Identify specific combinations of class variables to use to subdivide the data
"VAR Statement"	Identify the analysis variables and their order in the results
"WAYS Statement"	Specify the number of ways to make unique combinations of class variables
"WEIGHT Statement"	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 769.

Details

PRINT | NOPRINT
specifies whether PROC SUMMARY displays the descriptive statistics. By default, PROC SUMMARY produces no display output, but PROC MEANS does produce display output.

Default: NOPRINT

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, 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 793.

Chapter 53

TABULATE Procedure

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

Simple Tables

The following output shows a simple table that was produced by PROC TABULATE. The data set "ENERGY" on page 1742 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 53.1 *Simple Table Produced by PROC TABULATE*

The SAS System

		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 53.1 on page 1481. 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 1555.

Output 53.2 *Complex Table Produced by PROC TABULATE*

Energy Expenditures for Each Region 2
(millions of dollars)

		Customer Base		
		Residential Customers	Business Customers	All 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 display shows a table that is created in Hypertext Markup Language (HTML). You can use the Output Delivery System with PROC TABULATE to create customized output in HTML, Rich Text Format (RTF), Portable Document Format (PDF), and other output formats. For an explanation of the program that produces this table, see "Example 14: Specifying Style Elements for ODS Output" on page 1593.

Output 53.3 *HTML Table Produced by PROC TABULATE*

Energy Expenditures
(millions of dollars)

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
West	Division			
	Mountain	$5,476	$4,729	$10,205
	Pacific	$13,959	$12,619	$26,578
	Total	$19,435	$17,348	$36,783
Total	Division			
	New England	$7,477	$5,129	$12,606
	Middle Atlantic	$19,379	$15,078	$34,457
	Mountain	$5,476	$4,729	$10,205
	Pacific	$13,959	$12,619	$26,578
	Total	$46,291	$37,555	$83,846

Concepts: TABULATE Procedure

Terminology: TABULATE Procedure

The following figures illustrate some of the terms that are commonly used in discussions of PROC TABULATE.

Figure 53.1 *Parts of a PROC TABULATE Table*

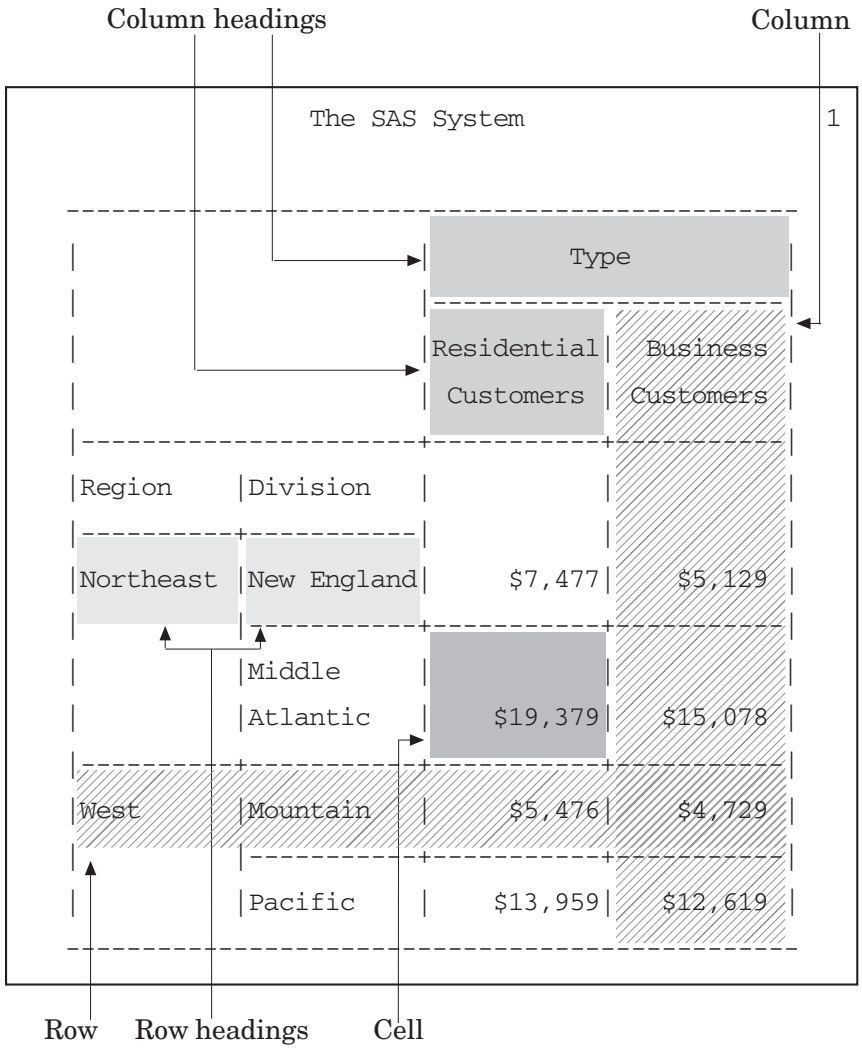
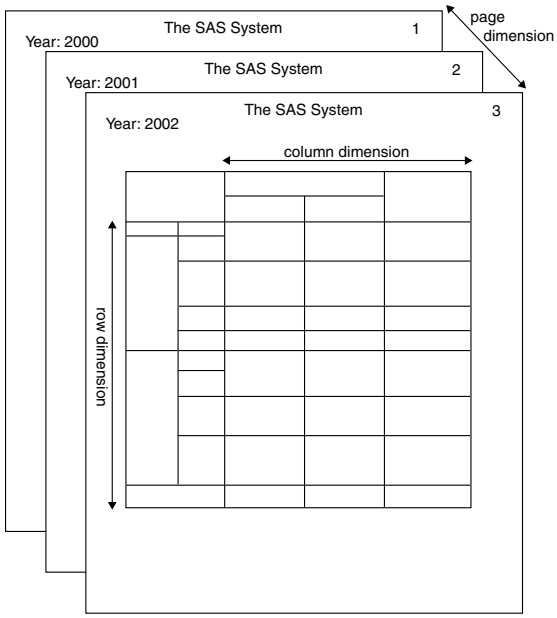


Figure 53.2 *PROC TABULATE Table Dimensions*



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 53.1 on page 1484 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 53.1 *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 53.1 on page 1484, 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 53.2 on page 1485, 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 "Introduction to the Output Delivery System" in Chapter 3 of *SAS Output Delivery System: User's Guide*.

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 53.1 on page 1484 contains no subtables. For an illustration of a table that consists of multiple subtables, see "Example 13: Using Denominator Definitions to Display Basic Frequency Counts and Percentages" on page 1578.

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	P90
P5	P95
P10	P99
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 1666 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 ranging 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 23, "FORMAT Procedure," on page 626.

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

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. See "TABLE Statement" on page 1516 for more information. 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.

How Using BY-Group Processing Differs from Using the Page Dimension

Using the page-dimension expression in a TABLE statement can have an effect similar to using a BY statement.

The following table contrasts the two methods.

Table 53.2 *Contrasting the BY Statement and the Page Dimension*

Issue	PROC TABULATE with a BY Statement	PROC TABULATE with a Page Dimension in the TABLE Statement
Order of observations in the input data set	The observations in the input data set must be sorted by the BY variables. [1]	Sorting is unnecessary.
One report summarizing all BY groups	You cannot create one report for all the BY groups.	Use ALL in the page dimension to create a report for all classes. (See "Example 6: Summarizing Information with the Universal Class Variable ALL" on page 1555.)
Percentages	The percentages in the tables are percentages of the total for that BY group. You cannot calculate percentages for a BY group compared to the totals for all BY groups because PROC TABULATE prepares the individual reports separately. Data for the report for one BY group are not available to the report for another BY group.	You can use denominator definitions to control the meaning of PCTN. (See "Calculating Percentages " on page 1490.)
Titles	You can use the #BYVAL, #BYVAR, and #BYLINE specifications in TITLE statements to customize the titles for each BY group. (See "Creating Titles That Contain BY-Group Information " on page 21.)	The BOX= option in the TABLE statement customizes the page headings, but you must use the same title on each page.
Ordering class variables	ORDER=DATA and ORDER=FREQ order each BY group independently.	The order of class variables is the same on every page.
Obtaining uniform headings	You might need to insert dummy observations into BY groups that do not have all classes represented.	The PRINTMISS option ensures that each page of the table has uniform headings.
Multiple ranges with the same format	PROC TABULATE produces a table for each range.	PROC TABULATE combines observations from the two ranges.

[1] You can use the BY statement without sorting the data set if the data set has an index for the BY variable.

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 12: Calculating Various Percentage Statistics" on page 1575 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 13: Using Denominator Definitions to Display Basic Frequency Counts and Percentages" on page 1578 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 1742.

```
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 53.3 *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/rts=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 53.4 *Three Different Uses of the PCTSUM Statistic with Sums Highlighted*

Expenditures in Each Division

		Type	
		1	2
		Expenditu-res	Expenditu-res
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 Style Elements in PROC TABULATE

What Are Style Elements?

If you use the Output Delivery System to create HTML, RTF, or Printer output from PROC TABULATE, then you can set the style element that the procedure uses for various parts of the table. Style elements determine presentation attributes, such as font face, font weight, color, and so on. See "Understanding Styles, Style Elements, and Style Attributes" in Chapter 3 of *SAS Output Delivery System: User's Guide* for more information. See Appendix 5, "ODS Style Elements," in *SAS Output Delivery System: User's Guide* for a comprehensive list of style elements.

The following table lists the default styles for various regions of a table.

Table 53.3 *Default Styles for Table Regions*

Region	Style
Column headings	Heading
Box	Heading
Page dimension text	Beforecaption
Row headings	Rowheading
Data cells	Data
Table	Table

Using the STYLE= Option

You specify style elements for PROC TABULATE with the STYLE= option. The following table shows where you can use this option. Specifications in the TABLE statement override the same specification in the PROC TABULATE statement. 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 STYLE= is provided in the documentation for individual statements.

Table 53.4 *Using the STYLE= Option in PROC TABULATE*

To Set the Style Element For	Use STYLE in this Statement
Data cells	PROC TABULATE statement or dimension expression(s) (p. 1496)
Page dimension text and class variable name headings	"CLASS Statement" (p. 1509)

To Set the Style Element For	Use STYLE in this Statement
Class level value headings	"CLASSLEV Statement" (p. 1513)
Keyword headings	"KEYWORD Statement" (p. 1515)
Table borders, rules, and other parts that are not specified elsewhere	"TABLE Statement" (p. 1516)
Box text	BOX= option of the TABLE statement (p. 1516)
Missing values	MISSTEXT= option of the TABLE statement (p. 1516)
Analysis variable name headings	"VAR Statement" (p. 1526)

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 1516. 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 html body='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 html close;
```

Syntax: TABULATE Procedure

Requirements: At least one TABLE statement is required.

Depending on the variables that appear in the TABLE statement, a CLASS statement, a VAR statement, or both are required.

Tips: Supports the Output Delivery System. See “Overview of How ODS Works” in Chapter 3 of *SAS Output Delivery System: User's Guide* for details.

You can use the ATTRIB, FORMAT, LABEL, and WHERE statements with the PROC TABULATE procedure. For more information, see “Statements with the Same Function in Multiple Procedures” on page 35. You can also use any global statement. For a list, see “Global Statements” in Chapter 1 of *SAS Statements: Reference*.

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 1528.

PROC TABULATE <*option(s)*>;
  BY <DESCENDING> *variable-1*
    <…<DESCENDING>*variable-n*>
    <NOTSORTED>;
  CLASS *variable(s)* </ *options*>;
  CLASSLEV *variable(s)* /
    STYLE=<*style-element-name* | PARENT>
    <[*style-attribute-specification(s)*] >;
  FREQ *variable*;
  KEYLABEL *keyword-1='description-1'*
    <…*keyword-n='description-n'*>;
  KEYWORD *keyword(s)* /
    STYLE=<*style-element-name* | PARENT>
    <[*style-attribute-specification(s)*] >;
  TABLE <<*page-expression*,>*row-expression*,> *column-expression*</ *table-option(s)*>;
  VAR *analysis-variable(s)*</ *options*>;
  WEIGHT *variable*;

Statement	Task	Example
“PROC TABULATE Statement”	Display descriptive statistics in tabular format	Ex. 1, Ex. 2, Ex. 3, Ex. 4, Ex. 6, Ex. 8,

Statement	Task	Example
		Ex. 12, Ex. 14, Ex. 15
"BY Statement"	Create a separate table for each BY group	
"CLASS Statement"	Identify variables in the input data set as class variables	Ex. 3, Ex. 4
"CLASSLEV Statement"	Specify a style for class variable level value headings	Ex. 14, Ex. 15
"FREQ Statement"	Identify a variable in the input data set whose values represent the frequency of each observation	
"KEYLABEL Statement"	Specify a label for a keyword	
"KEYWORD Statement"	Specify a style for keyword headings	Ex. 14
"TABLE Statement"	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, Ex. 15
"VAR Statement"	Identify variables in the input data set as analysis variables	Ex. 14
"WEIGHT Statement"	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.

Examples:

"Example 1: Creating a Basic Two-Dimensional Table" on page 1540
"Example 2: Specifying Class Variable Combinations to Appear in a Table" on page 1543
"Example 3: Using Preloaded Formats with Class Variables" on page 1545
"Example 4: Using Multilabel Formats" on page 1549
"Example 6: Summarizing Information with the Universal Class Variable ALL" on page 1555
"Example 8: Indenting Row Headings and Eliminating Horizontal Separators" on page 1559
"Example 12: Calculating Various Percentage Statistics" on page 1575
"Example 14: Specifying Style Elements for ODS Output" on page 1593
"Example 15: Style Precedence" on page 1598

Syntax

PROC TABULATE <*option(s)*>;

Summary of Optional Arguments

CONTENTS=*link-name*
: customizes the HTML contents link to the output.

DATA=*SAS-data-set*
: specifies the input data set.

NOTHREADS
: disables parallel processing of the input data set.

OUT=*SAS-data-set*
: specifies the output data set.

THREADS | NOTHREADS
: overrides the SAS system option THREADS | NOTHREADS.

TRAP
: enables floating point exception recovery.

Control the statistical analysis

ALPHA=*value*
: specifies the confidence level for the confidence limits.

EXCLNPWGT
: excludes observations with nonpositive weights.

PCTLDEF=
: specifies the mathematical definition to calculate quantiles.

QMARKERS=*number*
: specifies the sample size to use for the P^2 quantile estimation method.

QMETHOD=OS|P2|HIST
: specifies the quantile estimation method.

QNTLDEF=1|2|3|4|5
: specifies the mathematical definition to calculate quantiles.

VARDEF=*divisor*
: specifies the variance divisor.

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-element-name*|<PARENT>>[*style-attribute-name=style-attribute-value*<… *style-attribute-name=style-attribute-value*>]
: specifies the default style element or style elements (for the Output Delivery System) to use for each cell 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–*value*)×100. For example, **`ALPHA=.05`** results in a 95% confidence limit.

Default: .05

Range: between 0 and 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 combinations of values of the class variables that occur in the CLASSDATA= data set but not in the input data set appear in each table or output data set 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 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 1543

CONTENTS=*link-name*
enables you to name the link in the HTML table of contents that points to the ODS output of the first table that was produced by using the TABULATE procedure.

Note: CONTENTS= affects only the contents file of ODS HTML output. It has no effect on the actual TABULATE procedure reports.

DATA=*SAS-data-set*
specifies the input data set.

See: "Input Data Sets" on page 20

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 1527 and "WEIGHT Statement" on page 1527

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 this option is ignored.

Example: "Example 2: Specifying Class Variable Combinations to Appear in a Table" on page 1543

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: This 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 1540

"Example 6: Summarizing Information with the Universal Class Variable ALL" on page 1555

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 pound sign (#) to the seventh character, and does not alter the remaining characters:

```
formchar(3,7)='*#'
```

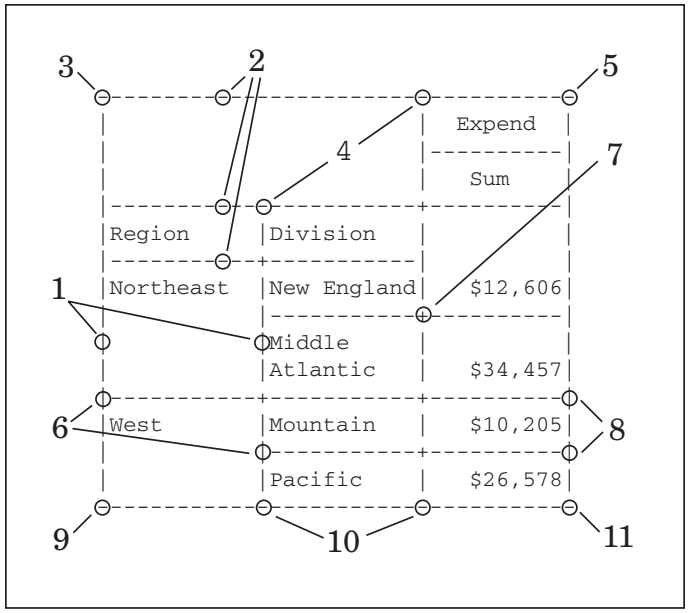
PROC TABULATE uses 11 of the 20 formatting characters that SAS provides. Table 53.5 on page 1499 shows the formatting characters that PROC TABULATE uses. Figure 53.5 on page 1500 illustrates the use of each formatting character in the output from PROC TABULATE.

Table 53.5 *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

Position	Default	Used to Draw
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
11	-	Bottom character in the right border

Figure 53.5 *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 2D 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 1533

"Creating Special Missing Values" in Chapter 5 of *SAS Language Reference: Concepts* in 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 1499.

Example: "Example 8: Indenting Row Headings and Eliminating Horizontal Separators" on page 1559

NOTHREADS

See "THREADS | NOTHREADS" on page 1507.

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 those procs In-database and you are using the order=data proc option, the results can vary because there really is no sense of ordering on the database unless you use order by (which the procs don't do).

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.

Alias: 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.

Alias: 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 1539

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 1545

PCTLDEF=
See "QNTLDEF=1|2|3|4|5" on page 1503.

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: an 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 798

QMETHOD=OS|P2|HIST
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|HIST
uses the P^2 method to approximate the quantile.

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 798

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 1671

STYLE=<*style-element-name*|<PARENT>>[*style-attribute-name=style-attribute-value*<… *style-attribute-name=style-attribute-value*>]

specifies the style element to use for the data cells of a table when it is used in the PROC TABULATE statement. For example, the following statement specifies that the background color for data cells be red:

```
proc tabulate data=one style=[backgroundcolor=red];
```

Note: This option can be used in other statements, or in dimension expressions, to specify style elements for other parts of a table.

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 a style definition that is registered with the Output Delivery System. SAS provides some style definitions. You can create your own style definitions with PROC TEMPLATE.

Default: If you do not specify a style element, then PROC TABULATE uses Data.

See: "Concepts: Styles and the TEMPLATE Procedure " in Chapter 13 of *SAS Output Delivery System: User's Guide* for information about PROC TEMPLATE and the default style definitions.For information about the style elements, see Appendix 5, "ODS Style Elements," in *SAS Output Delivery System: User's Guide*.

<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. Curly 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. The following table shows attributes that you can set or change with the STYLE= option in the PROC TABULATE statement (or in any other PROC TABULATE statement that uses the STYLE= option, except for the TABLE statement). Note that not all attributes are valid in all destinations.

Table 53.6 *Style Attributes for PROC REPORT and PROC TABULATE*

Attribute	PROC REPORT STATEMENT: REPORT Area	PROC REPORT Areas: CALLDEF, COLUMN, HEADER, LINES, SUMMARY	PROC TABULATE STATEMENT: TABLE	PROC TABULATE STATEMENTS: VAR, CLASS, BOXOpt, CLASSLEV, KEYWORD
ASIS=	X	X		X
BACKGOUNDCOLOR=	X	X	X	X
BACKGOUNDIMAGE=	X	X	X	X
BORDERBOTTOMCOLOR=	X	X		X
BORDERBOTTOMSTYLE=	X	X	X	X
BORDERBOTTOMWIDTH=	X	X	X	X
BORDERCOLOR=	X	X		X
BORDERCOLORDARK=	X	X	X	X
BORDERCOLORLIGHT=	X	X	X	X
BORDERTOPCOLOR=	X	X		X
BORDERTOPSTYLE=	X	X	X	X
BORDERTOPWIDTH=	X	X	X	X
BORDERWIDTH=	X	X	X	X
CELLPADDING=	X		X	
CELLSPACING=	X		X	
CELLWIDTH=	X	X	X	X
CLASS=	X	X	X	X
COLOR=	X	X	X	
FLYOVER=	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, BOXOpt, CLASSLEV, KEYWORD
FONT=	X	X	X	X
FONTFAMILY=	X	X	X	X
FONTSIZE=	X	X	X	X
FONTSTYLE=	X	X	X	X
FONTWEIGHT=	X	X	X	X
FONTWIDTH=	X	X	X	X
FRAME=	X		X	
HEIGHT=	X	X		X
HREFTARGET=		X		X
HTMLSTYLE=	X	X	X	X
NOBREAKSPACE=	X	X		X
OUTPUTWIDTH=	X	X	X	X
POSTHTML=	X	X	X	X
POSTIMAGE=	X	X	X	X
POSTTEXT=	X	X	X	X
PREHTML=	X	X	X	X
PREIMAGE=	X	X	X	X
PRETEXT=	X	X	X	X
PROTECTSPECIALCHARS=		X		X
RULES=	X		X	
TAGATTR=	X	X		X
TEXTALIGN=	X	X	X	X
URL=		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, BOXOpt, CLASSLEV, KEYWORD
VERTICALALIGN=		X		X
WIDTH=	X	X	X	X

See: "Style Attributes Tables " in Chapter 13 of *SAS Output Delivery System: User's Guide*

style-attribute-value
specifies a value for the attribute. Each attribute has a different set of valid values. See "Style Attributes Overview" in Chapter 13 of *SAS Output Delivery System: User's Guide*

Alias: S=

Restriction: This option affects only the HTML, RTF, and Printer destinations.

Tip: To specify a style element for data cells with missing values, use STYLE= in the TABLE statement MISSTEXT= option.

See: "Using Style Elements in PROC TABULATE " on page 1493

Example: "Example 14: Specifying Style Elements for ODS Output" on page 1593

THREADS | NOTHREADS
enables or disables parallel processing of the input data set. This option overrides the SAS system option THREADS | NOTHREADS unless the system option is restricted. (See Chapter 13, "Support for Parallel Processing," in *SAS Language Reference: Concepts* for more information.)

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 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.

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=*divisor*
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.

Table 53.7 *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	$(\Sigma_i w_i) - 1$
WEIGHT | WGT	Sum of weights	$\Sigma_i w_i$

The procedure computes the variance as *CSS* / *divisor*, where *CSS* is the corrected sums of squares and equals $\Sigma (x_i - \bar{x})^2$. When you weight the analysis variables, *CSS* equals $\Sigma w_i(x_i - \bar{x}_w)^2$ 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.

See: "Weighted Statistics Example" on page 44

BY Statement

Creates a separate table on a separate page for each BY group.

See: "BY" on page 36

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 the observations in the data set must either 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. For example, the observations are grouped in 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.

CLASS Statement

Identifies class variables for the table. Class variables determine the categories that PROC TABULATE uses to calculate statistics.

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 1545
"Example 4: Using Multilabel Formats" on page 1549

Syntax

CLASS *variable(s)* </*option(s)*>;

Summary of Optional Arguments

ASCENDING
DESCENDING
EXCLUSIVE
GROUPINTERNAL
MISSING

MLF

ORDER=DATA | FORMATTED | FREQ | UNFORMATTED

PRELOADFMT

STYLE=<*style-element-name*|<PARENT>>[*style-attribute-name=style-attribute-value*<… *style-attribute-name=style-attribute-value*>]

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.

Optional Arguments

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 1545

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: This 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 MISSING, then PROC TABULATE excludes the observations with any missing CLASS variable values from tables and output data sets.

See: "Creating Special Missing Values" in Chapter 5 of *SAS Language Reference: Concepts* 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 658 in the VALUE statement of the FORMAT procedure.

Example: "Example 4: Using Multilabel Formats" on page 1549

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 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.

Alias: 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.

Alias: 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 1539

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.

Example: "Example 3: Using Preloaded Formats with Class Variables" on page 1545

STYLE=<*style-element-name*|<PARENT>>[*style-attribute-name=style-attribute-value*<… *style-attribute-name=style-attribute-value*>]
specifies the style element to use for page dimension text and class variable name headings. For more information about the arguments for this option and how it is used, see STYLE= on page 1504 in the PROC TABULATE statement.

Note: 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.

Note: 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.

Alias: S=

Restriction: This option affects only the HTML, RTF, and Printer destinations.

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.

Example: "Example 14: Specifying Style Elements for ODS Output" on page 1593

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.

Restriction: This statement affects only the HTML, RTF, and Printer destinations.

Examples: "Example 14: Specifying Style Elements for ODS Output" on page 1593
"Example 15: Style Precedence" on page 1598

Syntax

CLASSLEV *variable(s)* / STYLE=<*style-element-name* | <PARENT>>
[*style-attribute-name= style-attribute-value* <… *style-attribute-name= style-attribute-value*>];

Required Argument

variable(s)

specifies one or more class variables from the CLASS statement for which you want to specify a style element.

Optional Argument

STYLE=<*style-element-name*|<PARENT>>[*style-attribute-name=style-attribute-value*<… *style-attribute-name=style-attribute-value*>]

specifies a style element for class variable level value headings. For information about the arguments of this option and how it is used, see STYLE= on page 1504 in the PROC TABULATE statement.

Note: 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.

Alias: S=

Restriction: This option affects only the HTML, RTF, and Printer destinations.

Tip: 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.

Example: "Example 14: Specifying Style Elements for ODS Output" on page 1593

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.

See: "FREQ" on page 40

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.

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-n*='*description-n*'>;

Required Arguments

keyword
is one of the keywords for statistics that is discussed in "Statistics That Are Available in PROC TABULATE" on page 1486 or is the universal class variable ALL. (See "Elements That You Can Use in a Dimension Expression" on page 1523.)

description
is up to 256 characters to use as a label. As the syntax shows, you must enclose *description* in quotation marks.

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.

KEYWORD Statement

Specifies a style element for keyword headings.

Restriction: This statement affects only the HTML, RTF, and Printer output.

Example: "Example 14: Specifying Style Elements for ODS Output" on page 1593

Syntax

KEYWORD *keyword(s)* / STYLE=<*style-element-name* | <PARENT>>
[*style-attribute-name=style-attribute-value*<… *style-attribute-name=style-attribute-value*>];

Required Argument

keyword
is one of the keywords for statistics that is discussed in "Statistics That Are Available in PROC TABULATE" on page 1486 or is the universal class variable ALL. (See "Elements That You Can Use in a Dimension Expression" on page 1523.)

Optional Argument

STYLE=<*style-element-name*|<PARENT>>[*style-attribute-name=style-attribute-value*<… *style-attribute-name=style-attribute-value*>]
specifies a style element for the keyword headings. For information about the arguments of this option and how it is used, see STYLE= on page 1504 in the PROC TABULATE statement.

Note: 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.

Alias: S=

Restriction: This option affects only the HTML, RTF, and Printer destinations.

Tip: 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.

Example: "Example 14: Specifying Style Elements for ODS Output" on page 1593

TABLE Statement

Describes a table to print.

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 26.

Examples:

"Example 1: Creating a Basic Two-Dimensional Table" on page 1540
"Example 3: Using Preloaded Formats with Class Variables" on page 1545
"Example 4: Using Multilabel Formats" on page 1549
"Example 5: Customizing Row and Column Headings" on page 1553
"Example 6: Summarizing Information with the Universal Class Variable ALL" on page 1555
"Example 7: Eliminating Row Headings" on page 1557
"Example 8: Indenting Row Headings and Eliminating Horizontal Separators" on page 1559
"Example 9: Creating Multipage Tables" on page 1561
"Example 10: Reporting on Multiple-Response Survey Data" on page 1564
"Example 11: Reporting on Multiple-Choice Survey Data" on page 1568
"Example 12: Calculating Various Percentage Statistics" on page 1575
"Example 13: Using Denominator Definitions to Display Basic Frequency Counts and Percentages" on page 1578
"Example 14: Specifying Style Elements for ODS Output" on page 1593
"Example 15: Style Precedence" on page 1598
"Example 16: NOCELLMERGE Option" on page 1602

Syntax

TABLE <<*page-expression*,>*row-expression*,>
column-expression </ *table-option(s)*>;

Summary of Optional Arguments

BOX=*value*
BOX={<*label*=*value*> <STYLE=<*style-element-name*>[*style-attribute-name=style-attribute-value*<… *style-attribute-name=style-attribute-value*>]>}
CONDENSE
CONTENTS=*link-name*
FORMAT_PRECEDENCE=PAGE|ROW|COLUMN|COL
FUZZ=*number*
INDENT=*number-of-spaces*
MISSTEXT='*text*'

MISSTEXT={<*label*='*text*'> <STYLE=<*style-element-name*> [*style-attribute-name=style-attribute-value* <... *style-attribute-name=style-attribute-value*>]>}

NOCELLMERGE

NOCONTINUED

page-expression

PRINTMISS

ROW=*spacing*

row-expression

RTSPACE=*number*

STYLE_PRECEDENCE=PAGE|ROW|COLUMN|COL

STYLE=<*style-element-name*> [*style-attribute-name=style-attribute-value*<… *style-attribute-name=style-attribute-value*>]

Required Argument

column-expression

defines the columns in the table. For information about constructing dimension expressions, see "Details" on page 1523.

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 1523.

Restriction: A page dimension is the first dimension in a table statement. Both a row dimension and a column dimension must follow a page dimension.

Example: "Example 9: Creating Multipage Tables" on page 1561

row-expression

defines the rows in the table. For information about constructing dimension expressions, see "Details" on page 1523.

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*
BOX={<*label*=*value*> <STYLE=<*style-element-name*>[*style-attribute-name=style-attribute-value*<… *style-attribute-name=style-attribute-value*>|>}

specifies text and a style element 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 1504 in the PROC TABULATE statement.

Examples:
"Example 9: Creating Multipage Tables" on page 1561
"Example 14: Specifying Style Elements for ODS Output" on page 1593

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. 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: CONDENSE 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 9: Creating Multipage Tables" on page 1561

CONTENTS=*link-name*
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.

Note: CONTENTS= affects only the contents file of ODS HTML output. It has no effect on the actual TABULATE procedure reports.

FORMAT_PRECEDENCE=PAGE|ROW|COLUMN|COL
specifies whether the format 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.

Default: COLUMN

FUZZ=*number*
supplies a numeric value against which analysis variable values and table cell values other than frequency counts are compared to eliminate trivial values (absolute values less than the FUZZ= value) from computation and printing. A number whose absolute value is less than the FUZZ= value is treated as zero in computations and printing. 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 HTML, 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.

See: "Example 8: Indenting Row Headings and Eliminating Horizontal Separators" on page 1559 (with crossings) "Example 9: Creating Multipage Tables" on page 1561 (without crossings)

MISSTEXT='*text*'
MISSTEXT={<*label*='*text*'> <STYLE=<*style-element-name*> [*style-attribute-name=style-attribute-value <... style-attribute-name=style-attribute-value*>]>}
supplies up to 256 characters of text to print and specifies a style element 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 1504 in the PROC TABULATE 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 1536
"Example 14: Specifying Style Elements for ODS Output" on page 1593

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 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 16: NOCELLMERGE Option" on page 1602

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 HTML 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 PRINTMISS, 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 does not print regardless of the PRINTMISS option.

See: "CLASSDATA=SAS-data-set" on page 1498

Example: "Providing Headings for All Categories" on page 1535

ROW=*spacing*
specifies whether all title elements in a row crossing are allotted space even when they are blank. The possible values for *spacing* are as follows:

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 1557

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 1540

STYLE=<*style-element-name*> [*style-attribute-name=style-attribute-value<… style-attribute-name=style-attribute-value*>]
specifies a style element to use for parts of the table other than table cells. See STYLE= on page 1504 in the PROC TABULATE statement for information about the style element arguments.

Note: You can use braces ({ and }) instead of square brackets ([and]).

The following table shows the attributes that you can set or change with the STYLE= option in the TABLE statement. Most of these attributes apply to parts of the table other than cells (for example, table borders and the lines between columns and rows). Attributes that you apply in the PROC TABULATE statement and in other locations in the PROC TABULATE step apply to cells within the table. Note that not all attributes are valid in all destinations. See "Style Attributes Tables " in Chapter 13 of *SAS Output Delivery System: User's Guide* for information about these style attributes, their valid values, and their applicable destinations.

Table 53.8 *Style Attributes for PROC REPORT and PROC TABULATE*

Attribute	PROC REPORT STATEMENT: REPORT Area	PROC REPORT Areas: CALLDEF, COLUMN, HEADER, LINES, SUMMARY	PROC TABULATE STATEMENT: TABLE	PROC TABULATE STATEMENTS: VAR, CLASS, BOXOpt, CLASSLEV, KEYWORD
ASIS=	X	X		X
BACKGOUNDCOLOR=	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, BOXOpt, CLASSLEV, KEYWORD
BACKGOUNDIMAGE=	X	X	X	X
BORDERBOTTOMCOLOR=	X	X		X
BORDERBOTTOMSTYLE=	X	X	X	X
BORDERBOTTOMWIDTH=	X	X	X	X
BORDERCOLOR=	X	X		X
BORDERCOLORDARK=	X	X	X	X
BORDERCOLORLIGHT=	X	X	X	X
BORDERTOPCOLOR=	X	X		X
BORDERTOPSTYLE=	X	X	X	X
BORDERTOPWIDTH=	X	X	X	X
BORDERWIDTH=	X	X	X	X
CELLPADDING=	X		X	
CELLSPACING=	X		X	
CELLWIDTH=	X	X	X	X
CLASS=	X	X	X	X
COLOR=	X	X	X	
FLYOVER=	X	X		X
FONT=	X	X	X	X
FONTFAMILY=	X	X	X	X
FONTSIZE=	X	X	X	X
FONTSTYLE=	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, BOXOpt, CLASSLEV, KEYWORD
FONTWEIGHT=	X	X	X	X
FONTWIDTH=	X	X	X	X
FRAME=	X		X	
HEIGHT=	X	X		X
HREFTARGET=		X		X
HTMLSTYLE=	X	X	X	X
NOBREAKSPACE=	X	X		X
OUTPUTWIDTH=	X	X	X	X
POSTHTML=	X	X	X	X
POSTIMAGE=	X	X	X	X
POSTTEXT=	X	X	X	X
PREHTML=	X	X	X	X
PREIMAGE=	X	X	X	X
PRETEXT=	X	X	X	X
PROTECTSPECIALCHARS=		X		X
RULES=	X		X	
TAGATTR=	X	X		X
TEXTALIGN=	X	X	X	X
URL=		X		X
VERTICALALIGN=		X		X
WIDTH=	X	X	X	X

Note: The list of attributes that you can set or change with the STYLE= option in the TABLE statement differs from the list of attributes of the PROC TABULATE statement.

Alias: S=

Restriction: This option affects only the HTML, RTF, and Printer destinations.

Tip: 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.

Example: "Example 14: Specifying Style Elements for ODS Output" on page 1593

STYLE_PRECEDENCE=PAGE|ROW|COLUMN|COL
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.

Default: COLUMN

Example: "Example 15: Style Precedence" on page 1598

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 1526.)

class variables
(See the CLASS statement on page 1509.)

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 1555

"Example 9: Creating Multipage Tables" on page 1561

"Example 13: Using Denominator Definitions to Display Basic Frequency Counts and Percentages" on page 1578

keywords for statistics
See "Statistics That Are Available in PROC TABULATE" on page 1486 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 10: Reporting on Multiple-Response Survey Data" on page 1564

"Example 13: Using Denominator Definitions to Display Basic Frequency Counts and Percentages" on page 1578

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 1488.

Example:

```
Sales*f=dollar8.2
```

Example: "Example 6: Summarizing Information with the Universal Class Variable ALL" on page 1555

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 1553

"Example 7: Eliminating Row Headings" on page 1557

style-element specifications

specify style elements for page dimension text, headings, or data cells. For details, see "Specifying Style Elements in Dimension Expressions" on page 1525.

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 1540

(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 1555

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 1555

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 1490.

Examples:

"Example 10: Reporting on Multiple-Response Survey Data" on page 1564

"Example 13: Using Denominator Definitions to Display Basic Frequency Counts and Percentages" on page 1578

Specifying Style Elements in Dimension Expressions

You can specify a style element in a dimension expression to control the appearance in HTML, RTF, and Printer output of the following table elements:

- analysis variable name headings
- class variable name headings
- class variable level value headings
- data cells
- keyword headings
- page dimension text

Specifying a style element in a dimension expression is useful when you want to override a 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 a style element in a dimension expression is

[STYLE<(CLASSLEV)>=<*style-element-name* | PARENT>
[*style-attribute-name=style-attribute-value*<… *style-attribute-name=style-attribute-value*>]]

Some examples of style elements in dimension expressions are

- `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 }).

With the exception of (CLASSLEV), all arguments are described in STYLE= on page 1504 in the PROC TABULATE statement.

(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: This option is used only in dimension expressions.

For an example that shows how to specify style elements within dimension expressions, see "Example 14: Specifying Style Elements for ODS Output" on page 1593.

VAR Statement

Identifies numeric variables to use as analysis variables.

Alias: VARIABLES

Tip: You can use multiple VAR statements.

Example: "Example 14: Specifying Style Elements for ODS Output" on page 1593

Syntax

VAR *analysis-variable(s)* </ *option(s)*>;

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.

Optional Arguments

STYLE=<*style-element-name*|<PARENT>>[*style-attribute-name=style-attribute-value*<… *style-attribute-name=style-attribute-value*>]
specifies a style element for analysis variable name headings. For more information about the arguments of this option, see STYLE= on page 1504.

Note: 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.

Alias: S=

Restriction: This option affects only the HTML, RTF, and Printer destinations.

Tip: 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.

Example: "Example 14: Specifying Style Elements for ODS Output" on page 1593

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. If the value of the weight variable is

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=divisor " on page 1507.

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.

See: For information about calculating weighted statistics and for an example that uses the WEIGHT statement, see "Calculating Weighted Statistics" on page 44.

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.

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=divisor " on page 1507and the calculation of weighted statistics in the "Keywords and Formulas" on page 1666 section of this document.

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 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 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.

In-database processing for PROC TABULATE supports the following database management systems:

- DB2
- Oracle
- Teradata
- Netezza

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: N, NMISS, MIN, MAX, RANGE, SUM, SUMWGT, CSS, USS, VAR, STD, STDERR, UCLM, LCLM, and CV.

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*.

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 53.9 *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
An observation contains a missing value for a class variable	Excludes that observation from the table[1]	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

[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.

formats that are used in this section and prints the data set. The data set COMPREV contains no missing values. (See the following figure.)

```
proc format;
   value cntryfmt 1='United States'
                  2='Japan';
   value compfmt  1='Supercomputer'
                  2='Mainframe'
                  3='Midrange'
                  4='Workstation'
                  5='Personal Computer'
                  6='Laptop';
run;

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 cntryfmt. computer compfmt.;
   title 'The Data Set COMPREV';
run;
```

Figure 53.6 *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 figure:

```
proc tabulate data=comprev;
   class country computer;
   var rev90 rev91 rev92;
   table computer*country,rev90 rev91 rev92 /
         rts=32;
   format country cntryfmt. 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.

Figure 53.7 *Computer Sales Data: No Missing Values*

Revenues from Computer Sales
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 Figure 53.8 on page 1533 is the same as the program that produces Figure 53.7 on page 1532. By default, 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 cntryfmt. 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 Figure 53.7 on page 1532.

Figure 53.8 *Computer Sales Data: Midrange, Japan, Deleted*

Revenues from Computer Sales
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 figure.)

```
proc tabulate data=compmiss missing;
   class country computer;
   var rev90 rev91 rev92;
   table computer*country,rev90 rev91 rev92 /
         rts=32;
   format country cntryfmt. 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.

Figure 53.9 *Computer Sales Data: Missing Values for Computer*

Revenues from Computer Sales
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 Figure 53.9 on page 1534, 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 figure.)

```
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 cntryfmt. 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.

Figure 53.10 *Computer Sales Data: Text Supplied for Missing Computer Value*

Revenues for Computer Sales
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, Figure 53.10 on page 1535 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 figure.)

```
proc tabulate data=compmiss missing;
   class country computer;
   var rev90 rev91 rev92;
   table computer*country,rev90 rev91 rev92 /
         rts=32 printmiss;
   format country cntryfmt. 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.

Figure 53.11 *Computer Sales Data: Missing Statistics Values*

Revenues for Computer Sales
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 figure.)

```
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 cntryfmt. 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.

Figure 53.12 *Computer Sales Data: Text Supplied for Missing Statistics Values*

Revenues for Computer Sales 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 1545 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 1543 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 figure.

```
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 cntryfmt. computer compfmt.;
   title 'Revenues for Computer Sales';
   title2 'for 1990 to 1992';
run;
```

This table contains a heading for each possible value of Computer.

Figure 53.13 *Computer Sales Data: All Possible Computer Values Included*

Revenues for Computer Sales
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 figure.) 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 figure 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.

```
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;
```

Figure 53.14 *Ordering the Headings of Class Variables*

Revenues for Computer Sales
for 1990 to 1992

Animal					
cat			dog		
Food			Food		
fish	meat	milk	fish	meat	bones
N	N	N	N	N	N
1	1	1	1	1	1

Portability of ODS Output with PROC TABULATE

Under certain circumstances, using PROC TABULATE with the Output Delivery System produces files that are not portable. If the SAS system option FORMCHAR= in your SAS session uses nonstandard line-drawing characters, then the output might include strange characters instead of lines in operating environments in which the SAS Monospace font is not installed. To avoid this problem, specify the following OPTIONS statement before executing PROC TABULATE:

```
options formchar="|----|+|---+=|-/\<>*";
```

Examples: TABULATE Procedure

Example 1: Creating a Basic Two-Dimensional Table

Features: PROC TABULATE statement options
FORMAT=
TABLE statement
crossing (*) operator
TABLE statement options
RTS=

Other features: FORMAT 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

```
data energy;
   length State $2;
   input Region Division state $ Type Expenditures;
   datalines;
1 1 ME 1 708
1 1 ME 2 379

 ... more data lines ...

4 4 HI 1 273
4 4 HI 2 298
;

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 ENERGY data set. ENERGY contains data on expenditures of energy for business and residential customers in individual states in the Northeast and West regions of the United States. A DATA step on page 1742 creates the data set.

```
data energy;
   length State $2;
```

```
   input Region Division state $ Type Expenditures;
   datalines;
1 1 ME 1 708
1 1 ME 2 379

 ... more data lines ...

4 4 HI 1 273
4 4 HI 2 298
;
```

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 53.4 *Basic Two-Dimensional Table*

Energy Expenditures for Each Region
(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: PROC TABULATE Statement options
CLASSDATA=
EXCLUSIVE

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 1540.

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

Energy Expenditures for Each Region
(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: PROC TABULATE statement option
OUT=
CLASS statement options
EXCLUSIVE
PRELOADFMT
TABLE statement option
PRINTMISS

Other features: PRINT procedure

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`**.

Energy Expenditures for Each Region (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
	Mountain	.	.
	Pacific	.	.
South	New England	.	.
	Middle Atlantic	.	.
	Mountain	.	.
	Pacific	.	.
Midwest	New England	.	.
	Middle Atlantic	.	.
	Mountain	.	.
	Pacific	.	.
West	New England	.	.
	Middle Atlantic	.	.
	Mountain	$5,476	$4,729
	Pacific	$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 1540.

Energy Expenditures for Each Region
(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.

Energy Expenditures for Each Region
(millions of dollars)

Obs	Region	Division	Type	_TYPE_	_PAGE_	_TABLE_	Expenditures_Sum
1	Northeast	New England	Residential Customers	111	1	1	7477
2	Northeast	New England	Business Customers	111	1	1	5129
3	Northeast	Middle Atlantic	Residential Customers	111	1	1	19379
4	Northeast	Middle Atlantic	Business Customers	111	1	1	15078
5	West	Mountain	Residential Customers	111	1	1	5476
6	West	Mountain	Business Customers	111	1	1	4729
7	West	Pacific	Residential Customers	111	1	1	13959
8	West	Pacific	Business Customers	111	1	1	12619

Example 4: Using Multilabel Formats

Features: CLASS statement options
MLF

PROC TABULATE statement options
:FORMAT=

TABLE statement
- ALL class variable
- concatenation (blank) operator
- crossing (*) operator
- grouping elements (parentheses) operator
- label
- variable list

Other features: FORMAT procedure

FORMAT statement

VALUE statement options
- MULTILABEL

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

```
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

 ... more data lines ...

77   61  92  88  77  85
78   24  87  88  88  91
79   18  54  50  62  74
80   62  90  91  90  86
;

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 CARSURVEY data set. CARSURVEY contains data from a survey that was distributed by a car manufacturer to a focus group of potential customers who were brought together to evaluate new car names. Each observation in the data set contains an identification number, the participant's age, and the participant's ratings of four car names. A DATA step creates the data set.

```
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

 ... more data lines ...

77   61  92  88  77  85
78   24  87  88  88  91
79   18  54  50  62  74
80   62  90  91  90  86
;
```

Create the AGEFMT. format. The FORMAT procedure creates a multilabel format for ages by using the "MULTILABEL" on page 658. 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

Rating Four Potential Car Names
Rating Scale 0-100 (100 is the highest rating)

	N	Potential Car Names			
		Progressa	Remark	Jupiter	Dynamo
		Mean	Mean	Mean	Mean
Age					
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: TABLE statement
labels

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.

Energy Expenditures for Each Region
(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: PROC TABULATE statement options
FORMAT=

TABLE statement
ALL class variable
concatenation (blank operator)
format modifiers
grouping elements (parentheses operator)

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.

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

Example 7: Eliminating Row Headings

Features: TABLE statement
labels
ROW=FLOAT

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 1553. 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.

Energy Expenditures for Each Region
(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: Indenting Row Headings and Eliminating Horizontal Separators

Features: PROC TABULATE statement options
NOSEPS
TABLE statement options
INDENT=

Data set: ENERGY

Format: REGFMT.

Format: DIVFMT.

Format: USETYPE.

Details

This example shows how to condense the structure of a table by doing the following:

- removing row headings for class variables
- indenting nested rows underneath parent rows instead of placing them next to each other
- eliminating horizontal separator lines from the row titles and the body of the table

Program

```
options nodate nonumber;
ods listing;

proc tabulate data=energy format=dollar12. noseps;

   class region division type;

   var expenditures;

    table region*division,
          type='Customer Base'*expenditures=' '*sum=' '
```

```
          / rts=25 indent=4;

   format region regfmt. division divfmt. type usetype.;

   title 'Energy Expenditures for Each Region';
   title2 '(millions of dollars)';
run;

ods listing close;
```

Program Description

Open the LISTING destination. The INDENT argument does not indent nested row headings for HTML output. The output will be captured as a listing with page numbering and date turned off.

```
options nodate nonumber;
ods listing;
```

Specify the table options. The FORMAT= option specifies DOLLAR12. as the default format for the value in each table cell. NOSEPS eliminates horizontal separator lines from row titles and from the body of the table.

```
proc tabulate data=energy format=dollar12. noseps;
```

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 in all dimensions 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 and indention value. RTS= provides 25 characters per line for row headings. INDENT= removes row headings for class variables, places values for Division beneath values for Region rather than beside them, and indents values for Division four spaces.

```
          / rts=25 indent=4;
```

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;
```

Close the LISTING destination.

```
ods listing close;
```

Output

NOSEPS removes the separator lines from the row titles and the body of the table. INDENT= eliminates the row headings for Region and Division and indents values for Division underneath values for Region.

Energy Expenditures for Each Region
(millions of dollars)

	Customer Base	
	Residential Customers	Business Customers
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 9: Creating Multipage Tables

Features: TABLE statement
ALL class variable
BOX=
CONDENSE
INDENT=
page expression

Data set: 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

Energy Expenditures for Each Region and All Regions
(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 10: Reporting on Multiple-Response Survey Data

Features: TABLE statement
- denominator definition (angle bracket operators)
- N statistic
- PCTN statistic
- variable list

Other features: FORMAT procedure

SAS system options
- FORMDLIM=
- NONUMBER

SYMPUT routine

Data set: CUSTOMER_RESPONSE

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.

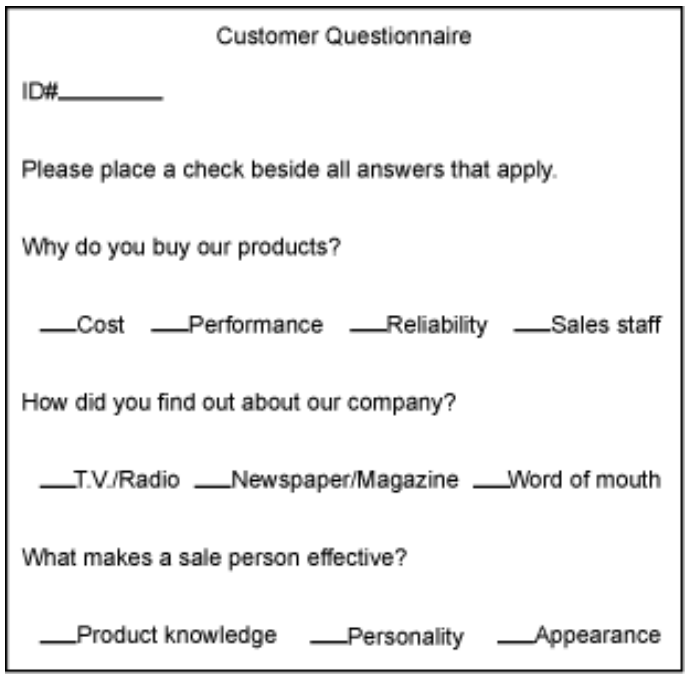
In addition to showing how to create these tables, this example shows how to do the following:

- 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

Collecting the Data

The following figure shows the survey form that is used to collect data.

Output 53.5 *Completed Survey Form*


Customer Questionnaire

ID#_____

Please place a check beside all answers that apply.

Why do you buy our products?

___Cost ___Performance ___Reliability ___Sales staff

How did you find out about our company?

___T.V./Radio ___Newspaper/Magazine ___Word of mouth

What makes a sale person effective?

___Product knowledge ___Personality ___Appearance

Program

```
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 . . . .

. . . more data lines . . .

119 . . . 1 . . . 1 . .
120 1 1 . 1 . . . . 1 .
;

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. pctn<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. pctn<customer>='Percent'*f=pctfmt9.) ;

   title 'Source of Company Name';
   footnote "Number of Respondents: &num";
run;

options formdlim='' number;
```

Program Description

Create the CUSTOMER_RESPONSE data set. CUSTOMER_RESPONSE contains data from a customer survey. Each observation in the data set contains information about factors that influence one respondent's decisions to buy products. A DATA step on page 1737 creates the data set. Using missing values rather than 0s is crucial for calculating frequency counts in PROC TABULATE.

```
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 . . . .

. . . more data lines . . .

119 . . . 1 . . . 1 . .
120 1 1 . 1 . . . . 1 .
;
```

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. pctn<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. pctn<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

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 11: Reporting on Multiple-Choice Survey Data

Features: TABLE statement
N statistic

Other features: FORMAT procedure
TRANSPOSE procedure
Data set options
RENAME=

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.

Collecting the Data

The following figure shows the survey form that is used to collect data.

Output 53.6 *Completed Survey Form*

phone_ _ _

LISTENER SURVEY

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 1763 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=(col1=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 1573 for a complete explanation of the PROC TRANSPOSE step.)

```
proc transpose data=radio
               out=radio_transposed(rename=(col1=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

Listening Preferences on Weekdays

	Choice of Radio Program					
	Don't Listen	Rock and Top 40	Country	Jazz, Classical, and Easy Listening	News/ Information /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. This section illustrates how PROC TRANSPOSE reshapes the data. The following section explains the PROC TRANSPOSE step in this example.

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.

Output 53.7 *Transposing Two Observations*

Input Data Set

Time1	Time2	Time3	Time4	Time5	Time6	Time7	Listener
7	5	5	5	7	0	0	1
5	0	0	0	5	0	0	2

❷ ❸ ❶

Listener	_NAME_	COL1
1	Time1	7
1	Time2	7
1	Time3	5
1	Time4	5
1	Time5	7
1	Time6	0
1	Time7	0
2	Time1	5
2	Time2	0
2	Time3	0
2	Time4	0
2	Time5	5
2	Time6	0
2	Time7	0

Output Data Set

❶ The BY variable is not transposed. All the observations created from the same BY group contain the same value of Listener.

❷ _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.

❸ COL1 contains the values of Time1–Time7.

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 12: Calculating Various Percentage Statistics

Features: PROC TABULATE statement options
- FORMAT=

TABLE statement
- ALL class variable
- COLPCTSUM statistic
- concatenation (blank) operator
- crossing (*) operator
- format modifiers
- grouping elements (parentheses) operator
- labels
- REPPCTSUM statistic
- ROWPCTSUM statistic
- variable list

TABLE statement options
- ROW=FLOAT
- RTS=

Other features: FORMAT procedure

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

Fundraiser Sales

	Classroom								All
	A				B				
	Sum	ColPctSum	RowPctSum	RepPctSum	Sum	ColPctSum	RowPctSum	RepPctSum	sales
team									
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:

- COLPCTSUM=31/91*100=34%
- ROWPCTSUM=31/67*100=46%
- REPPCTSUM=31/204*100=15%

Similar calculations were used to produce the output for the remaining teams and classrooms.

Example 13: Using Denominator Definitions to Display Basic Frequency Counts and Percentages

Features: TABLE statement
ALL class variable
denominator definitions (angle bracket operators)
N statistic
PCTN statistic

Other features: FORMAT procedure

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 1  1 1
1 2  1 2  1 2  1 2  1 2  1 2  1 2
1 3  1 3  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 2  1 2  1 3  1 3  1 4  1 4  1 4
1 4  1 4  1 4  1 1  1 1  1 1  1 1
1 1  1 2  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 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
;

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 1  1 1
1 2  1 2  1 2  1 2  1 2  1 2  1 2
1 3  1 3  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 2  1 2  1 3  1 3  1 4  1 4  1 4
1 4  1 4  1 4  1 1  1 1  1 1  1 1
1 1  1 2  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 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

Gender Distribution
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`**

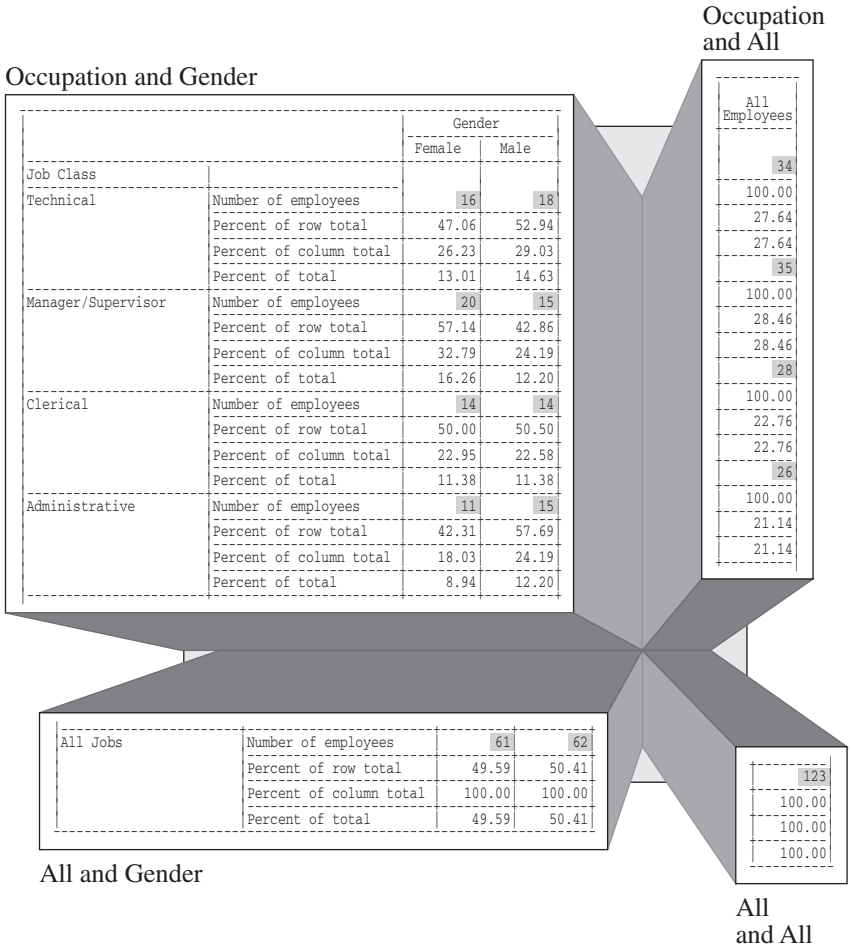
The following table describes each subtable.

Table 53.10 *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.

Output 53.8 *Illustration of the Four Subtables*



		Gender	
		Female	Male
Job Class			
Technical	Number of employees	16	18
	Percent of row total	47.06	52.94
	Percent of column total	26.23	29.03
	Percent of total	13.01	14.63
Manager/Supervisor	Number of employees	20	15
	Percent of row total	57.14	42.86
	Percent of column total	32.79	24.19
	Percent of total	16.26	12.20
Clerical	Number of employees	14	14
	Percent of row total	50.00	50.50
	Percent of column total	22.95	22.58
	Percent of total	11.38	11.38
Administrative	Number of employees	11	15
	Percent of row total	42.31	57.69
	Percent of column total	18.03	24.19
	Percent of total	8.94	12.20

All Employees
34
100.00
27.64
27.64
35
100.00
28.46
28.46
28
100.00
22.76
22.76
26
100.00
21.14
21.14

All Jobs	Number of employees	61	62
	Percent of row total	49.59	50.41
	Percent of column total	100.00	100.00
	Percent of total	49.59	50.41

123
100.00
100.00
100.00

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 53.9 *Subtable 1: Occupation and Gender*

Gender Distribution
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 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 53.10 *Subtable 2: All and Gender*

Gender Distribution
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 53.11 *Subtable 3: Occupation and All*

Gender Distribution
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.

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 53.12 *Subtable 4: All and All*

Gender Distribution
within Job Classes

		Gender		All
		Female	Male	Employees
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 53.13 *Subtable 1: Occupation and Gender*

Gender Distribution
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 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 53.14 *Subtable 2: All and Gender*

Gender Distribution
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 53.15 *Subtable 3: Occupation and All*

Gender Distribution
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 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 53.16 *Subtable 4: All and All*

Gender Distribution
within Job Classes

		Gender		All
		Female	Male	Employees
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 53.11 *Denominators for Total Percentages*

Class Variables Contributing to the Subtable	Frequency Counts	Total
Occupat and Gender	16, 18, 20, 15 14, 14, 11, 15	123

Class Variables Contributing to the Subtable	Frequency Counts	Total
Occupat 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 14: Specifying Style Elements for ODS Output

Features: STYLE= option
PROC TABULATE statement
CLASSLEV statement
KEYWORD statement
TABLE statement
VAR statement

Other features: ODS HTML statement
ODS PDF statement
ODS RTF statement

Data set: ENERGY

Format: REGFMT.

Format: DIVFMT.

Format: USETYPE.

Details

This example creates HTML, RTF, and PDF files and specifies style elements for various table regions.

Program

```
options nodate pageno=1;

ods html body='external-HTML-file';
ods pdf file='external-PDF-file';
ods rtf file='external-RTF-file';

proc tabulate data=energy style=[fontweight=bold];

   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 html close;
ods pdf close;
ods rtf 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. LINESIZE= and PAGESIZE= are not set for this example because they have no effect on HTML, RTF, and Printer output.

```
options nodate pageno=1;
```

Specify the ODS output filenames. By opening multiple ODS destinations, you can produce multiple output files in a single execution. The ODS HTML statement produces output that is written in HTML. 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 TABULATE goes to each of these files.

```
ods html body='external-HTML-file';
ods pdf file='external-PDF-file';
ods rtf file='external-RTF-file';
```

Specify the table options. The STYLE= option in the PROC TABULATE statement specifies the style element for the data cells of the table.

```
proc tabulate data=energy style=[fontweight=bold];
```

Specify subgroups for the analysis. The STYLE= option in the CLASS statement specifies the style element for the class variable name headings.

```
   class region division type / style=[textalign=center];
```

Specify the style attributes for the class variable value headings. The STYLE= option in the CLASSLEV statement specifies the style element for the class variable level value headings.

```
   classlev region division type / style=[textalign=left];
```

Specify the analysis variable and its style attributes. 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 the table rows and columns and their style attributes. The STYLE= option in the dimension expression overrides any other STYLE= specifications in PROC TABULATE that specify attributes for table cells. The STYLE= option after the slash (/) specifies attributes 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]
```

Specify the style attributes for cells with 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]]
```

Specify the style attributes for the box above the row titles. The STYLE= option in the BOX option of the TABLE statement specifies a style element to use for text in the box above the row titles.

```
      box=[label="Region by Division by Type"
         style=[fontstyle=italic]];
```

Format the class variable values. The FORMAT statement assigns formats to Region, Division, and Type.

```
format region regfmt. division divfmt. type usetype.;
```

Specify the titles.

```
   title 'Energy Expenditures';
   title2 '(millions of dollars)';
run;
```

Close the ODS destinations.

```
ods html close;
ods pdf close;
ods rtf close;
```

Output

Output 53.17 *HTML Output*

Energy Expenditures
(millions of dollars)

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
West	Division			
	Mountain	$5,476	$4,729	$10,205
	Pacific	$13,959	$12,619	$26,578
	Total	$19,435	$17,348	$36,783
Total	Division			
	New England	$7,477	$5,129	$12,606
	Middle Atlantic	$19,379	$15,078	$34,457
	Mountain	$5,476	$4,729	$10,205
	Pacific	$13,959	$12,619	$26,578
	Total	$46,291	$37,555	$83,846

Output 53.18 *PDF Output*

Energy Expenditures
(millions of dollars)

Region by Division by Type		Type		
		Residential Customers	Business Customers	Total
		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
West	Division			
	Mountain	$5,476	$4,729	$10,205
	Pacific	$13,959	$12,619	$26,578
	Total	$19,435	$17,348	$36,783
Total	Division			
	New England	$7,477	$5,129	$12,606
	Middle Atlantic	$19,379	$15,078	$34,457
	Mountain	$5,476	$4,729	$10,205
	Pacific	$13,959	$12,619	$26,578
	Total	$46,291	$37,555	$83,846

Output 53.19 *RTF Output*

Energy Expenditures
(millions of dollars)

Region by Division by Type		Type		
		Residential Customers	Business Customers	Total
		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
West	Division			
	Mountain	$5,476	$4,729	$10,205
	Pacific	$13,959	$12,619	$26,578
	Total	$19,435	$17,348	$36,783
Total	Division			
	New England	$7,477	$5,129	$12,606
	Middle Atlantic	$19,379	$15,078	$34,457
	Mountain	$5,476	$4,729	$10,205
	Pacific	$13,959	$12,619	$26,578
	Total	$46,291	$37,555	$83,846

Example 15: Style Precedence

Features: PROC TABULATE statement options
FORMAT=
STYLE= option
CLASSLEV statement
TABLE statement
TABLE statement
crossing (*) operator
STYLE_PRECEDENCE= option

Other features: ODS HTML statement
FORMAT 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 = red, yellow, or orange) color based on the STYLE_PRECEDENCE = option
- generates ODS HTML output

Program

```
proc format;
      value $saletypefmt 'R'='Retail'
                         'W'='WholeSale';
run;

ods html file="stylePrecedence.html";

title "Style Precedence";
title2 "First Table: no precedence, Orange";
title3 "Second Table: style_precedence=page, Red";

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=red}},
region*{style={background=yellow}},
saletype*{style={background=orange}};

table product *{style={background=red}},
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 HTML statement produces output that is written in HTML.

```
ods html file="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, Red";
```

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=red}},
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 red.

```
table product *{style={background=red}},
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 53.20 *Style Precedence*

Style Precedence
First Table: no precedence, Orange
Second Table: style_precedence=page, Red

Product A100

	SaleType	
	Retail	*WholeSale*
	N	N
Region		
NC	$3	$3
TX	.	$2

Style Precedence
First Table: no precedence, Orange
Second Table: style_precedence=page, Red

Product A100

	SaleType	
	Retail	*WholeSale*
	N	N
Region		
NC	$3	$3
TX	.	$2

Example 16: NOCELLMERGE Option

Features: PROC TABULATE statement options
STYLE=
CLASS statement
TABLE statement
crossing (*) operator
STYLE= option
NOCELLMERGE= option

Other features: ODS HTML 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 html file="tabstyle.html";

proc tabulate data=sashelp.class style={background=red};

class sex age;


table sex*{style={background=blue}} all, age;

title 'Data Cell Styles in Merged Cells';

run;


proc tabulate data=sashelp.class style={background=red};

class sex age;


table sex*{style={background=blue}} all, age/nocellmerge;

title1 'Data Cell Styles with NOCELLMERGE Option';

run;


ODS HTML close;
```

Program Description

Specify the ODS output filename. The ODS HTML statement produces output that is written in HTML.

```
ods html file="tabstyle.html";
```

Specify the PROC TABULATE options. The STYLE= option sets the background color for the cells in the table to red.

```
proc tabulate data=sashelp.class style={background=red};
```

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.

```
table sex*{style={background=blue}} 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 red.

```
proc tabulate data=sashelp.class style={background=red};
```

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=blue}} 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 re not merged.

```
title1 'Data Cell Styles with NOCELLMERGE Option';
```

Run the program.

```
run;
```

Close the ODS HTML output destination.

```
ODS HTML close;
```

Output

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

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 Chlamtac, Imrich "The P² Algorithm for Dynamic Calculation of Quantiles and Histograms without Storing Observations." 1985. *Communications of the Association of Computing Machinery* 28 (10): 1076–1085.

Chapter 54

TIMEPLOT Procedure

Overview: TIMEPLOT Procedure

The TIMEPLOT procedure plots one or more variables over time intervals. A listing of variable values accompanies the plot. Although the plot and the listing are similar to the ones produced by the PLOT and PRINT procedures, PROC TIMEPLOT output has these distinctive features:

- The vertical axis always represents the sequence of observations in the data set. Thus, if the observations are in order of date or time, then the vertical axis represents the passage of time.
- The horizontal axis represents the values of the variable that you are examining. Like PROC PLOT, PROC TIMEPLOT can overlay multiple plots on one set of axes so that each line of the plot can contain values for more than one variable.
- A plot produced by PROC TIMEPLOT can occupy more than one page.
- Each observation appears sequentially on a separate line of the plot; PROC TIMEPLOT does not hide observations as PROC PLOT sometimes does.
- The listing of the plotted values can include variables that do not appear in the plot.

The following output illustrates a simple report that you can produce with PROC TIMEPLOT. This report shows sales of refrigerators for two sales representatives during the first six weeks of the year. The statements that produce the output follow. A DATA step in "Example 1: Plotting a Single Variable" on page 1619 creates the data set SALES.

```
title 'The SAS System';
options source;

options linesize=64 pagesize=60 nodate
        pageno=1;

proc timeplot data=sales;
   plot icebox;
   id month week;
   title 'Weekly Sales of Refrigerators';
   title2 'for the';
   title3 'First Six Weeks of the Year';
run;
```

Output 54.1 *Simple Report Created with PROC TIMEPLOT*

```
                Weekly Sales of Refrigerators                 1
                          for the
                  First Six Weeks of the Year

Month     Week      Icebox      min                          max
                                2520.04                  3550.43
                              *--------------------------------*
    1        1     3450.94     |                              I  |
    1        1     2520.04     |I                                |
    1        2     3240.67     |                        I        |
    1        2     2675.42     |     I                           |
    1        3     3160.45     |                    I            |
    1        3     2805.35     |          I                      |
    1        4     3400.24     |                             I    |
    1        4     2870.61     |            I                    |
    2        1     3550.43     |                               I|
    2        1     2730.09     |      I                          |
    2        2     3385.74     |                             I    |
    2        2     2670.93     |     I                           |
                              *--------------------------------*
```

The following output is a more complicated report of the same data set that is used to create Output 54.1 on page 1608. The statements that create this report do the following:

- create one plot for the sale of refrigerators and one for the sale of stoves
- plot sales for both sales representatives on the same line
- identify points on the plots by the first letter of the sales representative's last name
- control the size of the horizontal axis
- control formats and labels

For an explanation of the program that produces this report, see "Example 5: Showing Multiple Observations on One Line of a Plot" on page 1629 .

Output 54.2 *More Complex Report Created with PROC TIMEPLOT*

```
                Weekly Appliance Sales for the First Quarter                1

                Seller :Kreitz  Seller :LeGrange
   Month   Week           Stove              Stove      min                    max
                                                        $184.24          $2,910.37
                                                       *------------------------*
January       1    $1,312.61          $728.13           |     L    K             |
January       2      $222.35          $184.24           |!                       |
January       3    $2,263.33          $267.35           | L                 K    |
January       4    $1,787.45          $274.51           | L             K        |
February      1    $2,910.37          $397.98           |  L                    K|
February      2      $819.69        $2,242.24           |      K            L    |
                                                       *------------------------*
```

```
                Weekly Appliance Sales for the First Quarter                2

                      Kreitz          LeGrange
   Month   Week       Icebox            Icebox      min                    max
                                                    $2,520.04        $3,550.43
                                                   *------------------------*
January       1    $3,450.94        $2,520.04       |L                      K |
January       2    $3,240.67        $2,675.42       |    L            K       |
January       3    $3,160.45        $2,805.35       |       L        K        |
January       4    $3,400.24        $2,870.61       |         L           K   |
February      1    $3,550.43        $2,730.09       |     L                  K|
February      2    $3,385.74        $2,670.93       |    L                K   |
                                                   *------------------------*
```

Syntax: TIMEPLOT Procedure

Requirement: At least one PLOT statement is required.

Tips: Supports the Output Delivery System. See Chapter 3, "Output Delivery System: Basic Concepts," in *SAS Output Delivery System: User's Guide* in the *SAS Output Delivery System: User's Guide* for details.

You can use the ATTRIB, FORMAT, LABEL, and WHERE statements. See "Statements with the Same Function in Multiple Procedures" on page 35 for details. You can also use any global statements. See "Global Statements" on page 20 for a list.

PROC TIMEPLOT <*option(s)*>;
  BY <DESCENDING> *variable-1*
    <…<DESCENDING>*variable-n*>
    <NOTSORTED>;
  CLASS *variable(s)*;
  ID *variable(s)*;
  PLOT *plot-request(s)*/*option(s)*;

Statement	Task	Example
"PROC TIMEPLOT Statement"	Request that the plots be produced	Ex. 4
"BY Statement"	Produce a separate plot for each BY group	
"CLASS Statement"	Group data according to the values of the class variables	Ex. 5
"ID Statement"	Print in the listing the values of the variables that you identify	Ex. 1, Ex. 2, Ex. 3
"PLOT Statement"	Specify the plots to produce	Ex. 1, Ex. 2, Ex. 3, Ex. 4, Ex. 5

PROC TIMEPLOT Statement

Requests that the plots be produced.

Example: "Example 4: Superimposing Two Plots" on page 1626

Syntax

PROC TIMEPLOT <*option(s)*>;

Optional Arguments

DATA=*SAS-data-set*
identifies the input data set.

MAXDEC=*number*
specifies the maximum number of decimal places to print in the listing.

Default: 2

Range: 0-12

Interaction: A decimal specification in a format overrides a MAXDEC= specification.

Example: "Example 4: Superimposing Two Plots" on page 1626

SPLIT='*split-character*'
specifies a split character, which controls line breaks in column headings. It also specifies that labels be used as column headings. PROC TIMEPLOT breaks a column heading when it reaches the split character and continues the heading on the next line. Unless the split character is a blank, it is not part of the column heading. Each occurrence of the split character counts toward the 256-character maximum for a label.

Alias: S=

Default: blank (' ')

Note: Column headings can occupy up to three lines. If the column label can be split into more lines than this fixed number, then the split character is used only as a recommendation on how to split the label.

UNIFORM
uniformly scales the horizontal axis across all BY groups. By default, PROC TIMEPLOT separately determines the scale of the axis for each BY group.

Interaction: UNIFORM also affects the calculation of means for reference lines (see "REF=reference-value(s) " on page 1617).

BY Statement

Produces a separate plot for each BY group.

See: "BY" on page 36

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. Unless you use the NOTSORTED option in the BY statement, the data must either be sorted by variables or indexed. These variables are called *BY variables*.

Optional Arguments

DESCENDING
specifies that the data set is 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 data is 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 that have the same values for the BY variables are not contiguous, then the procedure treats each contiguous set as a separate BY group.

CLASS Statement

Groups data according to the values of the class variables.

Tip: PROC TIMEPLOT uses the formatted values of the CLASS variables to form classes. Thus, if a format groups the values, then the procedure uses those groups.

Example: "Example 5: Showing Multiple Observations on One Line of a Plot" on page 1629

Syntax

CLASS *variable(s)*;

Required Argument

variable(s)

specifies one or more variables that the procedure uses to group the data. Variables in a CLASS statement are called *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.

The values of the class variables appear in the listing. PROC TIMEPLOT prints and plots one line each time the combination of values of the class variables changes. Therefore, the output typically is more meaningful if you sort or group the data according to values of the class variables.

Details

Using Multiple CLASS Statements

You can use any number of CLASS statements. If you use more than one CLASS statement, then PROC TIMEPLOT simply concatenates all variables from all of the CLASS statements. The following form of the CLASS statement includes three variables:

CLASS *variable-1 variable-2 variable-3*;

It has the same effect as this form:

CLASS *variable-1*;
CLASS *variable-2*;
CLASS *variable-3*;

Using a Symbol Variable

Normally, you use the CLASS statement with a symbol variable. (See the discussion of plot requests on page 1614.) In this case, the listing of the plot variable contains a column for each value of the symbol variable. Each row of the plot contains a point for each value of the symbol variable. The plotting symbol is the first character of the formatted value of the symbol variable. If more than one observation within a class has the same value of a symbol variable, then PROC TIMEPLOT plots and prints only the first occurrence of that value and writes a warning message to the SAS log.

ID Statement

Prints in the listing the values of the variables that you identify.

Examples: "Example 1: Plotting a Single Variable" on page 1619
"Example 2: Customizing an Axis and a Plotting Symbol" on page 1621

"Example 3: Using a Variable for a Plotting Symbol" on page 1624

Syntax

ID *variable(s)*;

Required Argument

variable(s)
identifies one or more *ID variables* to print in the listing.

PLOT Statement

Specifies the plots to produce.

Tip: Each PLOT statement produces a separate plot.

Examples:
"Example 1: Plotting a Single Variable" on page 1619
"Example 2: Customizing an Axis and a Plotting Symbol" on page 1621
"Example 3: Using a Variable for a Plotting Symbol" on page 1624
"Example 4: Superimposing Two Plots" on page 1626
"Example 5: Showing Multiple Observations on One Line of a Plot" on page 1629

Syntax

PLOT *plot-request(s)* </*option(s)*>;

Summary of Optional Arguments

Control the appearance of the plot

HILOC
connects the leftmost plotting symbol to the rightmost plotting symbol with a line of hyphens (-).

JOINREF
connects the leftmost and rightmost symbols on each line of the plot with a line of hyphens (-) regardless of whether the symbols are reference symbols or plotting symbols.

NOSYMNAME
suppresses the name of the symbol variable in column headings when you use a CLASS statement.

NPP
suppresses the listing of the values of the variables that appear in the PLOT statement.

POS=*print-positions-for-plot*
specifies the number of print positions to use for the horizontal axis.

Create and customize a reference line

REF=*reference-value(s)*
draws lines on the plot that are perpendicular to the specified values on the horizontal axis.

REFCHAR='*character*'
specifies the character for drawing reference lines.

Customize the axis

AXIS=*axis-specification*
specifies the range of values to plot on the horizontal axis, as well as the interval represented by each print position on the horizontal axis.

REVERSE
orders the values on the horizontal axis with the largest value in the leftmost position.

Display multiple plots on the same set of axes

OVERLAY
plots all requests in one PLOT statement on one set of axes.

OVPCHAR='*character*'
specifies the character to print if multiple plotting symbols coincide.

Required Argument

plot-request(s)
specifies the variable or variables to plot. (Optional) Also specifies the plotting symbol to use. By default, each plot request produces a separate plot.

A plot request can have the following forms. You can mix different forms of requests in one PLOT statement (see "Example 4: Superimposing Two Plots" on page 1626).

variable(s)
identifies one or more numeric variables to plot. PROC TIMEPLOT uses the first character of the variable name as the plotting symbol.

Example: "Example 1: Plotting a Single Variable" on page 1619

(*variable(s)*)='*plotting-symbol*'
identifies one or more numeric variables to plot and specifies the plotting symbol to use for all variables in the list. You can omit the parentheses if you use only one variable.

Example: "Example 2: Customizing an Axis and a Plotting Symbol" on page 1621

(*variable(s)*)=*symbol-variable*
identifies one or more numeric variables to plot and specifies a *symbol variable*. PROC TIMEPLOT uses the first nonblank character of the formatted value of the symbol variable as the plotting symbol for all variables in the list. The plotting symbol changes from one observation to the next if the value of the symbol variable changes. You can omit the parentheses if you use only one variable.

Example: "Example 3: Using a Variable for a Plotting Symbol" on page 1624

Optional Arguments

AXIS=*axis-specification*
specifies the range of values to plot on the horizontal axis, as well as the interval represented by each print position on the axis. PROC TIMEPLOT labels the first and last ends of the axis, if space permits.

For numeric values, *axis-specification* can be one of the following or a combination of both:

- *n*<. . .*n*>
- *n* **TO** *n* <BY*increment*>

The values must be in either ascending or descending order. Use a negative value for *increment* to specify descending order. The specified values are spaced evenly along the horizontal axis even if the values are not uniformly distributed. Numeric values can be specified in the following ways:

Specification	Comments
axis=1 2 10	Values are 1, 2, and 10.
axis=10 to 100 by 5	Values appear in increments of 5, starting at 10 and ending at 100.
axis=12 10 to 100 by 5	A combination of the two previous forms of specification.

For axis variables that contain datetime values, *axis-specification* is either an explicit list of values or a starting and an ending value with an increment specified:

- '*date-time-value*'*i* <. . . '*date-time-value*'*i*>
- '*date-time-value*'*i* **TO** '*date-time-value*'*i* <BY*increment*>

'date-time-value'i
any SAS date, time, or datetime value described for the SAS functions INTCK and INTNX. The suffix *i* is one of the following:

D date

T time

DT datetime

increment
one of the valid arguments for the INTCK or INTNX functions. For dates, *increment* can be one of the following:

- DAY
- WEEK
- MONTH
- QTR
- YEAR

For datetimes, *increment* can be one of the following:

- DTDAY
- DTWEEK
- DTMONTH
- DTQTR
- DTYEAR

For times, *increment* can be one of the following:

- HOUR
- MINUTE
- SECOND

For example,

```
axis='01JAN95'd to '01JAN96'd by month
axis='01JAN95'd to '01JAN96'd by qtr
```

For descriptions of individual intervals, see the chapter on dates, times, and intervals in *SAS Language Reference: Concepts*.

Note: You must use a FORMAT statement to print the tick-mark values in an understandable form.

Interaction: The value of POS= (see "POS=print-positions-for-plot" on page 1617) overrides an interval set with AXIS=.

Tip: It is possible for your data to be out of range and fall outside the axis area of the plot. When this happens, PROC TIMEPLOT places angle brackets, (<) or (>), on the sides of the plot to indicate that there is data not being represented.

Example: "Example 2: Customizing an Axis and a Plotting Symbol" on page 1621

HILOC
connects the leftmost plotting symbol to the rightmost plotting symbol with a line of hyphens (-).

Interaction: If you specify JOINREF, then PROC TIMEPLOT ignores HILOC.

JOINREF
connects the leftmost and rightmost symbols on each line of the plot with a line of hyphens (-), regardless of whether the symbols are reference symbols or plotting symbols. However, if a line contains only reference symbols, then PROC TIMEPLOT does not connect the symbols.

Example: "Example 3: Using a Variable for a Plotting Symbol" on page 1624

NOSYMNAME
suppresses the name of the symbol variable in column headings when you use a CLASS statement. If you use NOSYMNAME, then only the value of the symbol variable appears in the column heading.

Example: "Example 5: Showing Multiple Observations on One Line of a Plot" on page 1629

NPP
suppresses the listing of the values of the variables that appear in the PLOT statement.

Example: "Example 3: Using a Variable for a Plotting Symbol" on page 1624

OVERLAY
plots all requests in one PLOT statement on one set of axes. Otherwise, PROC TIMEPLOT produces a separate plot for each plot request.

Example: "Example 4: Superimposing Two Plots" on page 1626

OVPCHAR=*'character'*
specifies the character to print if multiple plotting symbols coincide. If a plotting symbol and a character in a reference line coincide, then PROC TIMEPLOT prints the plotting symbol.

Default: at sign (@)

Example: "Example 5: Showing Multiple Observations on One Line of a Plot" on page 1629

POS=*print-positions-for-plot*
specifies the number of print positions to use for the horizontal axis.

Default: If you omit both POS= and AXIS=, then PROC TIMEPLOT initially assumes that POS=20. However, if space permits, then this value increases so that the plot fills the available space.

Interaction: If you specify POS=0 and AXIS=, then the plot fills the available space. POS= overrides an interval set with AXIS= . See the discussion of "AXIS=axis-specification" on page 1614.

See: "Page Layout" on page 1618

Example: "Example 1: Plotting a Single Variable" on page 1619

REF=*reference-value(s)*
draws lines on the plot that are perpendicular to the specified values on the horizontal axis. The values for *reference-value(s)* can be constants, or you can use the form

MEAN(*variable(s)*)

If you use this form of REF=, then PROC TIMEPLOT evaluates the mean for each variable that you list and draws a reference line for each mean.

Interactions:
If you use the UNIFORM option in the PROC TIMEPLOT statement, then the procedure calculates the mean values for the variables over all observations for all BY groups. If you do not use UNIFORM, then the procedure calculates the mean for each variable for each BY group.

If a plotting symbol and a reference character coincide, then PROC TIMEPLOT prints the plotting symbol.

Examples:
"Example 3: Using a Variable for a Plotting Symbol" on page 1624

"Example 4: Superimposing Two Plots" on page 1626

REFCHAR=*'character'*
specifies the character for drawing reference lines.

Default: vertical bar (|)

Interaction: If you are using the JOINREF or HILOC option, then do not specify a value for REFCHAR= that is the same as a plotting symbol. If you do this then PROC TIMEPLOT will interpret the plotting symbols as reference characters and will not connect the symbols as you expect.

Example: "Example 3: Using a Variable for a Plotting Symbol" on page 1624

REVERSE
orders the values on the horizontal axis with the largest value in the leftmost position.

Example: "Example 4: Superimposing Two Plots" on page 1626

Results: TIMEPLOT Procedure

Data Considerations

The input data set usually contains a date variable to use as either a class or an ID variable. Although PROC TIMEPLOT does not require an input data set sorted by date, the output is usually more meaningful if the observations are in chronological order. In addition, if you use a CLASS statement, then the output is more meaningful if the input data set groups observations according to combinations of class variable values. (For more information, see "CLASS Statement" on page 1611.)

Procedure Output

Page Layout

For each plot request, PROC TIMEPLOT prints a listing and a plot. PROC TIMEPLOT determines the arrangement of the page as follows:

- If you use POS=, then the procedure does the following:
  - determines the size of the plot from the POS= value
  - determines the space for the listing from the width of the columns of printed values, equally spaced and with a maximum of five positions between columns
  - centers the output on the page
- If you omit POS=, then the procedure does the following:
  - determines the width of the plot from the value of the AXIS= option
  - expands the listing to fill the rest of the page

If there is not enough space to print the listing and the plot, then PROC TIMEPLOT produces no output and writes the following error message to the SAS log:

```
ERROR:  Too many variables/symbol values
        to print.
```

The error does not affect other plot requests.

Contents of the Listing

The listing in the output contains different information depending on whether you use a CLASS statement. If you do not use a CLASS statement (see "Example 1: Plotting a Single Variable" on page 1619), then PROC TIMEPLOT prints (and plots) each observation on a separate line. If you do use a CLASS statement, then the form of the output varies depending on whether you specify a symbol variable (see "Using a Symbol Variable" on page 1612).

ODS Table Names

The TIMEPLOT 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 ADD LINK TO ODS TOPIC HERE in *SAS Output Delivery System: User's Guide*.

Table 54.1 *ODS Tables Produced by the TIMEPLOT Procedure*

Table Name	Description	Conditions When Table Is Generated
Plot	A single plot	If you do not specify the OVERLAY option
OverlaidPlot	Two or more plots on a single set of axes	If you specify the OVERLAY option

Missing Values

Four types of variables can appear in the listing from PROC TIMEPLOT: plot variables, ID variables, class variables, and symbol variables (as part of some column headings). Plot variables and symbol variables can also appear in the plot.

Observations with missing values of a class variable form a class of observations.

In the listing, missing values appear as a period (.), a blank, or a special missing value (the letters A through Z and the underscore (_) character).

In the plot, PROC TIMEPLOT handles different variables in different ways:

- An observation or class of observations with a missing value of the plot variable does not appear in the plot.
- If you use a symbol variable (see the discussion of plot requests on page 1614), then PROC TIMEPLOT uses a period (.) as the symbol variable on the plot for all observations that have a missing value for the symbol variable.

Examples: TIMEPLOT Procedure

Example 1: Plotting a Single Variable

Features: ID statement

PLOT statement arguments:
simple plot request
POS=

Details

This example demonstrates the following:

- Use a single PLOT statement to plot sales of refrigerators
- Specify the number of print positions to use for the horizontal axis of the plot

- Provide context for the points in the plot by printing in the listing the values of two variables that are not in the plot

Program

```
options formchar="|----|+|---+=|-/\<>*";

data sales;
   input Month Week Seller $ Icebox Stove;
   datalines;
1 1 Kreitz   3450.94 1312.61
1 1 LeGrange 2520.04  728.13
1 2 Kreitz   3240.67  222.35
1 2 LeGrange 2675.42  184.24
1 3 Kreitz   3160.45 2263.33
1 3 LeGrange 2805.35  267.35
1 4 Kreitz   3400.24 1787.45
1 4 LeGrange 2870.61  274.51
2 1 Kreitz   3550.43 2910.37
2 1 LeGrange 2730.09  397.98
2 2 Kreitz   3385.74  819.69
2 2 LeGrange 2670.93 2242.24
;

proc timeplot data=sales;
   plot icebox / pos=50;

   id month week;

   title 'Weekly Sales of Iceboxes';
   title2 'for the';
   title3 'First Six Weeks of the Year';
run;
```

Program Description

Set the FORMCHAR option. Setting FORMCHAR to this exact string renders better HTML output when it is viewed outside of the SAS environment where SAS Monospace fonts are not available.

```
options formchar="|----|+|---+=|-/\<>*";
```

Create the SALES data set. SALES contains weekly information about the sales of refrigerators and stoves by two sales representatives.

```
data sales;
   input Month Week Seller $ Icebox Stove;
   datalines;
1 1 Kreitz   3450.94 1312.61
1 1 LeGrange 2520.04  728.13
1 2 Kreitz   3240.67  222.35
1 2 LeGrange 2675.42  184.24
1 3 Kreitz   3160.45 2263.33
1 3 LeGrange 2805.35  267.35
1 4 Kreitz   3400.24 1787.45
1 4 LeGrange 2870.61  274.51
2 1 Kreitz   3550.43 2910.37
2 1 LeGrange 2730.09  397.98
```

```
2 2 Kreitz   3385.74  819.69
2 2 LeGrange 2670.93 2242.24
;
```

Plot sales of refrigerators. The plot variable, Icebox, appears in both the listing and the output. POS= provides 50 print positions for the horizontal axis.

```
proc timeplot data=sales;
   plot icebox / pos=50;
```

Label the rows in the listing. The values of the ID variables, Month and Week, are used to uniquely identify each row of the listing.

```
   id month week;
```

Specify the titles.

```
   title 'Weekly Sales of Iceboxes';
   title2 'for the';
   title3 'First Six Weeks of the Year';
run;
```

Output

The column headings in the listing are the variables' names. The plot uses the default plotting symbol, which is the first character of the plot variable's name.

Output 54.3 *Plot for a Single Variable*

Weekly Sales of Iceboxes
for the
First Six Weeks of the Year

Month	Week	Icebox	min 2520.04 ... max 3550.43
1	1	3450.94	I
1	1	2520.04	I
1	2	3240.67	I
1	2	2675.42	I
1	3	3160.45	I
1	3	2805.35	I
1	4	3400.24	I
1	4	2870.61	I
2	1	3550.43	I
2	1	2730.09	I
2	2	3385.74	I
2	2	2670.93	I

Example 2: Customizing an Axis and a Plotting Symbol

Features: ID statement

PLOT statement arguments:
using a plotting symbol

AXIS=

Other features: LABEL statement
PROC FORMAT
SAS system options:
FMTSEARCH=

Data set: SALES

Details

This example demonstrates the following:

- Specify the character to use as the plotting symbol
- Specify the minimum and maximum values for the horizontal axis as well as the interval represented by each print position
- Provide context for the points in the plot by printing in the listing the values of two variables that are not in the plot
- Use a variable's label as a column heading in the listing
- Create and uses a permanent format

Program

```
libname proclib
'SAS-library';

options formchar="|----|+|---+=|-/\<>*" fmtsearch=(proclib);

proc format library=proclib;
   value monthfmt 1='January'
                  2='February';
run;

proc timeplot data=sales;
   plot icebox='R' / axis=2500 to 3600 by 25;

   id month week;

   label icebox='Refrigerator';


   format month monthfmt.;

   title 'Weekly Sales of Refrigerators';
   title2 'for the';
   title3 'First Six Weeks of the Year';
run;
```

Program Description

Declare the PROCLIB SAS library.

```
libname proclib
'SAS-library';
```

Set the SAS system options. Setting FORMCHAR to this exact string renders better HTML output when it is viewed outside of the SAS environment where SAS Monospace

fonts are not available. FMTSEARCH= adds the SAS library PROCLIB to the search path that is used to locate formats.

```
options formchar="|----|+|---+=|-/\<>*" fmtsearch=(proclib);
```

Create a format for the Month variable. PROC FORMAT creates a permanent format for Month. The LIBRARY= option specifies a permanent storage location so that the formats are available in subsequent SAS sessions. This format is used for examples throughout this chapter.

```
proc format library=proclib;
   value monthfmt 1='January'
                  2='February';
run;
```

Plot sales of refrigerators. The plot variable, Icebox, appears in both the listing and the output. The plotting symbol is 'R'. AXIS= sets the minimum value of the axis to 2500 and the maximum value to 3600. BY 25 specifies that each print position on the axis represents 25 units (in this case, dollars).

```
proc timeplot data=sales;
   plot icebox='R' / axis=2500 to 3600 by 25;
```

Label the rows in the listing. The values of the ID variables, Month and Week, are used to uniquely identify each row of the listing.

```
   id month week;
```

Apply a label to the sales column in the listing. The LABEL statement associates a label with the variable Icebox for the duration of the PROC TIMEPLOT step. PROC TIMEPLOT uses the label as the column heading in the listing.

```
   label icebox='Refrigerator';
```

Apply the MONTHFMT. format to the Month variable. The FORMAT statement assigns a format to use for Month in the report.

```
   format month monthfmt.;
```

Specify the titles.

```
   title 'Weekly Sales of Refrigerators';
   title2 'for the';
   title3 'First Six Weeks of the Year';
run;
```

Output

The column headings in the listing are the variables' names (for Month and Week, which have no labels) and the variable's label (for Icebox, which has a label). The plotting symbol is **R** (for Refrigerator).

Output 54.4 *Plot with Customized Axis and Plotting Symbol*

Weekly Sales of Refrigerators
for the
First Six Weeks of the Year

Month	Week	Refrigerator	min 2500 ... max 3600
January	1	3450.94	R
January	1	2520.04	R
January	2	3240.67	R
January	2	2675.42	R
January	3	3160.45	R
January	3	2805.35	R
January	4	3400.24	R
January	4	2870.61	R
February	1	3550.43	R
February	1	2730.09	R
February	2	3385.74	R
February	2	2670.93	R

Example 3: Using a Variable for a Plotting Symbol

Features: ID statement

PLOT statement arguments:
- using a variable as the plotting symbol
- JOINREF
- NPP
- REF=
- REFCHAR=

Data set: SALES

Format: MONTHFMT.

Details

This example demonstrates the following:

- Specify a variable to use as the plotting symbol to distinguish between points for each of two sales representatives
- Suppress the printing of the values of the plot variable in the listing
- Draw a reference line to a specified value on the axis and specifies the character to use to draw the line

- Connect the leftmost and rightmost symbols on each line of the plot

Program

```
libname proclib
'SAS-library';

options formchar="|----|+|---+=|-/\<>*" fmtsearch=(proclib);

proc timeplot data=sales;
   plot stove=seller /

                     npp

                     ref=1500 refchar=':'

                     joinref

                     axis=100 to 3000 by 50;

   id month week;

   format month monthfmt.;

   title 'Weekly Sales of Stoves';
   title2 'Compared to Target Sales of $1500';
   title3 'K for Kreitz; L for LeGrange';
run;
```

Program Description

Declare the PROCLIB SAS library.

```
libname proclib
'SAS-library';
```

Set the SAS system options. Setting FORMCHAR to this exact string renders better HTML output when it is viewed outside of the SAS environment where SAS Monospace fonts are not available. FMTSEARCH= adds the SAS library PROCLIB to the search path that is used to locate formats.

```
options formchar="|----|+|---+=|-/\<>*" fmtsearch=(proclib);
```

Plot sales of stoves. The PLOT statement specifies both the plotting variable, Stove, and a symbol variable, Seller. The plotting symbol is the first letter of the formatted value of the Seller (in this case, **L** or **K**).

```
proc timeplot data=sales;
   plot stove=seller /
```

Suppress the appearance of the plotting variable in the listing. The values of the Stove variable will not appear in the listing.

```
                     npp
```

Create a reference line on the plot. REF= and REFCHAR= draw a line of colons at the sales target of $1500.

```
                     ref=1500 refchar=':'
```

Draw a line between the symbols on each line of the plot. In this plot, JOINREF connects each plotting symbol to the reference line.

```
joinref
```

Customize the horizontal axis. AXIS= sets the minimum value of the horizontal axis to 100 and the maximum value to 3000. BY 50 specifies that each print position on the axis represents 50 units (in this case, dollars).

```
axis=100 to 3000 by 50;
```

Label the rows in the listing. The values of the ID variables, Month and Week, are used to identify each row of the listing.

```
id month week;
```

Apply the MONTHFMT. format to the Month variable. The FORMAT statement assigns a format to use for Month in the report.

```
format month monthfmt.;
```

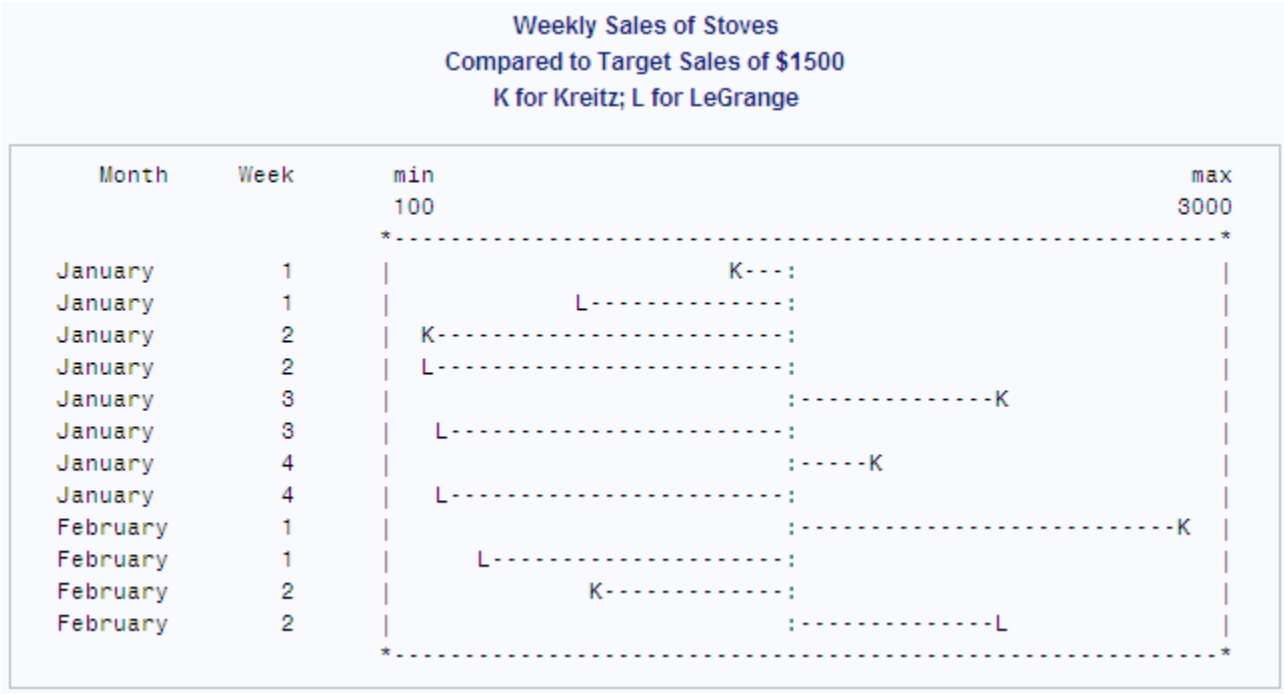
Specify the titles.

```
title 'Weekly Sales of Stoves';
title2 'Compared to Target Sales of $1500';
title3 'K for Kreitz; L for LeGrange';
run;
```

Output

The plot uses the first letter of the value of Seller as the plotting symbol.

Output 54.5 *Plot with Variable for Plotting Symbol*



Example 4: Superimposing Two Plots

Features: PROC TIMEPLOT statement options:

MAXDEC=

PLOT statement arguments:
using two types of plot requests
OVERLAY
REF=MEAN(*variable(s)*)
REVERSE

Data set: SALES

Details

This example demonstrates the following:

- Superimpose two plots on one set of axes
- Specify a variable to use as the plotting symbol for one plot and a character to use as the plotting symbol for the other plot
- Draw a reference line to the mean value of each of the two variables plotted
- Reverse the labeling of the axis so that the largest value is at the far left of the plot

Program

```
options formchar="|----|+|---+=|-/\<>*";

proc timeplot data=sales maxdec=0;

   plot stove=seller icebox='R' /

          overlay

            ref=mean(stove icebox)


reverse;

   label icebox='Refrigerators';

   title 'Weekly Sales of Stoves and Refrigerators';
   title2 'for the';
   title3 'First Six Weeks of the Year';
run;
```

Program Description

Set the FORMCHAR option. Setting FORMCHAR to this exact string renders better HTML output when it is viewed outside of the SAS environment where SAS Monospace fonts are not available.

```
options formchar="|----|+|---+=|-/\<>*";
```

Specify the number of decimal places to display. MAXDEC= specifies the number of decimal places to display in the listing.

```
proc timeplot data=sales maxdec=0;
```

Plot sales of both stoves and refrigerators. The PLOT statement requests two plots. One plot uses the first letter of the formatted value of Seller to plot the values of Stove. The other uses the letter **R** (to match the label Refrigerators) to plot the value of Icebox.

```
plot stove=seller icebox='R' /
```

Print both plots on the same set of axes.

```
overlay
```

Create two reference lines on the plot. REF= draws two reference lines: one perpendicular to the mean of Stove, the other perpendicular to the mean of Icebox.

```
ref=mean(stove icebox)
```

Order the values on the horizontal axis from largest to smallest.

```
reverse;
```

Apply a label to the sales column in the listing. The LABEL statement associates a label with the variable Icebox for the duration of the PROC TIMEPLOT step. PROC TIMEPLOT uses the label as the column heading in the listing.

```
label icebox='Refrigerators';
```

Specify the titles.

```
title 'Weekly Sales of Stoves and Refrigerators';
title2 'for the';
title3 'First Six Weeks of the Year';
run;
```

Output

The column heading for the variable Icebox in the listing is the variable's label (Refrigerators). One plot uses the first letter of the value of Seller as the plotting symbol. The other plot uses the letter **R**.

Output 54.6 *Two Plots Superimposed Using Different Plotting Symbols*

Weekly Sales of Stoves and Refrigerators
for the
First Six Weeks of the Year

```
Stove     Refrigerators      max                                          min
                             3550.43                                   184.24
                            *-------------------------------------------*
 1313              3451     |R      |                      K  |          |
  728              2520     |       |       R              |    L       |
  222              3241     |   R   |                      |          K |
  184              2675     |       |    R                 |           L|
 2263              3160     |    R  |          K           |            |
  267              2805     |       |  R                   |          L |
 1787              3400     | R     |                K     |            |
  275              2871     |       | R                    |          L |
 2910              3550     |R      | K                    |            |
  398              2730     |       |   R                  |        L   |
  820              3386     | R     |                      |   K        |
 2242              2671     |       |    R    L            |            |
                            *-------------------------------------------*
```

Example 5: Showing Multiple Observations on One Line of a Plot

Features: CLASS statement

PLOT statement arguments:
- creating multiple plots
- NOSYMNAME
- OVPCHAR=

Data set: SALES

Format: MONTHFMT.

Details

This example demonstrates the following:

- Group observations for the same month and week so that sales for the two sales representatives for the same week appear on the same line of the plot

- Specify a variable to use as the plotting symbol
- Suppress the name of the plotting variable from one plot
- Specify a size for the plots so that they both occupy the same amount of space

Program

```
libname proclib
'SAS-library';

options formchar="|----|+|---+=|-/\<>*" fmtsearch=(proclib);

proc timeplot data=sales;
   class month week;

   plot stove=seller  / pos=25 ovpchar='!';
   plot icebox=seller / pos=25 ovpchar='!' nosymname;

   format stove icebox dollar10.2 month monthfmt.;

   title 'Weekly Appliance Sales for the First Quarter';
run;
```

Program Description

Declare the PROCLIB SAS library.

```
libname proclib
'SAS-library';
```

Set the SAS system options. Setting FORMCHAR to this exact string renders better HTML output when it is viewed outside of the SAS environment where SAS Monospace fonts are not available. FMTSEARCH= adds the SAS library PROCLIB to the search path that is used to locate formats.

```
options formchar="|----|+|---+=|-/\<>*" fmtsearch=(proclib);
```

Specify subgroups for the analysis. The CLASS statement groups all observations with the same values of Month and Week into one line in the output. Using the CLASS statement with a symbol variable produces in the listing one column of the plot variable for each value of the symbol variable.

```
proc timeplot data=sales;
   class month week;
```

Plot sales of stoves and refrigerators. Each PLOT statement produces a separate plot. The plotting symbol is the first character of the formatted value of the symbol variable: **K** for Kreitz; **L** for LeGrange. POS= specifies that each plot uses 25 print positions for the horizontal axis. OVPCHAR= designates the exclamation point as the plotting symbol when the plotting symbols coincide. NOSYMNAME suppresses the name of the symbol variable Seller from the second listing.

```
   plot stove=seller  / pos=25 ovpchar='!';
   plot icebox=seller / pos=25 ovpchar='!' nosymname;
```

Apply formats to values in the listing. The FORMAT statement assigns formats to use for Stove, Icebox, and Month in the report. The TITLE statement specifies a title.

```
   format stove icebox dollar10.2 month monthfmt.;
```

Specify the title.

```
   title 'Weekly Appliance Sales for the First Quarter';
run;
```

Output

Output 54.7 *Weekly Stove Sales by Seller*

Weekly Appliance Sales for the First Quarter

Month	Week	Kreitz Icebox	LeGrange Icebox	min $2,520.04 — max $3,550.43
January	1	$3,450.94	$2,520.04	L … K
January	2	$3,240.67	$2,675.42	L … K
January	3	$3,160.45	$2,805.35	L … K
January	4	$3,400.24	$2,870.61	L … K
February	1	$3,550.43	$2,730.09	L … K
February	2	$3,385.74	$2,670.93	L … K

Output 54.8 *Weekly Icebox Sales by Seller*

Weekly Appliance Sales for the First Quarter

Month	Week	Seller :Kreitz Stove	Seller :LeGrange Stove	min $184.24 — max $2,910.37
January	1	$1,312.61	$728.13	L K
January	2	$222.35	$184.24	!
January	3	$2,263.33	$267.35	L … K
January	4	$1,787.45	$274.51	L … K
February	1	$2,910.37	$397.98	L … K
February	2	$819.69	$2,242.24	K … L

Chapter 55

TRANSPOSE Procedure

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 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 55.1 *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 55.2 *A Transposition with BY Groups*

Input Data Set 1

Location	Date	Length 1	Width 1	Length 2	Width 2	Length 3	Width 3	Length 4	Width 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 1649.

Syntax: TRANSPOSE Procedure

Tips: Does not support the Output Delivery System

You can use the ATTRIB, FORMAT, LABEL, and WHERE statements. For more information, see "Statements with the Same Function in Multiple Procedures" on page 35. You can also use any global statement. For a list, see "Global Statements" in Chapter 1 of *SAS Statements: Reference*.

PROC TRANSPOSE <DATA=*input-data-set*> <DELIMITER=*delimiter*> <LABEL=*label*> <LET> <NAME=*name*> <OUT=*output-data-set*> <PREFIX=*prefix*> <SUFFIX=*suffix*>;
  BY <DESCENDING> *variable-1*
    <…<DESCENDING>*variable-n*>
    <NOTSORTED>;
  COPY *variable(s)*;
  ID *variable*;
    IDLABEL *variable*;
  VAR *variable(s)*;

Statement	Task	Example
"PROC TRANSPOSE Statement"	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 Statement"	Transpose each BY group	Ex. 4
"COPY Statement"	Copy variables directly without transposing them	Ex. 6
"ID Statement"	Specify a variable whose values name the transposed variables	Ex. 2
"IDLABEL Statement"	Create labels for the transposed variables	Ex. 3
"VAR Statement"	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. For more information, see "Statements with the Same Function in Multiple Procedures" on page 35. You can also use any global statement. For a list, see "Global Statements" in Chapter 1 of *SAS Statements: Reference*.

Examples: "Example 1: Performing a Simple Transposition" on page 1644
"Example 2: Naming Transposed Variables" on page 1646
"Example 3: Labeling Transposed Variables" on page 1647
"Example 4: Transposing BY Groups" on page 1649
"Example 5: Naming Transposed Variables When the ID Variable Has Duplicate Values" on page 1652
"Example 6: Transposing Data for Statistical Analysis" on page 1654

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 will be 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 1640

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 1652

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 1646

OUT= *output-data-set*
names the output data set. If *output-data-set* does not exist, then PROC TRANSPOSE creates it by using the DATA*n* naming convention.

Default: DATA*n*

See: "Example 1: Performing a Simple Transposition" on page 1644

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, …,VAR*n*.

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 1646

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.

BY Statement

Defines BY groups.

Restriction: 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.

Example: "Example 4: Transposing BY Groups" on page 1649

Syntax

BY <DESCENDING> *variable-1*
<…<DESCENDING>*variable-n*>
<NOTSORTED>;

Required Argument

variable
specifies the variable that PROC TRANSPOSE 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 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 data set is 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 data is 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. 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.

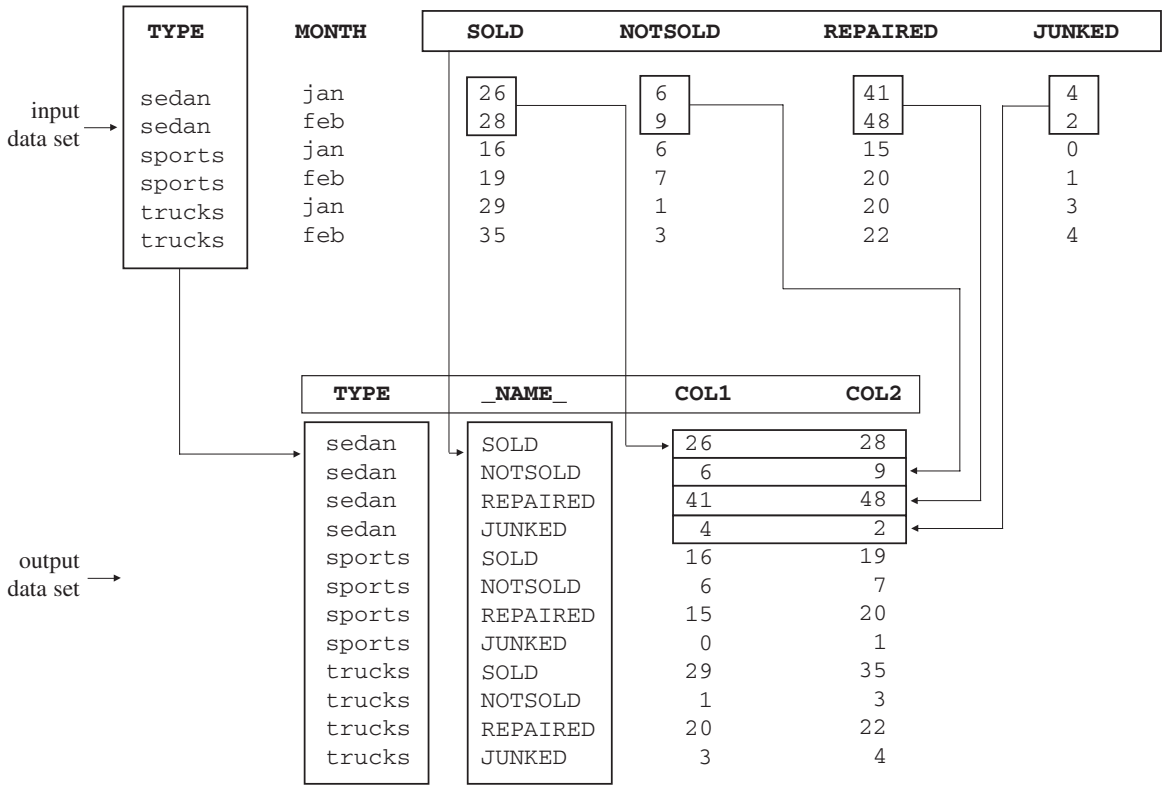
The NOBYSORTED system option disables observation sequence checking system-wide and applies to all procedures and BY statements. See the "BYSORTED System Option" in *SAS System Options: Reference*.

Details

PROC TRANSPOSE does not transpose BY groups. Instead, for each BY group, PROC TRANSPOSE creates one observation for each variable that it transposes.

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 55.1 *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.

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 1654

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.

ID Statement

Specifies one or more variables in the input data set whose 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 will be 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 1646

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 will issue a warning message about duplicate ID values and stop processing. However, if the LET option is specified in the PROC TRANSPOSE statement then the procedure will issue 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 combined with the PREFIX, DELIMITER, and SUFFIX option values can cause duplicate output data set variable names in cases where 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. 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 will be replaced with underscores or, if the name appears to be a numeric constant, an underscore will be used as a prefix and the characters +, -, and . will be changed to P, N, and D. The resulting name will be 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 will be 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 will be 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 1647

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.

VAR Statement

Lists the variables to transpose.

Examples: "Example 4: Transposing BY Groups" on page 1649
"Example 6: Transposing Data for Statistical Analysis" on page 1654

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 will be character variables.

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 DELMITER= 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, …, COL*n* 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 55.3 *Student Test Scores in Variables*

Student Test Scores in Variables 1

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 55.4 *Student Test Scores*

Student Test Scores 1

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 55.5 *Student Test Scores, IDLABEL*

```
                          Student Test Scores                              1

 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

```
proc contents data=idlabel;
run;
```

Program Description

Display the variable and label names. PROC CONTENTS will display 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 55.6 *The Contents Procedure*

```
Student Test Scores                                      2

                          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

Other features: 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=(col1=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

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 FISHDATA data set. 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.

```
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 55.7 *Fish Length Data*

Fish Length Data for Each Location and Date 1

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 55.8 *Closing Prices*

```
 Closing Prices for Horizon Kites and SkyHi Kites       1

       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 55.9 *Split Data Set*

SPLIT Data Set 1
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  'TOTSPLIT 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 55.10 *TOTSPLIT Data Set*

TOTSPLIT Data Set 1
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

Chapter 56

XSL Procedure

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 package 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 Web site **`http://www.saxonica.com/documentation9.3/about/intro.xml`**.

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 Web site **`http://www.w3.org/TR/xslt20`**.

Syntax: XSL Procedure

Example: "Example: Transforming an XML Document into Another XML Document" on page 1661

PROC XSL **IN**=*fileref* | '*external-file*'
OUT=*fileref* | '*external-file*' **XSL**=*fileref* | '*external-file*';

Statement	Task	Example
"PROC XSL Statement"	Transform an XML document	Ex. 1

PROC XSL Statement

Transforms an XML document.

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.

Example: "Example: Transforming an XML Document into Another XML Document" on page 1661

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: Transforming an XML Document into Another XML Document" on page 1661

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: Transforming an XML Document into Another XML Document" on page 1661

Example: Transforming an XML Document into Another XML Document

Details

The following 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

The following SAS program transforms the XML document. The procedure specifies the input XML document, the XSL style sheet, and the output XML document.

```
proc xsl
   in='C:\XMLInput.xml'
   xsl='C:\XSLTransform.xsl'
   out='C:\XMLOutput.xml';
run;
```

Output

Here is the resulting output XML document named XMLOutput.xml.

Output 56.1 *Output XML Document*

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
<model make="Ford">Mustang</model>
<model make="Chevrolet">Nova</model>
</root>
```

Part 3

Appendixes

Appendix 1

SAS Elementary Statistics Procedures

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 1667 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	$\Sigma\ w_i$
WDF	$\Sigma\ w_i - 1$

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 A1.1 *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
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	

Statistic	PROC MEANS and SUMMARY	PROC UNIVARIATE	PROC TABULATE	PROC REPORT	PROC CORR	PROC SQL
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	
Mode	X	X	X	X		
Percentiles/Deciles/ Quartiles	X	X	X	X		
t test						
for mean=0	X	X	X	X		X
for mean= μ_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

$$\Sigma\ w_i(x_i - \bar{x})^2$$

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

$$c_{4_n} \Sigma \; z_i^4 - \frac{3(n-1)^2}{(n-2)(n-3)}$$

where c_{4_n} is $\frac{n(n+1)}{(n-1)(n-2)(n-3)}$. The weighted kurtosis is computed as

$$= c_{4_n} \Sigma \; \left((x_i - \bar{x}) / \hat{\sigma}_i \right)^4 - \frac{3(n-1)^2}{(n-2)(n-3)}$$

$$= c_{4_n} \Sigma \; w_i^2 \left((x_i - \bar{x}) / \hat{\sigma} \right)^4 - \frac{3(n-1)^2}{(n-2)(n-3)}$$

When VARDEF=N, the kurtosis is computed as

$$= \frac{1}{n} \Sigma \; z_i^4 - 3$$

and the weighted kurtosis is computed as

$$= \frac{1}{n} \Sigma \; \left((x_i - \bar{x}) / \hat{\sigma}_i \right)^4 - 3$$

$$= \frac{1}{n} \Sigma \; w_i^2 \left((x_i - \bar{x}) / \hat{\sigma} \right)^4 - 3$$

where σ_i^2 is σ^2 / w_i. The formula is invariant under the transformation $w_i^* = z w_i, \; z > 0$. 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

$$c_{3_n} \sum z_i^3$$

where c_{3_n} is $\frac{n}{(n-1)(n-2)}$. The weighted skewness is computed as

$$= c_{3_n} \sum \left((x_i - \bar{x}) / \hat{\sigma}_j\right)^3$$

$$= c_{3_n} \sum w_i^{3/2} \left((x_i - \bar{x}) / \hat{\sigma}\right)^3$$

When VARDEF=N, the skewness is computed as

$$= \frac{1}{n} \sum z_i^3$$

and the weighted skewness is computed as

$$= \frac{1}{n} \sum \left((x_i - \bar{x}) / \hat{\sigma}_j\right)^3$$

$$= \frac{1}{n} \sum w_i^{3/2} \left((x_i - \bar{x}) / \hat{\sigma}\right)^3$$

The formula is invariant under the transformation $w_i^* = z w_i, \; z > 0$. 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

$$\Sigma\ w_i$$

USS
is the uncorrected sum of squares, computed as

$$\Sigma\ w_i x_i^2$$

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 1st percentile.

P5
is the 5th percentile.

P10
is the 10th percentile.

P90
is the 90th percentile.

P95
is the 95th percentile.

P99
is the 99th percentile.

Q1
is the lower quartile (25th percentile).

Q3
is the upper quartile (75th 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 $x_1, x_2, \ldots, x_n$ 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 tth percentile between 0 and 1, let $p = t/100$. Then define j as the integer part of np and g as the fractional part of np or $(n+1)p$, so that

$$np = j + g \qquad \text{when } QNTLDEF = 1, 2, 3, \text{ or } 5$$
$$(n+1)p = j + g \qquad \text{when } QNTLDEF = 4$$

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

$$y = \begin{cases} \frac{1}{2}\left(x_i + x_{i+1}\right) & \text{if } \sum_{j=1}^{i} w_j = pW \\ x_{i+1} & \text{if } \sum_{j=1}^{i} w_j < pW < \sum_{j=1}^{i+1} w_j \end{cases}$$

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 A1.2 *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	$y = \frac{1}{2}\left(x_j + x_{j+1}\right)$	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 μ_0 and is calculated as

$$\frac{\bar{x} - \mu_0}{s / \sqrt{\sum w_i}}$$

By default, μ_0 is equal to zero. You can use the MU0= option in the PROC UNIVARIATE statement to specify μ_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 α 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

For more information about using weights and an example, see "WEIGHT" on page 43.

Data Requirements for Summarization Procedures

The following 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, while 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 defined 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 μ, 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 pth 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 outputs 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

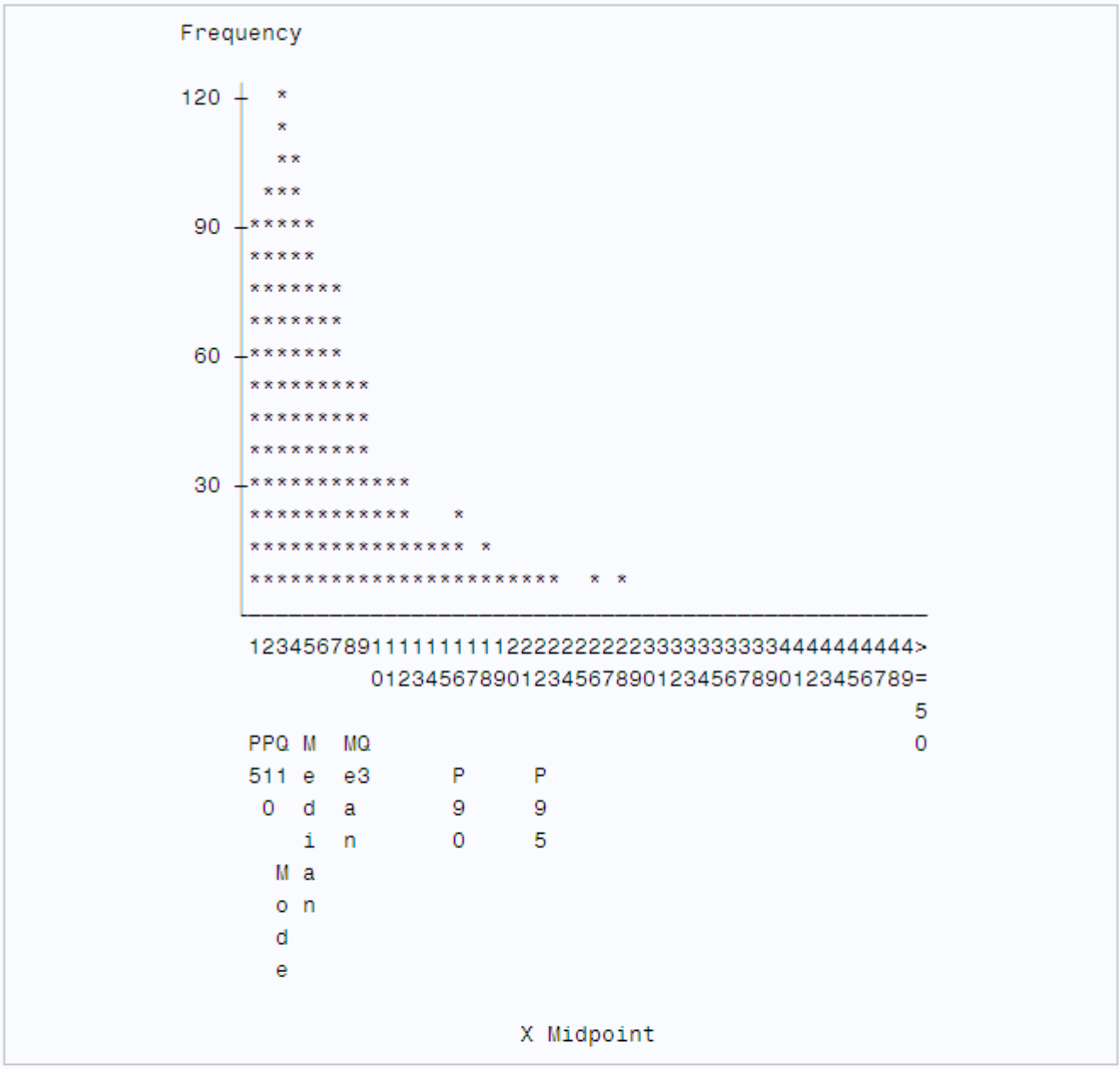
Tests for Location: Mu0=0				
Test	Statistic		p Value	
Student's t	t	32.57939	Pr > |t|	<.0001
Sign	M	494.5	Pr >= |M|	<.0001
Signed Rank	S	244777.5	Pr >= |S|	<.0001

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

Example of Quantiles and Measures of Location



P5 = 5TH PERCENTILE
P10 = 10TH PERCENTILE
P90 = 90TH PERCENTILE
P95 = 95TH PERCENTILE
Q1 = 1ST QUARTILE
Q3 = 3RD QUARTILE

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 σ^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 σ 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 σ^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. Hence 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, while large deviations are strongly emphasized. Because of the divisor σ^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

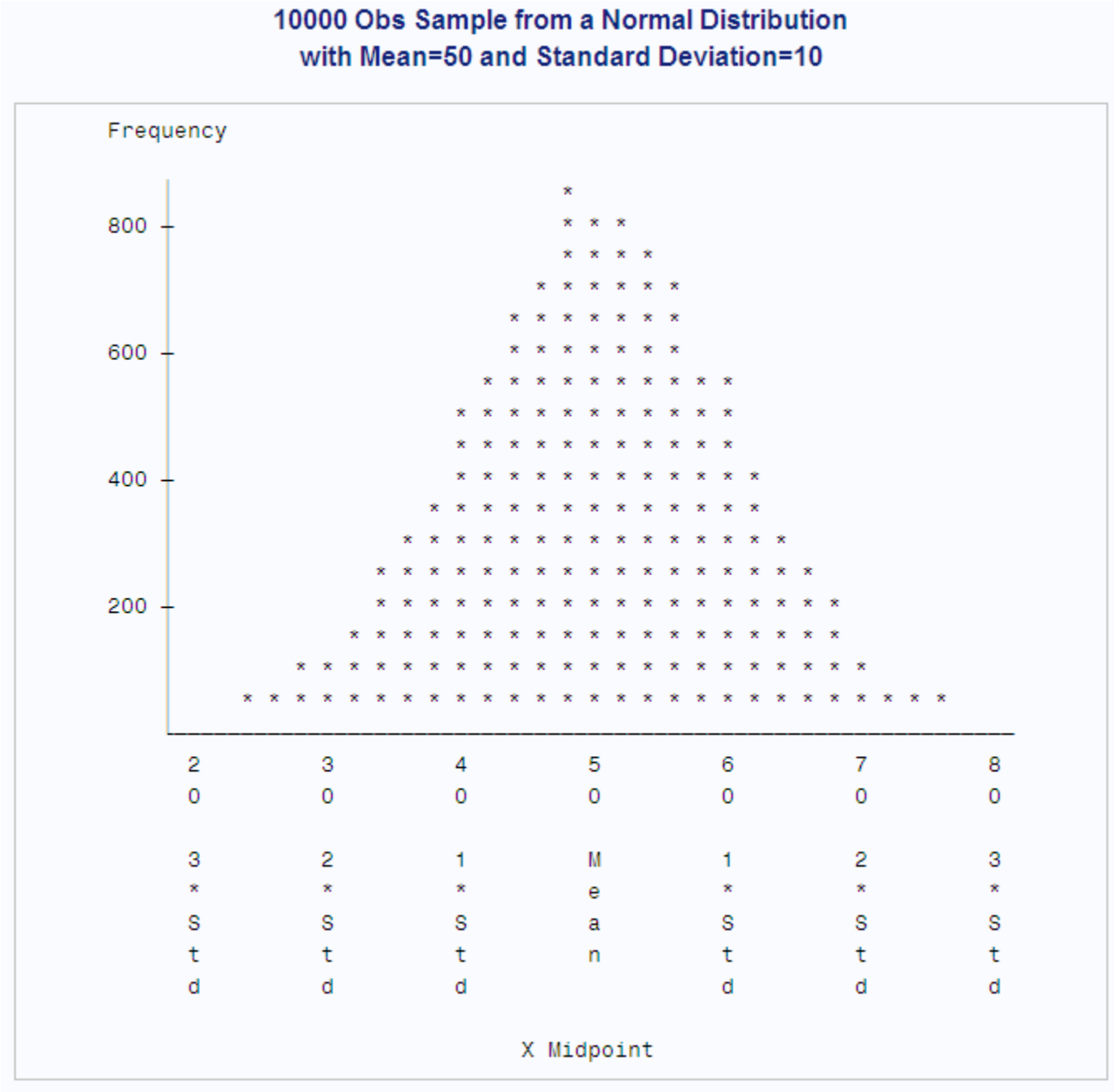
Basic Statistical Measures			
Location		Variability	
Mean	50.03237	Std Deviation	9.92014
Median	50.06492	Variance	98.40915
Mode	.	Range	76.51343
		Interquartile Range	13.28179

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



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*.

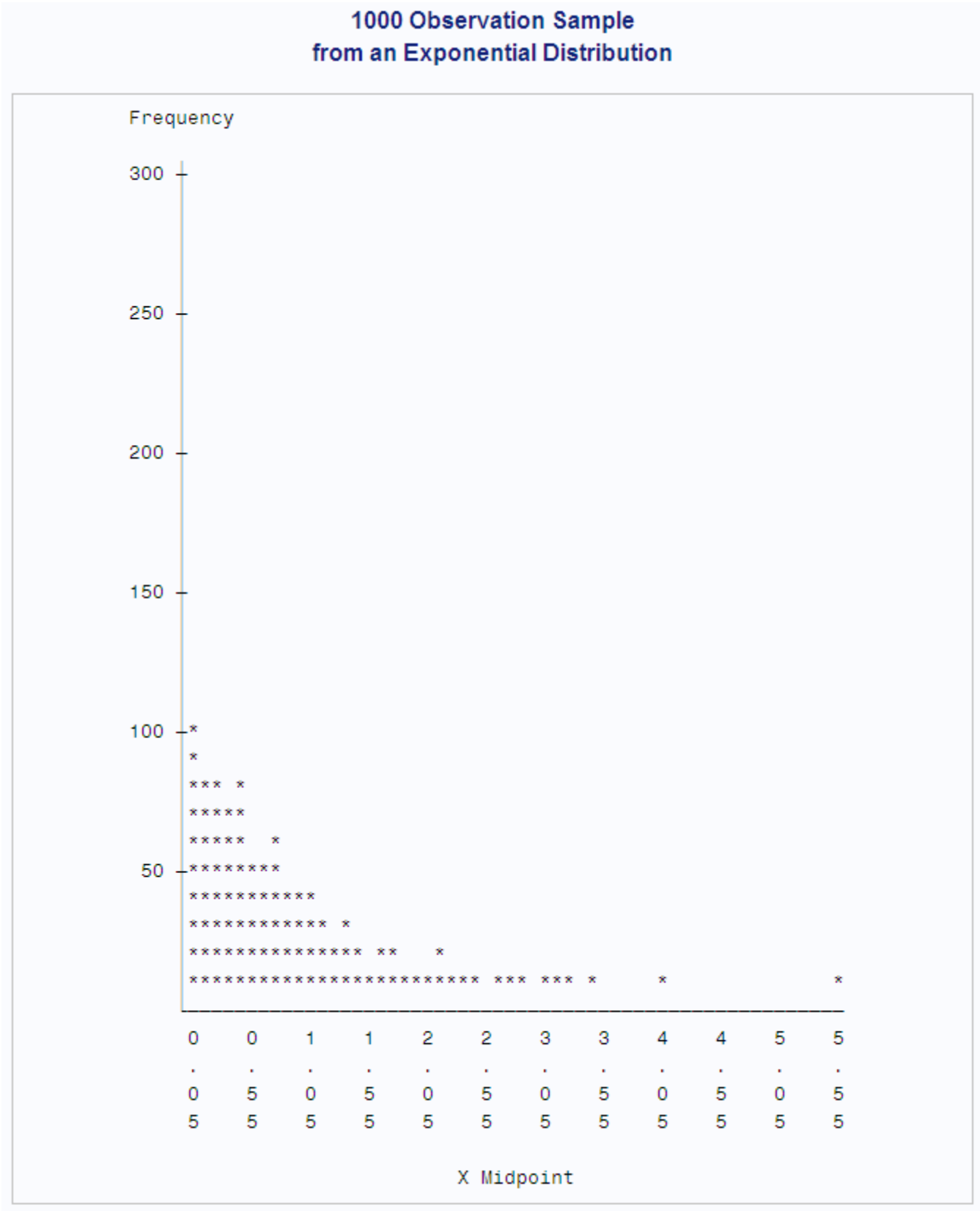
It can be proven mathematically that if the original population has mean μ and standard deviation σ, then the sampling distribution of the mean also has mean μ, but its standard deviation is $\sigma/\sqrt{n}$. The standard deviation of the sampling distribution of the mean is called the *standard error 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, while 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
Frequency
300
250
200
150
100
50
0 0 1 1 2 2 3 3 4 4 5 5
. . . .
0 5 0 5 0 5 0 5 0 5 0 5
5 5 5 5 5 5 5 5 5 5 5 5
X Midpoint

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

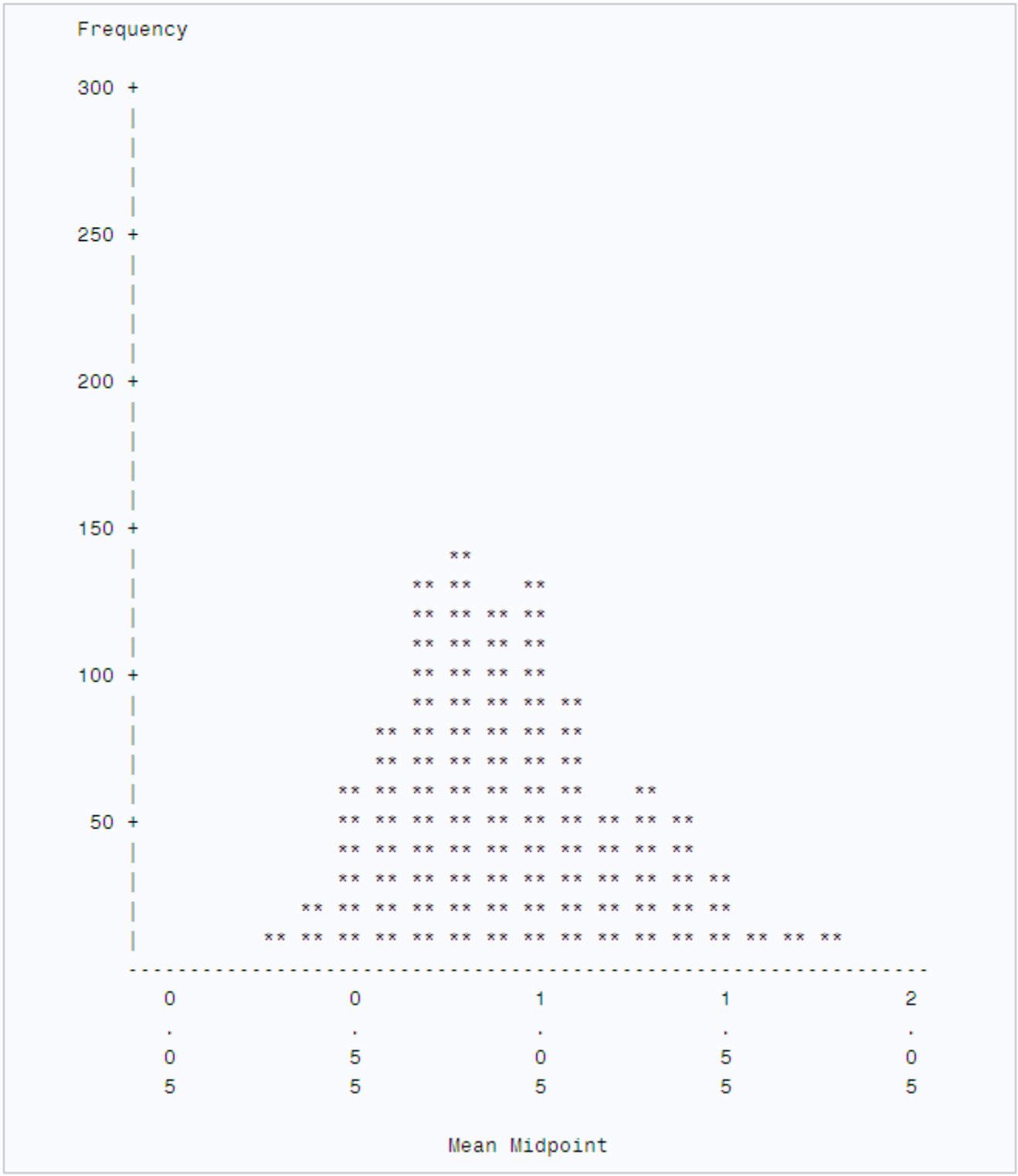
The next DATA step generates 1000 different samples from the same exponential distribution. Each sample contains ten observations. The MEANS procedure computes the mean of each sample. In the data set that is created by PROC MEANS, each observation represents the mean of a sample of ten observations 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

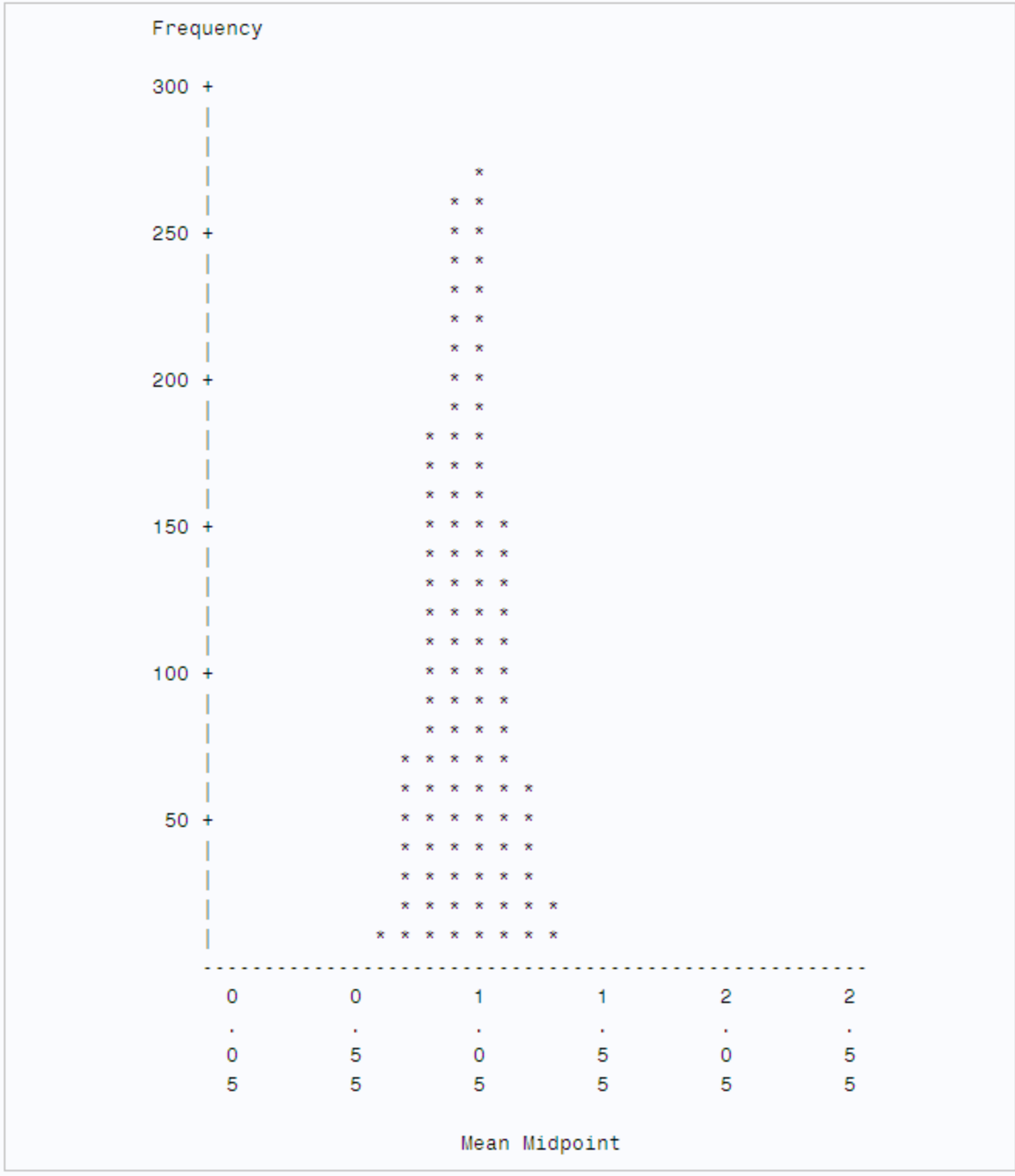
Tests for Normality				
Test	Statistic		p Value	
Shapiro-Wilk	W	0.9779	Pr < W	<0.0001
Kolmogorov-Smirnov	D	0.055498	Pr > D	<0.0100
Cramer-von Mises	W-Sq	0.953926	Pr > W-Sq	<0.0050
Anderson-Darling	A-Sq	5.945023	Pr > A-Sq	<0.0050

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
Frequency
300
250
200
150
100
50
0 0 1 1 2 2
. . . . . .
0 5 0 5 0 5
5 5 5 5 5 5
Mean Midpoint

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 μ 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 in many situations it corresponds 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 μ 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 μ. Assume temporarily that σ, 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 $-8 \le \bar{x} \le 8$, 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, and 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 α. 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%, while −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 $1 - \beta$. β 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 σ because you do not usually know the value of σ. You can, however, use *s* as an estimate of σ. 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 μ_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 elected 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., Tietjen, G.L., and Beckman, R.J. (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 Beaver, R.. (1998), *Introduction to Probability and Statistics*, 10th Edition, Belmont, CA: Wadsworth Publishing Company.
- Ott, R. and Mendenhall, W. (1994) *Understanding Statistics*, 6th Edition, North Scituate, MA: Duxbury Press.
- Schlotzhauer, S.D. and Littell, R.C. (1997), *SAS System for Elementary Statistical Analysis*, Second Edition, Cary, NC: SAS Institute Inc.
- Snedecor, G.W. and Cochran, W.C. (1989), *Statistical Methods*, 8th Edition, Ames, IA: Iowa State University Press.

Appendix 2

Operating Environment-Specific Procedures

The following table gives a brief description and the relevant releases for some common operating environment-specific procedures. All of these procedures are described in more detail in operating environment-companion documentation.

Table A2.1 *Host-Specific Procedures*

Procedure	Description	Releases
BMDP	Calls any BMDP program to analyze data in a SAS data set.	All
CONVERT	Converts BMDP, OSIRIS, and SPSS system files to SAS data sets.	All
C16PORT	Converts a 16-bit SAS library or catalog created in Release 6.08 to a transport file, which you can then convert to a 32-bit format for use in the current release of SAS by using the CIMPORT procedure.	6.10 - 6.12
FSDEVICE	Creates, copies, modifies, deletes, or renames device descriptions in a catalog.	All
PDS	Lists, deletes, or renames the members of a partitioned data set.	6.09E
PDSCOPY	Copies partitioned data sets from disk to disk, disk to tape, tape to tape, or tape to disk.	6.09E
RELEASE	Releases unused space at the end of a disk data set.	6.09E
SOURCE	Provides an easy way to back up and process source library data sets.	6.09E
TAPECOPY	Copies an entire tape volume, or files from one or more tape volumes, to one output tape volume.	6.09E
TAPELABEL	Writes the label information of an IBM standard-labeled tape volume to the SAS procedure output file.	6.09E

Appendix 3

Raw Data and DATA Steps

Overview of Raw Data and DATA Steps

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  2007 Allison 31.65 19
Monroe  2007 Barry   23.76 16
Monroe  2007 Candace 21.11  5
Monroe  2007 Danny    6.89 23
Monroe  2007 Edward  53.76 31
Monroe  2007 Fiona   48.55 13
Monroe  2007 Gert    24.00 16
Monroe  2007 Harold  27.55 17
Monroe  2007 Ima     15.98  9
Monroe  2007 Jack    20.00 23
Monroe  2007 Katie   22.11  2
Monroe  2007 Lisa    18.34 17
Monroe  2007 Tonya   55.16 40
Monroe  2007 Max     26.77 34
Monroe  2007 Ned     28.43 22
Monroe  2007 Opal    32.66 14
Monroe  2008 Patsy   18.33 18
Monroe  2008 Quentin 16.89 15
Monroe  2008 Randall 12.98 17
Monroe  2008 Sam     15.88  5
Monroe  2008 Tyra    21.88 23
Monroe  2008 Myrtle  47.33 26
Monroe  2008 Frank   41.11 22
Monroe  2008 Cameron 65.44 14
Monroe  2008 Vern    17.89 11
Monroe  2008 Wendell 23.00 10
Monroe  2008 Bob     26.88  6
Monroe  2008 Leah    28.99 23
Monroe  2009 Becky   30.33 26
Monroe  2009 Sally   35.75 27
Monroe  2009 Edgar   27.11 12
Monroe  2009 Dawson  17.24 16
Monroe  2009 Lou      5.12 16
Monroe  2009 Damien  18.74 17
Monroe  2009 Mona    27.43  7
Monroe  2009 Della   56.78 15
Monroe  2009 Monique 29.88 19
Monroe  2009 Carl    31.12 25
Monroe  2009 Reba    35.16 22
Monroe  2009 Dax     27.65 23
Monroe  2009 Gary    23.11 15
Monroe  2009 Suzie   26.65 11
Monroe  2009 Benito  47.44 18
Monroe  2009 Thomas  21.99 23
Monroe  2009 Annie   24.99 27
Monroe  2009 Paul    27.98 22
```

```
Monroe  2009 Alex    24.00 16
Monroe  2009 Lauren  15.00 17
Monroe  2009 Julia   12.98 15
Monroe  2009 Keith   11.89 19
Monroe  2009 Jackie  26.88 22
Monroe  2009 Pablo   13.98 28
Monroe  2009 L.T.    56.87 33
Monroe  2009 Willard 78.65 24
Monroe  2009 Kathy   32.88 11
Monroe  2009 Abby    35.88 10
Kennedy 2007 Arturo  34.98 14
Kennedy 2007 Grace   27.55 25
Kennedy 2007 Winston 23.88 22
Kennedy 2007 Vince   12.88 21
Kennedy 2007 Claude  15.62  5
Kennedy 2007 Mary    28.99 34
Kennedy 2007 Abner   25.89 22
Kennedy 2007 Jay     35.89 35
Kennedy 2007 Alicia  28.77 26
Kennedy 2007 Freddy  29.00 27
Kennedy 2007 Eloise  31.67 25
Kennedy 2007 Jenny   43.89 22
Kennedy 2007 Thelma  52.63 21
Kennedy 2007 Tina    19.67 21
Kennedy 2007 Eric    24.89 12
Kennedy 2007 Bubba   37.88 12
Kennedy 2007 G.L.    25.89 21
Kennedy 2007 Bert    28.89 21
Kennedy 2008 Clay    26.44 21
Kennedy 2008 Leeann  27.17 17
Kennedy 2008 Georgia 38.90 11
Kennedy 2008 Bill    42.23 25
Kennedy 2008 Holly   18.67 27
Kennedy 2008 Benny   19.09 25
Kennedy 2008 Cammie  28.77 28
Kennedy 2008 Amy     27.08 31
Kennedy 2008 Doris   22.22 24
Kennedy 2008 Robbie  19.80 24
Kennedy 2008 Ted     27.07 25
Kennedy 2008 Sarah   24.44 12
Kennedy 2008 Megan   28.89 11
Kennedy 2008 Jeff    31.11 12
Kennedy 2008 Taz     30.55 11
Kennedy 2008 George  27.56 11
Kennedy 2008 Heather 38.67 15
Kennedy 2009 Nancy   29.90 26
Kennedy 2009 Rusty   30.55 28
Kennedy 2009 Mimi    37.67 22
Kennedy 2009 J.C.    23.33 27
Kennedy 2009 Clark   27.90 25
Kennedy 2009 Rudy    27.78 23
Kennedy 2009 Samuel  34.44 18
Kennedy 2009 Forrest 28.89 26
Kennedy 2009 Luther  72.22 24
Kennedy 2009 Trey     6.78 18
Kennedy 2009 Albert  23.33 19
```

```
Kennedy 2009 Che-Min 26.66 33
Kennedy 2009 Preston 32.22 23
Kennedy 2009 Larry   40.00 26
Kennedy 2009 Anton   35.99 28
Kennedy 2009 Sid     27.45 25
Kennedy 2009 Will    28.88 21
Kennedy 2009 Morty   34.44 25
;
```

CONTROL Library

Contents of the CONTROL Library

The following 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
```

#	Name	Member Type	Obs, Entries or Indexes	Vars	Label	File Size	Last Modified
1	A1	CATALOG	23			62464	04Jan11:14:20:12
2	A2	CATALOG	1			17408	04Jan11:14:20:12
3	ALL	DATA	23	17		13312	04Jan11:14:20:12
4	BODYFAT	DATA	1	2		5120	04Jan11:14:20:12
5	CONFOUND	DATA	8	4		5120	04Jan11:14:20:12
6	CORONARY	DATA	39	4		5120	04Jan11:14:20:12
7	DRUG1	DATA	6	2	JAN11 Data	5120	04Jan11:14:20:12
8	DRUG2	DATA	13	2	MAY11 Data	5120	04Jan11:14:20:12
9	DRUG3	DATA	11	2	JUL10 Data	5120	04Jan11:14:20:12
10	DRUG4	DATA	7	2	JAN10 Data	5120	04Jan11:14:20:12
11	DRUG5	DATA	1	2	JUL10 Data	5120	04Jan11:14:20:12
12	ETEST1	CATALOG	1			17408	04Jan11:14:20:16
13	ETEST2	CATALOG	1			17408	04Jan11:14:20:16
14	ETEST3	CATALOG	1			17408	04Jan11:14:20:16
15	ETEST4	CATALOG	1			17408	04Jan11:14:20:16
16	ETEST5	CATALOG	1			17408	04Jan11:14:20:16
17	ETESTS	CATALOG	1			17408	04Jan11:14:20:16
18	FORMATS	CATALOG	6			17408	04Jan11:14:20:16
19	GROUP	DATA	148	11		25600	04Jan11:14:20:16
20	MLSCL	DATA	32	4	Multiple Sclerosis Data	5120	04Jan11:14:20:16
21	NAMES	DATA	7	4		5120	04Jan11:14:20:16
22	OXYGEN	DATA	31	7		9216	04Jan11:14:20:16
23	PERSONL	DATA	148	11		25600	04Jan11:14:20:16
24	PHARM	DATA	6	3	Sugar Study	5120	04Jan11:14:20:16
25	POINTS	DATA	6	6		5120	04Jan11:14:20:16

```
26  PRENAT    DATA         149         6                      17408 04Jan11:14:20:16
27  RESULTS   DATA          10         5                       5120 04Jan11:14:20:16
28  SLEEP     DATA         108         6                       9216 04Jan11:14:20:16
29  SYNDROME  DATA          46         8                       9216 04Jan11:14:20:16
30  TENSION   DATA           4         3                       5120 04Jan11:14:20:16
31  TEST2     DATA          15         5                       5120 04Jan11:14:20:16
32  TRAIN     DATA           7         2                       5120 04Jan11:14:20:16
33  VISION    DATA          16         3                       5120 04Jan11:14:20:16
34  WEIGHT    DATA          83        13   California         13312 04Jan11:14:20:16
                                          Results
35  WGHT      DATA          83        13   California         13312 04Jan11:14:20:16
                                          Results
16
```

CONTROL.ALL

The following 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

```
 ata 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='JAN2011 DATA');
    input CHAR $8.  NUM;
    datalines;
junk    0
junk    0
junk    0
junk    0
junk    0
junk    0
;
run;
```

CONTROL.DRUG2

```
 data control.drug2 (label='MAY2011 DATA');
    input CHAR $8.  NUM;
    datalines;
junk    0
junk    0
junk    0
junk    0
junk    0
```

```
junk    0
junk    0
junk    0
junk    0
junk    0
junk    0
junk    0
;
run;
```

CONTROL.DRUG3

```
data control.drug3 (label='JUL2010 DATA');
    input CHAR $8.  NUM;
    datalines;
junk    0
junk    0
junk    0
junk    0
junk    0
junk    0
junk    0
junk    0
junk    0
junk    0
junk    0
;
run;
```

CONTROL.DRUG4

```
data control.drug4 (label='JAN2010 DATA');
    input CHAR $8.  NUM;
    datalines;
junk    0
junk    0
junk    0
junk    0
junk    0
junk    0
junk    0
;
run;
```

CONTROL.DRUG5

```
data control.drug5 (label='JUL2010 DATA');
    input CHAR $8.  NUM;
    datalines;
junk    0;
run;
```

CONTROL.GROUP

```
data control.group;
    input IDNUM $ 1-4 LNAME $ 5-19 FNAME $ 20-34 CITY $ 35-49 STATE $
          50-52 SEX $ 53-54 JOBCODE $ 55-58 SALARY comma8. BIRTH
          HIRED date7. HPHONE $ 85-96;
    format salary comma8.;
    format hired date7.;
    informat hired date7.;
    datalines;
1919 ADAMS          GERALD          STAMFORD        CT M TA2   34,377    -4125 07JUN75 203/781-1255
1653 AHMAD          AZEEM           BRIDGEPORT      CT F ME2   35,109    -2631 12AUG78 203/675-7715
1400 ALVAREZ        GLORIA          NEW YORK        NY M ME1   29,770    -1515 19OCT78 212/586-0808
1350 ARTHUR         BARBARA         NEW YORK        NY F FA3   32,887    -2311 01AUG78 718/383-1549
1401 AVERY          JERRY           PATERSON        NJ M TA3   38,823    -7686 20NOV73 201/732-8787
1499 BAREFOOT       JOSEPH          PRINCETON       NJ M ME3   43,026    -6456 10JUN68 201/812-5665
1101 BASQUEZ        RICHARDO        NEW YORK        NY M SCP   18,724    -3493 04OCT78 212/586-8060
1333 BEAULIEU       ARMANDO         NEW YORK        NY M TA2   32,616    -3268 05DEC78 718/384-2849
1479 BOSTIC         MARIE           NEW YORK        NY F TA3   38,786    -1102 08OCT77 718/384-8816
1403 BOWDEN         EARL            BRIDGEPORT      CT M ME1   28,073    -1065 24DEC79 203/675-3434
1739 BOYCE          JONATHAN        NEW YORK        NY M PT1   66,518    -2560 30JAN79 212/587-1247
1658 BRADLEY        JEREMY          NEW YORK        NY M SCP   17,944    -1726 03MAR80 212/587-3622
1428 BRADY          CHRISTINE       STAMFORD        CT F PT1   68,768     -634 19NOV79 203/781-1212
1782 BROWN          JASON           STAMFORD        CT M ME2   35,346     -390 25FEB80 203/781-0019
1244 BRYANT         LEONARD         NEW YORK        NY M ME2   36,926    -3042 20JAN76 718/383-3334
1383 BURNETTE       THOMAS          NEW YORK        NY M BCK   25,824    -1434 23OCT80 718/384-3569
1574 CAHILL         MARSHALL        NEW YORK        NY M FA2   28,573    -4263 23DEC80 718/383-2338
1789 CANALES        VIVIANA         NEW YORK        NY M SCP   18,327    -5451 14APR66 212/587-9000
1404 CARTER         DONALD          NEW YORK        NY M PT2   91,377    -6882 04JAN68 718/384-2946
1437 CARTER         DOROTHY         BRIDGEPORT      CT F FA3   33,105    -4117 03SEP72 203/675-4117
1639 CARTER         KAREN           STAMFORD        CT F TA3   40,261    -5299 31JAN72 203/781-8839
1269 CASTON         FRANKLIN        STAMFORD        CT M NA1   41,691      126 01DEC80 203/781-3335
1065 CHAPMAN        NEIL            NEW YORK        NY M ME2   35,091   -10199 10JAN75 718/384-5618
1876 CHIN           JACK            NEW YORK        NY M TA3   39,676    -4971 30APR73 212/588-5634
1037 CHOW           JANE            STAMFORD        CT F TA1   28,559    -2819 16SEP80 203/781-8868
1129 COOK           BRENDA          NEW YORK        NY F ME2   34,930    -3673 20AUG79 718/383-2313
1988 COOPER         ANTHONY         NEW YORK        NY M FA3   32,218    -4412 21SEP72 212/587-1228
1405 DAVIDSON       JASON           PATERSON        NJ M SCP   18,057    -2125 29JAN80 201/732-2323
1430 DEAN           SANDRA          BRIDGEPORT      CT F TA2   32,926    -3591 30APR75 203/675-1647
1983 DEAN           SHARON          NEW YORK        NY F FA3   33,420    -3591 30APR75 718/384-1647
1134 DELGADO        MARIA           STAMFORD        CT F TA2   33,463    -1029 24DEC76 203/781-1528
1118 DENNIS         ROGER           NEW YORK        NY M PT3  111,380   -10209 21DEC68 718/383-1122
1438 DESAI          AAKASH          STAMFORD        CT F TA3   39,224    -2480 21NOV75 203/781-2229
1125 DUNLAP         DONNA           NEW YORK        NY F FA2   28,889    -1146 14DEC75 718/383-2094
1475 EATON          ALICIA          NEW YORK        NY F FA2   27,788    -3666 16JUL78 718/383-2828
1117 EDGERTON       JOSHUA          NEW YORK        NY M TA3   39,772    -3129 16AUG80 212/588-1239
1935 FERNANDEZ      KATRINA         BRIDGEPORT      CT F NA2   51,082    -6485 19OCT69 203/675-2962
1124 FIELDS         DIANA           WHITE PLAINS    NY F FA1   23,178    -4920 04OCT78 914/455-2998
1422 FLETCHER       MARIE           PRINCETON       NJ F FA1   22,455    -2764 09APR79 201/812-0902
1616 FLOWERS        ANNETTE         NEW YORK        NY F TA2   34,138     -668 07JUN81 718/384-3329
1406 FOSTER         GERALD          BRIDGEPORT      CT M ME2   35,186    -3948 20FEB75 203/675-6363
1120 GARCIA         JACK            NEW YORK        NY M ME1   28,620      257 10OCT81 718/384-4930
1094 GOMEZ          ALAN            BRIDGEPORT      CT M FA1   22,269     -636 20APR79 203/675-7181
1389 GORDON         LEVI            NEW YORK        NY M BCK   25,029    -4550 21AUG78 718/384-9326
```

1905	GRAHAM	ALVIN	NEW YORK	NY	M	PT1	65,112	109	01JUN80 212/586-8815
1407	GRANT	DANIEL	MT. VERNON	NY	M	PT1	68,097	-1011	21MAR78 914/468-1616
1114	GREEN	JANICE	NEW YORK	NY	F	TA2	32,929	-832	30JUN75 212/588-1092
1410	HARRIS	CHARLES	STAMFORD	CT	M	PT2	84,686	-1701	10NOV74 203/781-0937
1439	HARRISON	FELICIA	BRIDGEPORT	CT	F	PT1	70,737	-2854	13SEP78 203/675-4987
1409	HARTFORD	RAYMOND	STAMFORD	CT	M	ME3	41,552	-7924	25OCT69 203/781-9697
1408	HENDERSON	WILLIAM	PRINCETON	NJ	M	TA2	34,139	-4292	17OCT75 201/812-4789
1121	HERNANDEZ	MICHAEL	NEW YORK	NY	M	ME1	29,113	-94	10DEC79 718/384-3313
1991	HOLMES	GABRIEL	BRIDGEPORT	CT	F	TA1	27,646	130	15DEC80 203/675-0007
1102	HOLMES	SHANE	WHITE PLAINS	NY	M	TA2	34,543	-4472	18APR79 914/455-0976
1356	HOLMES	SHAWN	NEW YORK	NY	M	ME2	36,870	-5207	25FEB71 212/586-8411
1545	HUNTER	CLYDE	STAMFORD	CT	M	PT1	66,131	-4522	01JUN78 203/781-1119
1292	HUNTER	HELEN	BRIDGEPORT	CT	F	ME2	36,692	-2618	05JUL77 203/675-4830
1440	JACKSON	LAURA	STAMFORD	CT	F	ME2	35,758	-3380	12APR79 203/781-0088
1368	JEPSEN	RONALD	STAMFORD	CT	M	FA2	27,809	-3853	06NOV72 203/781-8413
1369	JOHNSON	ANTHONY	NEW YORK	NY	M	TA2	33,706	-3653	16MAR75 212/587-5385
1411	JOHNSON	JACKSON	PATERSON	NJ	M	FA2	27,266	-3868	04DEC77 201/732-3678
1113	JONES	LESLIE	NEW YORK	NY	F	FA1	22,368	-1444	20OCT79 718/383-3003
1704	JOSHI	ABHAY	NEW YORK	NY	M	BCK	25,466	-1947	01JUL75 718/384-0049
1900	KING	WILLIAM	NEW YORK	NY	M	ME2	35,106	-3505	30OCT75 718/383-3698
1126	KOSTECKA	NICHOLAS	NEW YORK	NY	F	TA3	40,900	-3137	24NOV68 212/586-1229
1677	KRAMER	JACKSON	BRIDGEPORT	CT	M	BCK	26,008	-2976	30MAR77 203/675-7432
1441	LAWRENCE	KATHY	PRINCETON	NJ	F	FA2	27,159	-770	26MAR79 201/812-3337
1421	LEE	RUSSELL	MT. VERNON	NY	M	TA2	33,156	-4738	03MAR78 914/468-9143
1119	LI	JEFF	NEW YORK	NY	M	TA1	26,925	-3479	09SEP76 212/586-2344
1834	LONG	RUSSELL	NEW YORK	NY	M	BCK	26,897	41	05JUL80 718/384-0040
1777	LUFKIN	ROY	NEW YORK	NY	M	PT3	109,631	-7402	24JUN69 718/383-4413
1663	MAHANNAHS	SHANTHA	NEW YORK	NY	M	BCK	26,453	-1813	14AUG79 212/587-7742
1106	MARSHBURN	JASPER	STAMFORD	CT	M	PT2	89,633	-5166	19AUG72 203/781-1457
1103	MCDANIEL	RONDA	NEW YORK	NY	F	FA1	23,739	-1412	26JUL80 212/586-0013
1477	MEYERS	PRESTON	BRIDGEPORT	CT	M	FA2	28,567	-2839	10MAR76 203/675-8125
1476	MONROE	JOYCE	STAMFORD	CT	F	TA2	34,804	-2039	20MAR75 203/781-2837
1379	MORGAN	ALFRED	STAMFORD	CT	M	ME3	42,265	-3795	13JUN72 203/781-2216
1104	MORGAN	CHRISTOPHER	NEW YORK	NY	M	SCP	17,947	-3170	13JUN79 718/383-9740
1009	MORGAN	GEORGE	NEW YORK	NY	M	TA1	28,881	-4685	29MAR80 212/586-7753
1412	MURPHEY	JOHN	PRINCETON	NJ	M	ME1	27,800	-5672	08DEC79 201/812-4414
1115	MURPHY	ALICE	NEW YORK	NY	F	FA3	32,700	-4146	03MAR68 718/384-1982
1128	NELSON	FELICIA	BRIDGEPORT	CT	F	TA2	32,778	-2411	23OCT78 203/675-1166
1442	NEWKIRK	SANDRA	PRINCETON	NJ	F	PT2	84,537	-1941	15APR76 201/812-3331
1417	NEWKIRK	WILLIAM	PATERSON	NJ	M	NA2	52,271	-2741	10MAR77 201/732-6611
1478	NEWTON	JAMES	NEW YORK	NY	M	PT2	84,204	-4525	27OCT78 212/587-5549
1673	NICHOLLS	HENRY	STAMFORD	CT	M	BCK	25,478	-670	18JUL79 203/781-7770
1839	NORRIS	DIANE	NEW YORK	NY	F	NA1	43,434	-395	06JUL81 718/384-1767
1347	O'NEAL	BRYAN	NEW YORK	NY	M	TA3	40,080	-1560	09SEP72 718/384-0230
1423	OSWALD	LESLIE	MT. VERNON	NY	F	ME2	35,774	-1324	22AUG78 914/468-9171
1200	OVERMAN	MICHELLE	STAMFORD	CT	F	ME1	27,817	-353	17AUG80 203/781-1835
1970	PAPI	PAOLO	NEW YORK	NY	F	FA1	22,616	-2651	15MAR79 718/383-3895
1521	PAPIA	ISMAEL	NEW YORK	NY	M	ME3	41,527	-3183	16JUL76 212/587-7603
1354	PAPIA	FRANCISCO	WHITE PLAINS	NY	F	SCP	18,336	-214	19JUN80 914/455-2337
1424	PATTERSON	RENEE	NEW YORK	NY	F	FA2	28,979	-877	14DEC77 212/587-8991
1132	PEARCE	CAROL	NEW YORK	NY	F	FA1	22,414	153	25OCT81 718/384-1986
1845	PEARSON	JAMES	NEW YORK	NY	M	BCK	25,997	-4422	25MAR68 718/384-2311
1556	PENNINGTON	MICHAEL	NEW YORK	NY	M	PT1	71,350	-2746	14DEC79 718/383-5681
1413	PETERS	RANDALL	PRINCETON	NJ	M	FA2	27,436	-2295	05JAN78 201/812-2478
1907	PHELPS	WILLIAM	STAMFORD	CT	M	TA2	33,330	-4061	09JUL75 203/781-1118
1436	PORTER	SUSAN	NEW YORK	NY	F	TA2	34,476	-2757	15MAR75 718/383-5777

```
1385 RAYNOR        MILTON         BRIDGEPORT     CT M ME3   43,901    -3634 04APR74 203/675-2846
1432 REED          MARILYN        MT. VERNON     NY F ME2   35,328    -3708 13FEB73 914/468-5454
1111 RHODES        JEREMY         PRINCETON      NJ M NA1   40,587      563 03NOV80 201/812-1837
1116 RICHARDS      CASEY          NEW YORK       NY F FA1   22,863     -822 24MAR79 212/587-1224
1352 RIVERS        SIMON          NEW YORK       NY M NA2   53,799    -4044 19OCT74 718/383-3345
1555 RODRIGUEZ     JULIA          BRIDGEPORT     CT F FA2   27,500    -1383 07JUL80 203/675-2401
1038 RODRIGUEZ     MARIA          BRIDGEPORT     CT F TA1   26,534     -780 26NOV79 203/675-2048
1420 ROUSE         JEREMY         PATERSON       NJ M ME3   43,072    -2504 25JUL75 201/732-9834
1561 SANDERS       RAYMOND        NEW YORK       NY M TA2   34,515    -2951 10OCT75 212/588-6615
1434 SANDERSON     EDITH          STAMFORD       CT F FA2   28,623    -3458 31OCT78 203/781-1333
1414 SARKAR        ABHEEK         BRIDGEPORT     CT M FA1   23,645       86 15APR80 203/675-1715
1112 SAYERS        RANDY          NEW YORK       NY M TA1   26,906    -2586 10DEC80 718/384-4895
1390 SMART         JONATHAN       NEW YORK       NY M FA2   27,762    -2504 26JUN79 718/383-1141
1332 STEPHENSON    ADAM           BRIDGEPORT     CT M NA1   42,179     -468 07JUN79 203/675-1497
1890 STEPHENSON    ROBERT         NEW YORK       NY M PT2   85,897    -7467 28NOV67 718/384-9874
1429 THOMPSON      ALICE          STAMFORD       CT F TA1   27,940    -4322 10AUG80 203/781-3857
1107 THOMPSON      WAYNE          NEW YORK       NY M PT2   89,978    -6412 13FEB67 718/384-3785
1908 TRENTON       MELISSA        NEW YORK       NY F TA2   32,996     -749 26APR78 212/586-6262
1830 TRIPP         KATHY          BRIDGEPORT     CT F PT2   84,472    -5329 01FEB71 203/675-2479
1882 TUCKER        ALAN           NEW YORK       NY M ME3   41,539    -5285 24NOV66 718/384-0216
1050 TUTTLE        THOMAS         WHITE PLAINS   NY M ME2   35,168    -3090 27AUG74 914/455-2119
1425 UNDERWOOD     JENNY          STAMFORD       CT F FA1   23,980       -1 03MAR81 203/781-0978
1928 UPCHURCH      LARRY          WHITE PLAINS   NY M PT2   89,859    -6313 16JUL78 914/455-5009
1480 UPDIKE        THERESA        NEW YORK       NY F TA3   39,584    -5230 28MAR69 212/587-8729
1100 VANDEUSEN     RICHARD        NEW YORK       NY M BCK   25,005    -4045 10MAY76 212/586-2531
1995 VARNER        ELIZABETH      NEW YORK       NY F ME1   28,811      604 22SEP81 718/384-7113
1135 VEGA          ANNA           NEW YORK       NY F FA2   27,322    -4117 03APR78 718/384-5913
1415 VEGA          FRANKLIN       NEW YORK       NY M FA2   28,279    -5043 15FEB76 718/384-2823
1076 VENTER        RANDALL        NEW YORK       NY M PT1   66,559      290 06OCT79 718/383-2321
1426 VICK          THERESA        PRINCETON      NJ F TA2   32,992    -1850 28JUN78 201/812-2424
1564 WALTERS       ANNE           NEW YORK       NY F SCP   18,834    -3548 04JUL80 212/587-3257
1221 WALTERS       DIANE          NEW YORK       NY F FA2   27,897    -1559 07OCT79 718/384-1918
1133 WANG          CHIN           NEW YORK       NY M TA1   27,702    -1995 15FEB80 212/587-1956
1435 WARD          ELAINE         NEW YORK       NY F TA3   38,809    -4614 11FEB68 718/383-4987
1418 WATSON        BERNARD        NEW YORK       NY M ME1   28,006    -5388 09JAN80 718/383-1298
1017 WELCH         DARIUS         NEW YORK       NY M TA3   40,859    -5114 19OCT69 212/586-5535
1443 WELLS         AGNES          STAMFORD       CT F NA1   42,275    -1137 01SEP79 203/781-5546
1131 WELLS         NADINE         NEW YORK       NY F TA2   32,576       -3 22APR79 718/383-1045
1427 WHALEY        CAROLYN        MT. VERNON     NY F TA2   34,047     -424 02FEB78 914/468-4528
1036 WONG          LESLIE         NEW YORK       NY F TA3   39,393    -2415 26OCT72 212/587-2570
1130 WOOD          DEBORAH        NEW YORK       NY F FA1   23,917     -227 08JUN80 212/587-0013
1127 WOOD          SANDRA         NEW YORK       NY F TA2   33,012    -2606 10DEC74 212/587-2881
1433 YANCEY        ROBIN          PRINCETON      NJ F FA3   32,983    -2000 20JAN75 201/812-1874
1431 YOUNG         DEBORAH        STAMFORD       CT F FA3   33,231    -2759 08APR76 203/781-2987
1122 YOUNG         JOANN          NEW YORK       NY F FA2   27,957    -3164 30NOV76 718/384-2021
1105 YOUNG         LAWRENCE       NEW YORK       NY M ME2   34,806    -3590 16AUG78 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 $ 5-19 FNAME $ 20-34 CITY $ 35-49 STATE $ 50-51
          SEX $ 53-53 JOBCODE $ 55-57 SALARY BIRTH date. @66
          HIRED date7. @74 HPHONE $ 84-95;
    format birth date7.;
    informat birth date.;
    format hired date7.;
    informat hired date.;
    datalines;
1919 ADAMS          GERALD         STAMFORD       CT M TA2  34376 15SEP70 07JUN05  203/781-1255
1653 AHMAD          AZEEM          BRIDGEPORT     CT F ME2  35108 18OCT72 12AUG98  203/675-7715
1400 ALVAREZ        GLORIA         NEW YORK       NY M ME1  29769 08NOV85 19OCT06  212/586-0808
1350 ARTHUR         BARBARA        NEW YORK       NY F FA3  32886 03SEP63 01AUG00  718/383-1549
1401 AVERY          JERRY          PATERSON       NJ M TA3  38822 16DEC68 20NOV93  201/732-8787
1499 BAREFOOT       JOSEPH         PRINCETON      NJ M ME3  43025 29APR62 10JUN95  201/812-5665
1101 BASQUEZ        RICHARDO       NEW YORK       NY M SCP  18723 09JUN80 04OCT98  212/586-8060
1333 BEAULIEU       ARMANDO        STAMFORD       CT M PT2  88606 02APR79 13FEB03  203/781-1777
1402 BLALOCK        RALPH          NEW YORK       NY M TA2  32615 20JAN71 05DEC98  718/384-2849
1479 BOSTIC         MARIE          NEW YORK       NY F TA3  38785 25DEC66 08OCT03  718/384-8816
```

1403	BOWDEN	EARL	BRIDGEPORT	CT	M	ME1	28072	31JAN79	24DEC99	203/675-3434
1739	BOYCE	JONATHAN	NEW YORK	NY	M	PT1	66517	28DEC82	30JAN00	212/587-1247
1658	BRADLEY	JEREMY	NEW YORK	NY	M	SCP	17943	11APR65	03MAR00	212/587-3622
1428	BRADY	CHRISTINE	STAMFORD	CT	F	PT1	68767	07APR80	19NOV02	203/781-1212
1428	BRADy	CHRISTINE	STAMFORD	CT	F	PT1	68767	07APR80	19NOV02	203/781-1212
1782	BROWN	JASON	STAMFORD	CT	M	ME2	35345	07DEC73	25FEB00	203/781-0019
1244	BRYANT	LEONARD	NEW YORK	NY	M	ME2	36925	03SEP71	20JAN96	718/383-3334
1383	BURNETTE	THOMAS	NEW YORK	NY	M	BCK	25824	28JAN76	23OCT00	718/384-3569
1574	CAHILL	MARSHALL	NEW YORK	NY	M	FA2	28572	30APR74	23DEC97	718/383-2338
1789	CANALES	VIVIANA	NEW YORK	NY	M	SCP	18326	28JAN85	14APR04	212/587-9000
1404	CARTER	DONALD	NEW YORK	NY	M	PT2	91376	27FEB71	04JAN98	718/384-2946
1437	CARTER	DOROTHY	BRIDGEPORT	CT	F	FA3	33104	23SEP68	03SEP92	203/675-4117
1639	CARTER	KAREN	STAMFORD	CT	F	TA3	40260	29JUN65	31JAN92	203/781-8839
1269	CASTON	FRANKLIN	STAMFORD	CT	M	NA1	41690	06MAY80	01DEC00	203/781-3335
1065	CHAPMAN	NEIL	NEW YORK	NY	M	ME2	35090	29JAN72	10JAN95	718/384-5618
1876	CHIN	JACK	NEW YORK	NY	M	TA3	39675	23MAY66	30APR96	212/588-5634
1037	CHOW	JANE	STAMFORD	CT	F	TA1	28558	13APR82	16SEP04	203/781-8868
1129	COOK	BRENDA	NEW YORK	NY	F	ME2	34929	11DEC79	20AUG03	718/383-2313
1988	COOPER	ANTHONY	NEW YORK	NY	M	FA3	32217	03DEC57	21SEP92	212/587-1228
1405	DAVIDSON	JASON	PATERSON	NJ	M	SCP	18056	08MAR54	29JAN00	201/732-2323
1430	DEAN	SANDRA	BRIDGEPORT	CT	F	TA2	32925	03MAR70	30APR05	203/675-1647
1983	DEAN	SHARON	NEW YORK	NY	F	FA3	33419	03MAR50	30APR85	718/384-1647
1134	DELGADO	MARIA	STAMFORD	CT	F	TA2	33462	08MAR77	24DEC04	203/781-1528
1118	DENNIS	ROGER	NEW YORK	NY	M	PT3	111379	19JAN57	21DEC88	718/383-1122
1438	DESAI	AAKASH	STAMFORD	CT	F	TA3	39223	18MAR63	21NOV03	203/781-2229
1125	DUNLAP	DONNA	NEW YORK	NY	F	FA2	28888	11NOV76	14DEC95	718/383-2094
1475	EATON	ALICIA	NEW YORK	NY	F	FA2	27787	18DEC71	16JUL98	718/383-2828
1117	EDGERTON	JOSHUA	NEW YORK	NY	M	TA3	39771	08JUN56	16AUG00	212/588-1239
1935	FERNANDEZ	KATRINA	BRIDGEPORT	CT	F	NA2	51081	31MAR72	19OCT01	203/675-2962
1124	FIELDS	DIANA	WHITE PLAINS	NY	F	FA1	23177	13JUL82	04OCT01	914/455-2998
1422	FLETCHER	MARIE	PRINCETON	NJ	F	FA1	22454	07JUN79	09APR99	201/812-0902
1616	FLOWERS	ANNETTE	NEW YORK	NY	F	TA2	34137	04MAR68	07JUN01	718/384-3329
1406	FOSTER	GERALD	BRIDGEPORT	CT	M	ME2	35185	11MAR69	20FEB95	203/675-6363
1120	GARCIA	JACK	NEW YORK	NY	M	ME1	28619	14SEP80	10OCT01	718/384-4930
1094	GOMEZ	ALAN	BRIDGEPORT	CT	M	FA1	22268	05APR78	20APR99	203/675-7181
1389	GORDON	LEVI	NEW YORK	NY	M	BCK	25028	18JUL67	21AUG03	718/384-9326
1905	GRAHAM	ALVIN	NEW YORK	NY	M	PT1	65111	19APR80	01JUN00	212/586-8815
1407	GRANT	DANIEL	MT. VERNON	NY	M	PT1	68096	26MAR77	21MAR98	914/468-1616
1114	GREEN	JANICE	NEW YORK	NY	F	TA2	32928	21SEP77	30JUN06	212/588-1092
1410	HARRIS	CHARLES	STAMFORD	CT	M	PT2	84685	06MAY75	10NOV99	203/781-0937
1439	HARRISON	FELICIA	BRIDGEPORT	CT	F	PT1	70736	09MAR72	13SEP98	203/675-4987
1409	HARTFORD	RAYMOND	STAMFORD	CT	M	ME3	41551	22APR68	25OCT99	203/781-9697
1408	HENDERSON	WILLIAM	PRINCETON	NJ	M	TA2	34138	01APR78	17OCT05	201/812-4789
1121	HERNANDEZ	MICHAEL	NEW YORK	NY	M	ME1	29112	29SEP79	10DEC99	718/384-3313
1991	HOLMES	GABRIEL	BRIDGEPORT	CT	F	TA1	27645	10MAY80	15DEC00	203/675-0007
1102	HOLMES	SHANE	WHITE PLAINS	NY	M	TA2	34542	04OCT77	18APR05	914/455-0976
1356	HOLMES	SHAWN	NEW YORK	NY	M	ME2	36869	29SEP75	25FEB01	212/586-8411
1545	HUNTER	CLYDE	STAMFORD	CT	M	PT1	66130	15AUG77	01JUN05	203/781-1119
1292	HUNTER	HELEN	BRIDGEPORT	CT	F	ME2	36691	31OCT72	05JUL97	203/675-4830
1440	JACKSON	LAURA	STAMFORD	CT	F	ME2	35757	30SEP70	12APR99	203/781-0088
1368	JEPSEN	RONALD	STAMFORD	CT	M	FA2	27808	14JUN69	06NOV02	203/781-8413
1369	JOHNSON	ANTHONY	NEW YORK	NY	M	TA2	33705	31DEC79	16MAR05	212/587-5385
1411	JOHNSON	JACKSON	PATERSON	NJ	M	FA2	27265	30MAY69	04DEC97	201/732-3678
1113	JONES	LESLIE	NEW YORK	NY	F	FA1	22367	18JAN76	20OCT99	718/383-3003
1704	JOSHI	ABHAY	NEW YORK	NY	M	BCK	25465	02SEP74	01JUL95	718/384-0049
1900	KING	WILLIAM	NEW YORK	NY	M	ME2	35105	28MAY70	30OCT95	718/383-3698

```
1126 kOSTECKA      NICHOLAS      NEW YORK       NY F TA3  40899 31MAY71 24NOV98  212/586-1229
1677 KRAMER        JACKSON       BRIDGEPORT     CT M BCK  26007 08NOV71 30MAR97  203/675-7432
1441 LAWRENCE      KATHY         PRINCETON      NJ F FA2  27158 22NOV77 26MAR99  201/812-3337
1421 LEE           RUSSELL       MT. VERNON     NY M TA2  33155 11JAN67 03MAR98  914/468-9143
1119 LI            JEFF          NEW YORK       NY M TA1  26924 23JUN70 09SEP96  212/586-2344
1834 LONG          RUSSELL       NEW YORK       NY M BCK  26896 11FEB80 05JUL00  718/384-0040
1777 LUFKIN        ROY           NEW YORK       NY M PT3 109630 26SEP59 24JUN89  718/383-4413
1663 MAHANNAHS     SHANTHA       NEW YORK       NY M BCK  26452 14JAN75 14AUG99  212/587-7742
1106 MARSHBURN     JASPER        STAMFORD       CT M PT2  89632 09NOV65 19AUG92  203/781-1457
1103 MCDANIEL      RONDA         NEW YORK       NY F FA1  23738 19FEB76 26JUL90  212/586-0013
1477 MEYERS        PRESTON       BRIDGEPORT     CT M FA2  28566 24MAR72 10MAR96  203/675-8125
1476 MONROE        JOYCE         STAMFORD       CT F TA2  34803 02JUN74 20MAR95  203/781-2837
1379 MORGAN        ALFRED        STAMFORD       CT M ME3  42264 11AUG69 13JUN02  203/781-2216
1104 MORGAN        CHRISTOPHER   NEW YORK       NY M SCP  17946 28APR71 13JUN99  718/383-9740
1009 MORGAN        GEORGE        NEW YORK       NY M TA1  28880 05MAR67 29MAR00  212/586-7753
1412 MURPHEY       JOHN          PRINCETON      NJ M ME1  27799 21JUN64 08DEC99  201/812-4414
1115 MURPHY        ALICE         NEW YORK       NY F FA3  32699 25AUG68 03MAR91  718/384-1982
1128 NELSON        FELICIA       BRIDGEPORT     CT F TA2  32777 26MAY73 23OCT98  203/675-1166
1442 NEWKIRK       SANDRA        PRINCETON      NJ F PT2  84536 08SEP74 15APR96  201/812-3331
1417 NEWKIRK       WILLIAM       PATERSON       NJ M NA2  52270 30JUN72 10MAR97  201/732-6611
1478 NEWTON        JAMES         NEW YORK       NY M PT2  84203 12AUG67 27OCT98  212/587-5549
1673 NICHOLLS      HENRY         STAMFORD       CT M BCK  25477 02MAR78 18JUL99  203/781-7770
1839 NORRIS        DIANE         NEW YORK       NY F NA1  44433 02DEC78 06JUL01  718/384-1767
1347 O'NEAL        BRYAN         NEW YORK       NY M TA3  40079 24SEP75 09SEP95  718/384-0230
1423 OSWALD        LESLIE        MT. VERNON     NY F ME2  35773 17MAY76 22AUG98  914/468-9171
1200 OVERMAN       MICHELLE      STAMFORD       CT F ME1  27816 13JAN79 17AUG02  203/781-1835
1970 PAPI          PAOLO         NEW YORK       NY F FA1  22615 28SEP72 15MAR99  718/383-3895
1521 PAPIA         ISMAEL        NEW YORK       NY M ME3  41526 15APR71 16JUL96  212/587-7603
1354 PAPIA         FRANCISCO     WHITE PLAINS   NY F SCP  18335 01JUN79 19JUN00  914/455-2337
1424 PATTERSON     RENEE         NEW YORK       NY F FA2  28978 07AUG77 14DEC97  212/587-8991
1132 PEARCE        CAROL         NEW YORK       NY F FA1  22413 02JUN80 25OCT01  718/384-1986
1845 PEARSON       JAMES         NEW YORK       NY M BCK  25996 23NOV67 25MAR95  718/384-2311
1556 PENNINGTON    MICHAEL       NEW YORK       NY M PT1  71349 25JUN72 14DEC99  718/383-5681
1413 PETERS        RANDALL       PRINCETON      NJ M FA2  27435 19SEP73 05JAN98  201/812-2478
1907 PHELPS        WILLIAM       STAMFORD       CT M TA2  33329 18NOV68 09JUL95  203/781-1118
1436 PORTER        SUSAN         NEW YORK       NY F TA2  34475 14JUN72 15MAR95  718/383-5777
1385 RAYNOR        MILTON        BRIDGEPORT     CT M ME3  43900 19JAN70 04APR94  203/675-2846
1432 REED          MARILYN       MT. VERNON     NY F ME2  35327 06NOV69 13FEB93  914/468-5454
1111 RHODES        JEREMY        PRINCETON      NJ M NA1  40586 17JUL81 03NOV02  201/812-1837
1116 RICHARDS      CASEY         NEW YORK       NY F FA1  22862 01OCT77 24MAR99  212/587-1224
1352 RIVERS        SIMON         NEW YORK       NY M NA2  53798 05DEC68 19OCT94  718/383-3345
1555 RODRIGUEZ     JULIA         BRIDGEPORT     CT F FA2  27499 19MAR76 07JUL00  203/675-2401
1038 RODRIGUEZ     MARIA         BRIDGEPORT     CT F TA1  26533 12NOV77 26NOV99  203/675-2048
1420 ROUSE         JEREMY        PATERSON       NJ M ME3  43071 22FEB73 25JUL95  201/732-9834
1561 SANDERS       RAYMOND       NEW YORK       NY M TA2  34514 03DEC71 10OCT95  212/588-6615
1434 SANDERSON     EDITH         STAMFORD       CT F FA2  28622 14JUL70 31OCT98  203/781-1333
1414 SARKAR        ABHEEK        BRIDGEPORT     CT M FA1  23644 27MAR80 15APR00  203/675-1715
1112 SAYERS        RANDY         NEW YORK       NY M TA1  26905 02DEC72 10DEC00  718/384-4895
1390 SMART         JONATHAN      NEW YORK       NY M FA2  27761 22FEB73 26JUN09  718/383-1141
1332 STEPHENSON    ADAM          BRIDGEPORT     CT M NA1  42178 20SEP78 07JUN99  203/675-1497
1890 STEPHENSON    ROBERT        NEW YORK       NY M PT2  85896 23JUL59 28NOV87  718/384-9874
1429 THOMPSON      ALICE         STAMFORD       CT F TA1  27939 02MAR68 10AUG00  203/781-3857
1107 THOMPSON      WAYNE         NEW YORK       NY M PT2  89977 12JUN62 13FEB87  718/384-3785
1908 TRENTON       MELISSA       NEW YORK       NY F TA2  32995 13DEC77 26APR98  212/586-6262
1830 TRIPP         KATHY         BRIDGEPORT     CT F PT2  84471 30MAY65 01FEB91  203/675-2479
1882 TUCKER        ALAN          NEW YORK       NY M ME3  41538 13JUL65 24NOV86  718/384-0216
```

```
1050 TUTTLE        THOMAS         WHITE PLAINS   NY M ME2  35167 17JUL71 27AUG94  914/455-2119
1425 UNDERWOOD     JENNY          STAMFORD       CT F FA1  23979 31DEC79 03MAR01  203/781-0978
1928 UPCHURCH      LARRY          WHITE PLAINS   NY M PT2  89858 19SEP62 16JUL98  914/455-5009
1480 UPDIKE        THERESA        NEW YORK       NY F TA3  39583 06SEP65 28MAR89  212/587-8729
1100 VANDEUSEN     RICHARD        NEW YORK       NY M BCK  25004 04DEC68 10MAY96  212/586-2531
1995 VARNER        ELIZABETH      NEW YORK       NY F ME1  28810 27AUG81 22SEP01  718/384-7113
1135 VEGA          ANNA           NEW YORK       NY F FA2  27321 23SEP68 03APR98  718/384-5913
1415 VEGA          FRANKLIN       NEW YORK       NY M FA2  28278 12MAR66 15FEB96  718/384-2823
1076 VENTER        RANDALL        NEW YORK       NY M PT1  66558 17OCT80 06OCT99  718/383-2321
1426 VICK          THERESA        PRINCETON      NJ F TA2  32991 08DEC74 28JUN98  201/812-2424
1564 WALTERS       ANNE           NEW YORK       NY F SCP  18833 15APR70 04JUL00  212/587-3257
1221 WALTERS       DIANE          NEW YORK       NY F FA2  27896 25SEP75 07OCT99  718/384-1918
1133 WANG          CHIN           NEW YORK       NY M TA1  27701 16JUL74 15FEB00  212/587-1956
1435 WARD          ELAINE         NEW YORK       NY F TA3  38808 15MAY67 11FEB88  718/383-4987
1418 WATSON        BERNARD        NEW YORK       NY M ME1  28005 01APR65 09JAN00  718/383-1298
1017 WELCH         DARIUS         NEW YORK       NY M TA3  40858 31DEC65 19OCT89  212/586-5535
1443 WELLS         AGNES          STAMFORD       CT F NA1  42274 20NOV66 01SEP99  203/781-5546
1131 WELLS         NADINE         NEW YORK       NY F TA2  32575 29DEC79 22APR99  718/383-1045
1427 WHALEY        CAROLYN        MT. VERNON     NY F TA2  34046 03NOV78 02FEB98  914/468-4528
1036 WONG          LESLIE         NEW YORK       NY F TA3  39392 22MAY73 26OCT92  212/587-2570
1130 WOOD          DEBORAH        NEW YORK       NY F FA1  23916 19MAY79 08JUN00  212/587-0013
1127 WOOD          SANDRA         NEW YORK       NY F TA2  33011 12NOV72 10DEC94  212/587-2881
1433 YANCEY        ROBIN          PRINCETON      NJ F FA3  32982 11JUL74 20JAN95  201/812-1874
1431 YOUNG         DEBORAH        STAMFORD       CT F FA3  33230 12JUN72 08APR96  203/781-2987
1122 YOUNG         JOANN          NEW YORK       NY F FA2  27956 04MAY71 30NOV96  718/384-2021
1105 YOUNG         LAWRENCE       NEW YORK       NY M ME2  34805 04MAR70 16AUG98  718/384-0008
;
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      01MAR11     7:10     LGA     LAX      2475      172         210
202      01MAR11    10:43     LGA     ORD       740      151         210
219      01MAR11     9:31     LGA     LON      3442      198         250
622      01MAR11    12:19     LGA     FRA      3857      207         250
132      01MAR11    15:35     LGA     YYZ       366      115         178
271      01MAR11    13:17     LGA     PAR      3635      138         250
302      01MAR11    20:22     LGA     WAS       229      105         180
114      02MAR11     7:10     LGA     LAX      2475      119         210
202      02MAR11    10:43     LGA     ORD       740      120         210
219      02MAR11     9:31     LGA     LON      3442      147         250
622      02MAR11    12:19     LGA     FRA      3857      176         250
132      02MAR11    15:35     LGA     YYZ       366      106         178
302      02MAR11    20:22     LGA     WAS       229       78         180
271      02MAR11    13:17     LGA     PAR      3635      104         250
114      03MAR11     7:10     LGA     LAX      2475      197         210
202      03MAR11    10:43     LGA     ORD       740      118         210
219      03MAR11     9:31     LGA     LON      3442      197         250
622      03MAR11    12:19     LGA     FRA      3857      180         250
132      03MAR11    15:35     LGA     YYZ       366       75         178
271      03MAR11    13:17     LGA     PAR      3635      147         250
302      03MAR11    20:22     LGA     WAS       229      123         180
114      04MAR11     7:10     LGA     LAX      2475      178         210
202      04MAR11    10:43     LGA     ORD       740      148         210
219      04MAR11     9:31     LGA     LON      3442      232         250
622      04MAR11    12:19     LGA     FRA      3857      137         250
132      04MAR11    15:35     LGA     YYZ       366      117         178
271      04MAR11    13:17     LGA     PAR      3635      146         250
```

```
302      04MAR11    20:22     LGA     WAS       229      115         180
114      05MAR11     7:10     LGA     LAX      2475      117         210
202      05MAR11    10:43     LGA     ORD       740      104         210
219      05MAR11     9:31     LGA     LON      3442      160         250
622      05MAR11    12:19     LGA     FRA      3857      185         250
132      05MAR11    15:35     LGA     YYZ       366      157         178
271      05MAR11    13:17     LGA     PAR      3635      177         250
114      06MAR11     7:10     LGA     LAX      2475      128         210
202      06MAR11    10:43     LGA     ORD       740      115         210
219      06MAR11     9:31     LGA     LON      3442      163         250
132      06MAR11    15:35     LGA     YYZ       366      150         178
302      06MAR11    20:22     LGA     WAS       229       66         180
114      07MAR11     7:10     LGA     LAX      2475      160         210
202      07MAR11    10:43     LGA     ORD       740      175         210
219      07MAR11     9:31     LGA     LON      3442      241         250
622      07MAR11    12:19     LGA     FRA      3857      210         250
132      07MAR11    15:35     LGA     YYZ       366      164         178
271      07MAR11    13:17     LGA     PAR      3635      155         250
302      07MAR11    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  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;
```

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 . 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 . 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 date7. High LowDate date7. Low;
      format highdate lowdate date7.;
      datalines;
1968 03DEC68   985.21 21MAR68   825.13
1969 14MAY69   968.85 17DEC69   769.93
1970 29DEC70   842.00 06MAY70   631.16
1971 28APR71   950.82 23NOV71   797.97
1972 11DEC72  1036.27 26JAN72   889.15
1973 11JAN73  1051.70 05DEC73   788.31
1974 13MAR74   891.66 06DEC74   577.60
1975 15JUL75   881.81 02JAN75   632.04
1976 21SEP76  1014.79 02JAN76   858.71
1977 03JAN77   999.75 02NOV77   800.85
1978 08SEP78   907.74 28FEB78   742.12
1979 05OCT79   897.61 07NOV79   796.67
1980 20NOV80  1000.17 21APR80   759.13
1981 27APR81  1024.05 25SEP81   824.01
1982 27DEC82  1070.55 12AUG82   776.92
1983 29NOV83  1287.20 03JAN83  1027.04
1984 06JAN84  1286.64 24JUL84  1086.57
1985 16DEC85  1553.10 04JAN85  1184.96
1986 02DEC86  1955.57 22JAN86  1502.29
1987 25AUG87  2722.42 19OCT87  1738.74
```

```
1988 21OCT88  2183.50 20JAN88  1879.14
1989 09OCT89  2791.41 03JAN89  2144.64
1990 16JUL90  2999.75 11OCT90  2365.10
1991 31DEC91  3168.83 09JAN91  2470.30
1992 01JUN92  3413.21 09OCT92  3136.58
1993 29DEC93  3794.33 20JAN93  3241.95
1994 31JAN94  3978.36 04APR94  3593.35
1995 13DEC95  5216.47 30JAN95  3832.08
1996 27DEC96  6560.90 10JAN96  5032.94
1997 06AUG97  8259.30 11APR97  6391.69
1998 23NOV98  9374.27 31AUG98  7539.06
1999 31DEC99 11497.12 22JAN99  9120.67
2000 14JAN00 11722.98 07MAR00  9796.04
2001 21MAY01 11337.92 21SEP01  8235.81
2002 19MAR02 10635.25 09OCT02  7286.27
2003 31DEC03 10453.92 11MAR03  7524.06
2004 28DEC04 10854.54 25OCT04  9749.99
2005 04MAR05 10940.55 20APR05 10012.36
2006 27DEC06 12510.57 20JAN06 10667.39
2007 09OCT07 14164.53 05MAR07 12050.41
2008 02MAY08 13058.20 10OCT08  8451.19
;
```

EDUCATION

```
data education;
   input State $14. +1 Code $ DropoutRate Expenditures MathScore Region $;
   label dropoutrate='Dropout Percentage - 2008'
         expenditures='Expenditure Per Pupil - 2008'
           mathscore='8th Grade Math Exam - 2009';
   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 $ 9-19 FirstName $ 20-29
      City $ 30-42 State $ 43-44 /
      Gender $ 1 JobCode $ 9-11 Salary 20-29 @30 Birth date7.
      @43 Hired date7. HomePhone $ 54-65;
format birth hired date7.;
datalines;
1919    Adams      Gerald    Stamford     CT M       TA2        34376     15SEP70      07JUN05    203/781-1255
1653    Alexander  Susan     Bridgeport   CT F       ME2        35108     18OCT72      12AUG98    203/675-7715
1400    Apple      Troy      New York     NY M       ME1        29769     08NOV85      19OCT06    212/586-0808
1350    Arthur     Barbara   New York     NY F       FA3        32886     03SEP63      01AUG00    718/383-1549
1401    Avery      Jerry     Paterson     NJ M       TA3        38822     16DEC68      20NOV93    201/732-8787
1499    Barefoot   Joseph    Princeton    NJ M       ME3        43025     29APR62      10JUN95    201/812-5665
1101    Baucom     Walter    New York     NY M       SCP        18723     09JUN80      04OCT98    212/586-8060
1333    Blair      Justin    Stamford     CT M       PT2        88606     02APR79      13FEB03    203/781-1777
1402    Blalock    Ralph     New York     NY M       TA2        32615     20JAN71      05DEC98    718/384-2849
1479    Bostic     Marie     New York     NY F       TA3        38785     25DEC66      08OCT03    718/384-8816
1403    Bowden     Earl      Bridgeport   CT M       ME1        28072     31JAN79      24DEC99    203/675-3434
1739    Boyce      Jonathan  New York     NY M       PT1        66517     28DEC82      30JAN00    212/587-1247
1658    Bradley    Jeremy    New York     NY M       SCP        17943     11APR65      03MAR00    212/587-3622
1428    Brady      Christine Stamford     CT F       PT1        68767     07APR80      19NOV02    203/781-1212
1782    Brown      Jason     Stamford     CT M       ME2        35345     07DEC73      25FEB00    203/781-0019
1244    Bryant     Leonard   New York     NY M       ME2        36925     03SEP71      20JAN96    718/383-3334
1383    Burnette   Thomas    New York     NY M       BCK        25823     28JAN76      23OCT00    718/384-3569
1574    Cahill     Marshall  New York     NY M       FA2        28572     30APR74      23DEC97    718/383-2338
1789    Caraway    Davis     New York     NY M       SCP        18326     28JAN85      14APR04    212/587-9000
1404    Carter     Donald    New York     NY M       PT2        91376     27FEB71      04JAN98    718/384-2946
1437    Carter     Dorothy   Bridgeport   CT F       A3         33104     23SEP68      03SEP92    203/675-4117
1639    Carter     Karen     Stamford     CT F       A3         40260     29JUN65      31JAN92    203/781-8839
1269    Caston     Franklin  Stamford     CT M       NA1        41690     06MAY80      01DEC00    203/781-3335
1065    Chapman    Neil      New York     NY M       ME2        35090     29JAN72      10JAN95    718/384-5618
1876    Chin       Jack      New York     NY M       TA3        39675     23MAY66      30APR96    212/588-5634
1037    Chow       Jane      Stamford     CT F       TA1        28558     13APR82      16SEP04    203/781-8868
1129    Cook       Brenda    New York     NY F       ME2        34929     11DEC79      20AUG03    718/383-2313
1988    Cooper     Anthony   New York     NY M       FA3        32217     03DEC57      21SEP92    212/587-1228
1405    Davidson   Jason     Paterson     NJ M       SCP        18056     08MAR54      29JAN00    201/732-2323
```

```
1430    Dean       Sandra    Bridgeport   CT F      TA2       32925     03MAR70     30APR05    203/675-1647
1983    Dean       Sharon    New York     NY F      FA3       33419     03MAR50     30APR85    718/384-1647
1134    Delgado    Maria     Stamford     CT F      TA2       33462     08MAR77     24DEC04    203/781-1528
1118    Dennis     Roger     New York     NY M      PT3       111379    19JAN57     21DEC88    718/383-1122
1438    Donaldson  Karen     Stamford     CT F      TA3       39223     18MAR63     21NOV03    203/781-2229
1125    Dunlap     Donna     New York     NY F      FA2       28888     11NOV76     14DEC95    718/383-2094
1475    Eaton      Alicia    New York     NY F      FA2       27787     18DEC71     16JUL98    718/383-2828
1117    Edgerton   Joshua    New York     NY M      TA3       39771     08JUN56     16AUG00    212/588-1239
1935    Fernandez  Katrina   Bridgeport   CT F      NA2       51081     31MAR72     19OCT01    203/675-2962
1124    Fields     Diana     White Plains NY F      FA1       23177     13JUL82     04OCT01    914/455-2998
1422    Fletcher   Marie     Princeton    NJ F      FA1       22454     07JUN79     09APR99    201/812-0902
1616    Flowers    Annette   New York     NY F      TA2       34137     04MAR68     07JUN01    718/384-3329
1406    Foster     Gerald    Bridgeport   CT M      ME2       35185     11MAR69     20FEB95    203/675-6363
1120    Garcia     Jack      New York     NY M      ME1       28619     14SEP80     10OCT01    718/384-4930
1094    Gomez      Alan      Bridgeport   CT M      FA1       22268     05APR78     20APR99    203/675-7181
1389    Gordon     Levi      New York     NY M      BCK       25028     18JUL67     21AUG03    718/384-9326
1905    Graham     Alvin     New York     NY M      PT1       65111     19APR80     01JUN00    212/586-8815
1407    Grant      Daniel    Mt. Vernon   NY M      PT1       68096     26MAR77     21MAR98    914/468-1616
1114    Green      Janice    New York     NY F      TA2       32928     21SEP77     30JUN06    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.

```
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
14    surgery    203.60    169.78     38
17    surgery    171.52    150.33     42
18    surgery    207.46    155.22     41
;
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

```
ods html close;
data exprev;
input Country $ 1-24 Emp_ID $ 25-32 Order_Date $  Ship_Date $  Sale_Type $ 67-75 Quantity Price Cost;

datalines;
Antarctica              99999999    1/1/12     1/7/12          Internet    2     92.60      20.70
Puerto Rico             99999999    1/1/12     1/5/12          Catalog    14     51.20      12.10
Virgin Islands (U.S.)   99999999    1/1/12     1/4/12          In Store   25     31.10      15.65
Aruba                   99999999    1/1/12     1/4/12          Catalog    30    123.70      59.00
Bahamas                 99999999    1/1/12     1/4/12          Catalog     8    113.40      28.45
Bermuda                 99999999    1/1/12     1/4/12          Catalog     7     41.00       9.25
Belize                  120458      1/2/12     1/2/12          In Store    2    146.40      36.70
British Virgin Islands  99999999    1/2/12     1/5/12          Catalog    11     40.20      20.20
Canada                  99999999    1/2/12     1/5/12          Catalog   100     11.80       5.00
Cayman Islands          120454      1/2/12     1/2/12          In Store   20     71.00      32.30
Costa Rica              99999999    1/2/12     1/6/12          Internet   31     53.00      26.60
Cuba                    121044      1/2/12     1/2/12          Internet   12     42.40      19.35
Dominican Republic      121040      1/2/12     1/2/12          Internet   13     48.00      23.95
El Salvador             99999999    1/2/12     1/6/12          Catalog    21    266.40      66.70
Guatemala               120931      1/2/12     1/2/12          In Store   13    144.40      65.70
Haiti                   121059      1/2/12     1/2/12          Internet    5     47.90      23.45
Honduras                120455      1/2/12     1/2/12          Internet   20     66.40      30.25
Jamaica                 99999999    1/2/12     1/4/12          In Store   23    169.80      38.70
Mexico                  120127      1/2/12     1/2/12          In Store   30    211.80      33.65
Montserrat              120127      1/2/12     1/2/12          In Store   19    184.20      36.90
Nicaragua               120932      1/2/12     1/2/12          Internet   16    122.00      28.75
Panama                  99999999    1/2/12     1/6/12          Internet   20     88.20      38.40
Saint Kitts/Nevis       99999999    1/2/12     1/6/12          Internet   20     41.40      18.00
St. Helena              120360      1/2/12     1/2/12          Internet   19     94.70      47.45
St. Pierre/Miquelon     120842      1/2/12     1/16/12         Internet   16    103.80      47.25
Turks/Caicos Islands    120372      1/2/12     1/2/12          Internet   10     57.70      28.95
United States           120372      1/2/12     1/2/12          Internet   20     88.20      38.40
Anguilla                99999999    1/2/12     1/6/12          In Store   15    233.50      22.25
Antigua/Barbuda         120458      1/2/12     1/2/12          In Store   31     99.60      45.35
Argentina               99999999    1/2/12     1/6/12          In Store   42    408.80      87.15
Barbados                99999999    1/2/12     1/6/12          In Store   26     94.80      42.60
Bolivia                 120127      1/2/12     1/2/12          In Store   26     66.00      16.60
Brazil                  120127      1/2/12     1/2/12          Catalog    12     73.40      18.45
Chile                   120447      1/2/12     1/2/12          In Store   20     19.10       8.75
Colombia                121059      1/2/12     1/2/12          Internet   28    361.40      90.45
Dominica                121043      1/2/12     1/2/12          Internet   35    121.30      57.80
Ecuador                 121042      1/2/12     1/2/12          In Store   11    100.90      50.55
Falkland Islands        120932      1/2/12     1/2/12          In Store   15     61.40      30.80
French Guiana           120935      1/2/12     1/2/12          Catalog    15     96.40      43.85
Grenada                 120931      1/2/12     1/2/12          Catalog    19     56.30      25.05
```

```
Guadeloupe              120445      1/2/12      1/2/12          Internet   21    231.60      48.70
Guyana                  120455      1/2/12      1/2/12          In Store   25    132.80      30.25
Martinique              120841      1/2/12      1/3/12          In Store   16    56.30       31.05
Netherlands Antilles    99999999    1/2/12      1/6/12          In Store   31    41.80       19.45
Paraguay                120603      1/2/12      1/2/12          Catalog    17    117.60      58.90
Peru                    120845      1/2/12      1/2/12          Catalog    12    93.80       41.75
St. Lucia               120845      1/2/12      1/2/12          Internet   19    64.30       28.65
Suriname                120538      1/3/12      1/3/12          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;
   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     01MAR08  LGA   LAX   1-10 Minutes     Domestic              8
202     01MAR08  LGA   ORD   No Delay         Domestic             -5
219     01MAR08  LGA   LON   11+ Minutes      International        18
622     01MAR08  LGA   FRA   No Delay         International        -5
132     01MAR08  LGA   YYZ   11+ Minutes      International        14
271     01MAR08  LGA   PAR   1-10 Minutes     International         5
302     01MAR08  LGA   WAS   No Delay         Domestic             -2
114     02MAR08  LGA   LAX   No Delay         Domestic              0
202     02MAR08  LGA   ORD   1-10 Minutes     Domestic              5
219     02MAR08  LGA   LON   11+ Minutes      International        18
622     02MAR08  LGA   FRA   No Delay         International         0
132     02MAR08  LGA   YYZ   1-10 Minutes     International         5
271     02MAR08  LGA   PAR   1-10 Minutes     International         4
302     02MAR08  LGA   WAS   No Delay         Domestic              0
114     03MAR08  LGA   LAX   No Delay         Domestic             -1
202     03MAR08  LGA   ORD   No Delay         Domestic             -1
219     03MAR08  LGA   LON   1-10 Minutes     International         4
622     03MAR08  LGA   FRA   No Delay         International        -2
132     03MAR08  LGA   YYZ   1-10 Minutes     International         6
271     03MAR08  LGA   PAR   1-10 Minutes     International         2
```

```
302     03MAR08  LGA   WAS   1-10 Minutes     Domestic              5
114     04MAR08  LGA   LAX   11+ Minutes      Domestic             15
202     04MAR08  LGA   ORD   No Delay         Domestic             -5
219     04MAR08  LGA   LON   1-10 Minutes     International         3
622     04MAR08  LGA   FRA   11+ Minutes      International        30
132     04MAR08  LGA   YYZ   No Delay         International        -5
271     04MAR08  LGA   PAR   1-10 Minutes     International         5
302     04MAR08  LGA   WAS   1-10 Minutes     Domestic              7
114     05MAR08  LGA   LAX   No Delay         Domestic             -2
202     05MAR08  LGA   ORD   1-10 Minutes     Domestic              2
219     05MAR08  LGA   LON   1-10 Minutes     International         3
622     05MAR08  LGA   FRA   No Delay         International        -6
132     05MAR08  LGA   YYZ   1-10 Minutes     International         3
271     05MAR08  LGA   PAR   1-10 Minutes     International         5
114     06MAR08  LGA   LAX   No Delay         Domestic             -1
202     06MAR08  LGA   ORD   No Delay         Domestic             -3
219     06MAR08  LGA   LON   11+ Minutes      International        27
132     06MAR08  LGA   YYZ   1-10 Minutes     International         7
302     06MAR08  LGA   WAS   1-10 Minutes     Domestic              1
114     07MAR08  LGA   LAX   No Delay         Domestic             -1
202     07MAR08  LGA   ORD   No Delay         Domestic             -2
219     07MAR08  LGA   LON   11+ Minutes      International        15
622     07MAR08  LGA   FRA   11+ Minutes      International        21
132     07MAR08  LGA   YYZ   No Delay         International        -2
271     07MAR08  LGA   PAR   1-10 Minutes     International         4
302     07MAR08  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     01MAR08  LON        198
622     01MAR08  FRA        207
132     01MAR08  YYZ        115
271     01MAR08  PAR        138
219     02MAR08  LON        147
622     02MAR08  FRA        176
132     02MAR08  YYZ        106
271     02MAR08  PAR        172
219     03MAR08  LON        197
622     03MAR08  FRA        180
132     03MAR08  YYZ         75
271     03MAR08  PAR        147
219     04MAR08  LON        232
622     04MAR08  FRA        137
132     04MAR08  YYZ        117
271     04MAR08  PAR        146
219     05MAR08  LON        160
622     05MAR08  FRA        185
132     05MAR08  YYZ        157
271     05MAR08  PAR        177
219     06MAR08  LON        163
132     06MAR08  YYZ        150
219     07MAR08  LON        241
622     07MAR08  FRA        210
132     07MAR08  YYZ        164
271     07MAR08  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     01MAR08    7:10  LGA   LAX       2475       172       210
202     01MAR08   10:43  LGA   ORD        740       151       210
219     01MAR08    9:31  LGA   LON       3442       198       250
622     01MAR08   12:19  LGA   FRA       3857       207       250
132     01MAR08   15:35  LGA   YYZ        366       115       178
271     01MAR08   13:17  LGA   PAR       3635       138       250
302     01MAR08   20:22  LGA   WAS        229       105       180
114     02MAR08    7:10  LGA   LAX       2475       119       210
202     02MAR08   10:43  LGA   ORD        740       120       210
219     02MAR08    9:31  LGA   LON       3442       147       250
622     02MAR08   12:19  LGA   FRA       3857       176       250
132     02MAR08   15:35  LGA   YYZ        366       106       178
302     02MAR08   20:22  LGA   WAS        229        78       180
271     02MAR08   13:17  LGA   PAR       3635       104       250
114     03MAR08    7:10  LGA   LAX       2475       197       210
202     03MAR08   10:43  LGA   ORD        740       118       210
219     03MAR08    9:31  LGA   LON       3442       197       250
622     03MAR08   12:19  LGA   FRA       3857       180       250
132     03MAR08   15:35  LGA   YYZ        366        75       178
271     03MAR08   13:17  LGA   PAR       3635       147       250
302     03MAR08   20:22  LGA   WAS        229       123       180
114     04MAR08    7:10  LGA   LAX       2475       178       210
202     04MAR08   10:43  LGA   ORD        740       148       210
219     04MAR08    9:31  LGA   LON       3442       232       250
622     04MAR08   12:19  LGA   FRA       3857       137       250
132     04MAR08   15:35  LGA   YYZ        366       117       178
271     04MAR08   13:17  LGA   PAR       3635       146       250
302     04MAR08   20:22  LGA   WAS        229       115       180
114     05MAR08    7:10  LGA   LAX       2475       117       210
202     05MAR08   10:43  LGA   ORD        740       104       210
219     05MAR08    9:31  LGA   LON       3442       160       250
```

```
622     05MAR08   12:19  LGA   FRA       3857       185       250
132     05MAR08   15:35  LGA   YYZ        366       157       178
271     05MAR08   13:17  LGA   PAR       3635       177       250
114     06MAR08    7:10  LGA   LAX       2475       128       210
202     06MAR08   10:43  LGA   ORD        740       115       210
219     06MAR08    9:31  LGA   LON       3442       163       250
132     06MAR08   15:35  LGA   YYZ        366       150       178
302     06MAR08   20:22  LGA   WAS        229        66       180
114     07MAR08    7:10  LGA   LAX       2475       160       210
202     07MAR08   10:43  LGA   ORD        740       175       210
219     07MAR08    9:31  LGA   LON       3442       241       250
622     07MAR08   12:19  LGA   FRA       3857       210       250
132     07MAR08   15:35  LGA   YYZ        366       164       178
271     07MAR08   13:17  LGA   PAR       3635       155       250
302     07MAR08   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=date7.
                  format=date7.,
        Hired num informat=date7.
                  format=date7.);

insert into proclib.paylist2
values('1919','M','TA2',34376,'12SEP66'd,'04JUN87'd)
values('1653','F','ME2',31896,'15OCT64'd,'09AUG92'd)
values('1350','F','FA3',36886,'31AUG55'd,'29JUL91'd)
values('1401','M','TA3',38822,'13DEC55'd,'17NOV93'd)
values('1499','M','ME1',23025,'26APR74'd,'07JUN92'd);

title 'PROCLIB.PAYLIST2 Table';
select * from proclib.paylist2;
```

PROCLIB.PAYROLL

This data set (table) is updated in “Example 3: Updating Data in a PROC SQL Table” in Chapter 7 of *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. +9 Salary 5.
         +2 Birth date7. +2 Hired date7.;
   informat birth date7. hired date7.;
```

```
   format birth date7. hired date7.;
   datalines;
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
1403   M    ME1         28072  28JAN69  21DEC91
1739   M    PT1         66517  25DEC64  27JAN91
1658   M    SCP         17943  08APR67  29FEB92
1428   F    PT1         68767  04APR60  16NOV91
1782   M    ME2         35345  04DEC70  22FEB92
1244   M    ME2         36925  31AUG63  17JAN88
1383   M    BCK         25823  25JAN68  20OCT92
1574   M    FA2         28572  27APR60  20DEC92
1789   M    SCP         18326  25JAN57  11APR78
1404   M    PT2         91376  24FEB53  01JAN80
1437   F    FA3         33104  20SEP60  31AUG84
1639   F    TA3         40260  26JUN57  28JAN84
1269   M    NA1         41690  03MAY72  28NOV92
1065   M    ME2         35090  26JAN44  07JAN87
1876   M    TA3         39675  20MAY58  27APR85
1037   F    TA1         28558  10APR64  13SEP92
1129   F    ME2         34929  08DEC61  17AUG91
1988   M    FA3         32217  30NOV59  18SEP84
1405   M    SCP         18056  05MAR66  26JAN92
1430   F    TA2         32925  28FEB62  27APR87
1983   F    FA3         33419  28FEB62  27APR87
1134   F    TA2         33462  05MAR69  21DEC88
1118   M    PT3        111379  16JAN44  18DEC80
1438   F    TA3         39223  15MAR65  18NOV87
1125   F    FA2         28888  08NOV68  11DEC87
1475   F    FA2         27787  15DEC61  13JUL90
1117   M    TA3         39771  05JUN63  13AUG92
1935   F    NA2         51081  28MAR54  16OCT81
1124   F    FA1         23177  10JUL58  01OCT90
1422   F    FA1         22454  04JUN64  06APR91
1616   F    TA2         34137  01MAR70  04JUN93
1406   M    ME2         35185  08MAR61  17FEB87
1120   M    ME1         28619  11SEP72  07OCT93
1094   M    FA1         22268  02APR70  17APR91
1389   M    BCK         25028  15JUL59  18AUG90
1905   M    PT1         65111  16APR72  29MAY92
1407   M    PT1         68096  23MAR69  18MAR90
1114   F    TA2         32928  18SEP69  27JUN87
1410   M    PT2         84685  03MAY67  07NOV86
1439   F    PT1         70736  06MAR64  10SEP90
1409   M    ME3         41551  19APR50  22OCT81
1408   M    TA2         34138  29MAR60  14OCT87
1121   M    ME1         29112  26SEP71  07DEC91
1991   F    TA1         27645  07MAY72  12DEC92
```

```
1102   M    TA2         34542  01OCT59  15APR91
1356   M    ME2         36869  26SEP57  22FEB83
1545   M    PT1         66130  12AUG59  29MAY90
1292   F    ME2         36691  28OCT64  02JUL89
1440   F    ME2         35757  27SEP62  09APR91
1368   M    FA2         27808  11JUN61  03NOV84
1369   M    TA2         33705  28DEC61  13MAR87
1411   M    FA2         27265  27MAY61  01DEC89
1113   F    FA1         22367  15JAN68  17OCT91
1704   M    BCK         25465  30AUG66  28JUN87
1900   M    ME2         35105  25MAY62  27OCT87
1126   F    TA3         40899  28MAY63  21NOV80
1677   M    BCK         26007  05NOV63  27MAR89
1441   F    FA2         27158  19NOV69  23MAR91
1421   M    TA2         33155  08JAN59  28FEB90
1119   M    TA1         26924  20JUN62  06SEP88
1834   M    BCK         26896  08FEB72  02JUL92
1777   M    PT3        109630  23SEP51  21JUN81
1663   M    BCK         26452  11JAN67  11AUG91
1106   M    PT2         89632  06NOV57  16AUG84
1103   F    FA1         23738  16FEB68  23JUL92
1477   M    FA2         28566  21MAR64  07MAR88
1476   F    TA2         34803  30MAY66  17MAR87
1379   M    ME3         42264  08AUG61  10JUN84
1104   M    SCP         17946  25APR63  10JUN91
1009   M    TA1         28880  02MAR59  26MAR92
1412   M    ME1         27799  18JUN56  05DEC91
1115   F    FA3         32699  22AUG60  29FEB80
1128   F    TA2         32777  23MAY65  20OCT90
1442   F    PT2         84536  05SEP66  12APR88
1417   M    NA2         52270  27JUN64  07MAR89
1478   M    PT2         84203  09AUG59  24OCT90
1673   M    BCK         25477  27FEB70  15JUL91
1839   F    NA1         43433  29NOV70  03JUL93
1347   M    TA3         40079  21SEP67  06SEP84
1423   F    ME2         35773  14MAY68  19AUG90
1200   F    ME1         27816  10JAN71  14AUG92
1970   F    FA1         22615  25SEP64  12MAR91
1521   M    ME3         41526  12APR63  13JUL88
1354   F    SCP         18335  29MAY71  16JUN92
1424   F    FA2         28978  04AUG69  11DEC89
1132   F    FA1         22413  30MAY72  22OCT93
1845   M    BCK         25996  20NOV59  22MAR80
1556   M    PT1         71349  22JUN64  11DEC91
1413   M    FA2         27435  16SEP65  02JAN90
1123   F    TA1         28407  31OCT72  05DEC92
1907   M    TA2         33329  15NOV60  06JUL87
1436   F    TA2         34475  11JUN64  12MAR87
1385   M    ME3         43900  16JAN62  01APR86
1432   F    ME2         35327  03NOV61  10FEB85
1111   M    NA1         40586  14JUL73  31OCT92
1116   F    FA1         22862  28SEP69  21MAR91
1352   M    NA2         53798  02DEC60  16OCT86
1555   F    FA2         27499  16MAR68  04JUL92
1038   F    TA1         26533  09NOV69  23NOV91
1420   M    ME3         43071  19FEB65  22JUL87
```

```
1561   M    TA2         34514  30NOV63  07OCT87
1434   F    FA2         28622  11JUL62  28OCT90
1414   M    FA1         23644  24MAR72  12APR92
1112   M    TA1         26905  29NOV64  07DEC92
1390   M    FA2         27761  19FEB65  23JUN91
1332   M    NA1         42178  17SEP70  04JUN91
1890   M    PT2         91908  20JUL51  25NOV79
1429   F    TA1         27939  28FEB60  07AUG92
1107   M    PT2         89977  09JUN54  10FEB79
1908   F    TA2         32995  10DEC69  23APR90
1830   F    PT2         84471  27MAY57  29JAN83
1882   M    ME3         41538  10JUL57  21NOV78
1050   M    ME2         35167  14JUL63  24AUG86
1425   F    FA1         23979  28DEC71  28FEB93
1928   M    PT2         89858  16SEP54  13JUL90
1480   F    TA3         39583  03SEP57  25MAR81
1100   M    BCK         25004  01DEC60  07MAY88
1995   F    ME1         28810  24AUG73  19SEP93
1135   F    FA2         27321  20SEP60  31MAR90
1415   M    FA2         28278  09MAR58  12FEB88
1076   M    PT1         66558  14OCT55  03OCT91
1426   F    TA2         32991  05DEC66  25JUN90
1564   F    SCP         18833  12APR62  01JUL92
1221   F    FA2         27896  22SEP67  04OCT91
1133   M    TA1         27701  13JUL66  12FEB92
1435   F    TA3         38808  12MAY59  08FEB80
1418   M    ME1         28005  29MAR57  06JAN92
1017   M    TA3         40858  28DEC57  16OCT81
1443   F    NA1         42274  17NOV68  29AUG91
1131   F    TA2         32575  26DEC71  19APR91
1427   F    TA2         34046  31OCT70  30JAN90
1036   F    TA3         39392  19MAY65  23OCT84
1130   F    FA1         23916  16MAY71  05JUN92
1127   F    TA2         33011  09NOV64  07DEC86
1433   F    FA3         32982  08JUL66  17JAN87
1431   F    FA3         33230  09JUN64  05APR88
1122   F    FA2         27956  01MAY63  27NOV88
1105   M    ME2         34805  01MAR62  13AUG90
;
```

PROCLIB.PAYROLL2

```
data proclib.payroll2;
   input idnum $4. +3 gender $1. +4 jobcode $3. +9 salary 5.
         +2 birth date7. +2 hired date7.;
   informat birth date7. hired date7.;
   format birth date7. hired date7.;
   datalines;
1639   F    TA3         42260  26JUN57  28JAN84
1065   M    ME3         38090  26JAN44  07JAN87
1561   M    TA3         36514  30NOV63  07OCT87
1221   F    FA3         29896  22SEP67  04OCT91
1447   F    FA1         22123  07AUG72  29OCT92
```

```
1998   M    SCP         23100  10SEP70  02NOV92
1036   F    TA3         42465  19MAY65  23OCT84
1106   M    PT3         94039  06NOV57  16AUG84
1129   F    ME3         36758  08DEC61  17AUG91
1350   F    FA3         36098  31AUG65  29JUL90
1369   M    TA3         36598  28DEC61  13MAR87
1076   M    PT1         69742  14OCT55  03OCT91
;
```

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     01MAR08  YYZ   1739
132     01MAR08  YYZ   1478
132     01MAR08  YYZ   1130
132     01MAR08  YYZ   1390
132     01MAR08  YYZ   1983
132     01MAR08  YYZ   1111
219     01MAR08  LON   1407
219     01MAR08  LON   1777
219     01MAR08  LON   1103
219     01MAR08  LON   1125
219     01MAR08  LON   1350
219     01MAR08  LON   1332
271     01MAR08  PAR   1439
271     01MAR08  PAR   1442
271     01MAR08  PAR   1132
271     01MAR08  PAR   1411
271     01MAR08  PAR   1988
271     01MAR08  PAR   1443
622     01MAR08  FRA   1545
622     01MAR08  FRA   1890
622     01MAR08  FRA   1116
622     01MAR08  FRA   1221
622     01MAR08  FRA   1433
622     01MAR08  FRA   1352
132     02MAR08  YYZ   1556
132     02MAR08  YYZ   1478
132     02MAR08  YYZ   1113
132     02MAR08  YYZ   1411
132     02MAR08  YYZ   1574
132     02MAR08  YYZ   1111
219     02MAR08  LON   1407
219     02MAR08  LON   1118
219     02MAR08  LON   1132
219     02MAR08  LON   1135
219     02MAR08  LON   1441
219     02MAR08  LON   1332
271     02MAR08  PAR   1739
```

```
271     02MAR08  PAR   1442
271     02MAR08  PAR   1103
271     02MAR08  PAR   1413
271     02MAR08  PAR   1115
271     02MAR08  PAR   1443
622     02MAR08  FRA   1439
622     02MAR08  FRA   1890
622     02MAR08  FRA   1124
622     02MAR08  FRA   1368
622     02MAR08  FRA   1477
622     02MAR08  FRA   1352
132     03MAR08  YYZ   1739
132     03MAR08  YYZ   1928
132     03MAR08  YYZ   1425
132     03MAR08  YYZ   1135
132     03MAR08  YYZ   1437
132     03MAR08  YYZ   1111
219     03MAR08  LON   1428
219     03MAR08  LON   1442
219     03MAR08  LON   1130
219     03MAR08  LON   1411
219     03MAR08  LON   1115
219     03MAR08  LON   1332
271     03MAR08  PAR   1905
271     03MAR08  PAR   1118
271     03MAR08  PAR   1970
271     03MAR08  PAR   1125
271     03MAR08  PAR   1983
271     03MAR08  PAR   1443
622     03MAR08  FRA   1545
622     03MAR08  FRA   1830
622     03MAR08  FRA   1414
622     03MAR08  FRA   1368
622     03MAR08  FRA   1431
622     03MAR08  FRA   1352
132     04MAR08  YYZ   1428
132     04MAR08  YYZ   1118
132     04MAR08  YYZ   1103
132     04MAR08  YYZ   1390
132     04MAR08  YYZ   1350
132     04MAR08  YYZ   1111
219     04MAR08  LON   1739
219     04MAR08  LON   1478
219     04MAR08  LON   1130
219     04MAR08  LON   1125
219     04MAR08  LON   1983
219     04MAR08  LON   1332
271     04MAR08  PAR   1407
271     04MAR08  PAR   1410
271     04MAR08  PAR   1094
271     04MAR08  PAR   1411
271     04MAR08  PAR   1115
271     04MAR08  PAR   1443
622     04MAR08  FRA   1545
622     04MAR08  FRA   1890
622     04MAR08  FRA   1116
```

```
622     04MAR08  FRA   1221
622     04MAR08  FRA   1433
622     04MAR08  FRA   1352
132     05MAR08  YYZ   1556
132     05MAR08  YYZ   1890
132     05MAR08  YYZ   1113
132     05MAR08  YYZ   1475
132     05MAR08  YYZ   1431
132     05MAR08  YYZ   1111
219     05MAR08  LON   1428
219     05MAR08  LON   1442
219     05MAR08  LON   1422
219     05MAR08  LON   1413
219     05MAR08  LON   1574
219     05MAR08  LON   1332
271     05MAR08  PAR   1739
271     05MAR08  PAR   1928
271     05MAR08  PAR   1103
271     05MAR08  PAR   1477
271     05MAR08  PAR   1433
271     05MAR08  PAR   1443
622     05MAR08  FRA   1545
622     05MAR08  FRA   1830
622     05MAR08  FRA   1970
622     05MAR08  FRA   1441
622     05MAR08  FRA   1350
622     05MAR08  FRA   1352
132     06MAR08  YYZ   1333
132     06MAR08  YYZ   1890
132     06MAR08  YYZ   1414
132     06MAR08  YYZ   1475
132     06MAR08  YYZ   1437
132     06MAR08  YYZ   1111
219     06MAR08  LON   1106
219     06MAR08  LON   1118
219     06MAR08  LON   1425
219     06MAR08  LON   1434
219     06MAR08  LON   1555
219     06MAR08  LON   1332
132     07MAR08  YYZ   1407
132     07MAR08  YYZ   1118
132     07MAR08  YYZ   1094
132     07MAR08  YYZ   1555
132     07MAR08  YYZ   1350
132     07MAR08  YYZ   1111
219     07MAR08  LON   1905
219     07MAR08  LON   1478
219     07MAR08  LON   1124
219     07MAR08  LON   1434
219     07MAR08  LON   1983
219     07MAR08  LON   1332
271     07MAR08  PAR   1410
271     07MAR08  PAR   1777
271     07MAR08  PAR   1103
271     07MAR08  PAR   1574
271     07MAR08  PAR   1115
```

```
271     07MAR08  PAR   1443
622     07MAR08  FRA   1107
622     07MAR08  FRA   1890
622     07MAR08  FRA   1425
622     07MAR08  FRA   1475
622     07MAR08  FRA   1433
622     07MAR08  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	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
;
```

PROCLIB.STAFF2

```
data proclib.staff;
   input Name & $16. IdNumber $ Salary
         Site $ HireDate date7.;
   format hiredate date7.;
   datalines;
Capalleti, Jimmy  2355 21163 BR1 30JAN09
Chen, Len         5889 20976 BR1 18JUN06
Davis, Brad       3878 19571 BR2 20MAR84
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
;
```

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 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 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 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 0 7 0 0 0 0
490 35 f 1 1 7
7 0 0 0 7 0 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 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 4

ICU License

Index

B

C

D

E

F

G

H

M

N

O

P

S

T

U

V

W

X

Z