SAS Publishing

Base SAS® 9.1 Procedures Guide

The Power to Know®

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2004. *Base SAS® 9.1 Procedures Guide*. Cary, NC: SAS Institute Inc.

Base SAS® 9.1 Procedures Guide

ISBN 1-59047-204-7

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

1st printing, January 2004

Contents

x

What's New

Overview

New and enhanced Base SAS procedures in 9 and 9.1

- improve ODS formatting
- enable import and export of Microsoft Excel 2002 spreadsheets and Microsoft Access 2002 tables
- support long format and informat names
- list and compare SAS registries
- support parallel sorting operations
- improve statistical processing
- improve printer definitions.

A list of ODS table names is now provided for each procedure that supports ODS. You can use these names to reference the table when using the Output Delivery System (ODS) to select tables and create output data sets.

Note:

- This section describes the features of Base SAS procedures that are new or enhanced since SAS 8.2.
- z/OS is the successor to the OS/390 operating system. SAS 9.1 is supported on both OS/390 and z/OS operating systems and, throughout this document, any reference to z/OS also applies to OS/390, unless otherwise stated.

△

Details

The CONTENTS Procedure

The new look for output from the CONTENTS procedure and the CONTENTS statement in PROC DATASETS provides a better format for the Output Delivery

System (ODS). PROC CONTENTS output now displays the data representation of a file by reporting the native platform for each file, rather than just telling you whether the data representation is native or foreign. Also, PROC CONTENTS output now provides the encoding value, whether a character variable is transcoded if required, and whether the data set is part of a generation group. A new example was added that shows how to get PROC CONTENTS output into an ODS output data set for processing.

The ORDER= option was added to the CONTENTS statement to enable you to print a list of variables in alphabetical order even if they include mixed-case names.

The COPY Procedure

The following options are new or enhanced in the COPY procedure and the COPY statement in PROC DATASETS:

- The FORCE option enables you to use the MOVE option for a SAS data set that has an audit trail.
- The CLONE option now copies the data representation and encoding data set attributes.

The CORR Procedure

The CORR procedure has the following new features:

- The FISHER option in the PROC CORR statement requests confidence limits and *p*–values for Pearson and Spearman correlation coefficients based on Fisher's *z* transformation. With the FISHER option, you can specify an alpha value and a null hypothesis value. You can also specify the type of confidence limit (upper, lower, or two-sided) and whether the bias adjustment is to be used for the confidence limits.
- The PLOTS=MATRIX option in the PROC CORR statement uses ODS graphics to produce either a rectangular matrix plot (if you also specify a WITH statement) or a symmetric matrix plot (if you do not specify a WITH statement) for variables.
- The PLOTS=SCATTER option in the PROC CORR statement uses ODS graphics to produce scatter plots for variables. By default, the scatter plot also includes a 95% prediction ellipse. You can use the ELLIPSE= option with the PLOTS=SCATTER option to include prediction ellipses for new observations, confidence ellipses for the mean, or no ellipses.

The DATASETS Procedure

Directory listings from the DATASETS procedure provide a new look for its output, which improves the format for the Output Delivery System (ODS).

The following statements are enhanced in the DATASETS procedure:

- The AUDIT statement provides the AUDIT_ALL= option in order to specify whether logging can be suspended and whether audit settings can be changed. In addition, the LOG option in the AUDIT statement now enables you to control the logging of administrative events to the audit file with ADMIN_IMAGE=.
- The ICCREATE statement now enables you to create overlapping constraints. This means that variables in a SAS data set are part of both a primary key definition and a foreign key definition.
- The MODIFY statement provides the CORRECTENCODING= option to change the encoding indicator, which is stamped in the file's descriptor information, in order to match the actual encoding of the file's data.

The DOCUMENT Procedure

The new DOCUMENT procedure enables you to customize or modify your output hierarchy and replay your output to different destinations without rerunning the PROC or DATA step. For complete information, see *SAS Output Delivery System: User's Guide*.

The EXPORT Procedure

The EXPORT procedure now enables you to

- export to Microsoft Excel 2002 spreadsheets and Microsoft Access 2002 tables. The new data sources are available for the Windows operating environment on 32-bit platforms if your site has a license for SAS/ACCESS Interface to PC File Formats.
- specify SAS data set options in the DATA= argument when you are exporting to all data sources except for delimited, comma-separated, and tab-delimited external files. For example, if the data set that you are exporting has an assigned password, use the ALTER=, PW=, READ=, or WRITE= data set option. To export only data that meets a specified condition, use the WHERE= data set option.
- identify a specific spreadsheet in a workbook by specifying the SHEET= option. Exporting to multiple sheets is available for Microsoft Excel 97, 2000, and 2002 spreadsheets for the Windows operating environment on 32-bit platforms if your site has a license for the SAS/ACCESS Interface to PC File Formats.

The FCMP Procedure (Experimental)

The new FCMP procedure enables you to create, test, and store SAS functions and subroutines for use by other SAS procedures.

For more information about the FCMP procedure, go to **`http://support.sas.com/documentation/onlinedoc`**. Select **`Base SAS`** from the Product-Specific Documentation list.

The FONTREG Procedure

The new FONTREG procedure enables you to add system fonts to the SAS registry.

The FORMAT Procedure

- The maximum length for character format names is now 31. The maximum length for numeric format names is now 32.
- The maximum length for character informat names is now 30. The maximum length for numeric informat names is now 31.

The FREQ Procedure

In the PROC FREQ statement, the new NLEVELS option displays a table that shows the number of levels for each variable that is named in the TABLES statement(s).

The WEIGHT statement now has the ZEROS option to include observations with zero weight values. The frequency and crosstabulation tables will display any levels that correspond to observations with 0 weights. PROC FREQ includes levels with 0 weights in the chi-square goodness-of-fit test for one-way tables, in the binomial computations for one-way tables, and in the computation of kappa statistics for two-way tables.

The following new options are available in the TABLES statement:

- The CONTENTS= option enables you to specify the text for the HTML contents file links to crosstabulation tables.
- The BDT option enables you to request Tarone's adjustment in the Breslow-Day test for homogeneity of odds ratios when you use the CMH option to compute the Breslow-Day test for stratified 2×2 tables.
- The NOWARN option suppresses the log warning message that the asymptotic chi-square test might not be valid when more than 20% of the table cells have expected frequencies less than 5.
- The CROSSLIST option displays crosstabulation tables in ODS column format. This option creates a table that has a table definition that you can customize with the TEMPLATE procedure.

Additionally, the FREQ procedure now produces exact confidence limits for the common odds ratio and related tests.

The IMPORT Procedure

The IMPORT procedure now enables you to

- import Microsoft Excel 2002 spreadsheets and Microsoft Access 2002 tables. The new data sources are available for the Windows operating environment on 32-bit platforms if your site has a license for SAS/ACCESS Interface to PC File Formats.
- specify SAS data set options in the OUT= argument when you are importing from all data sources except for delimited, comma-separated, and tab-delimited external files. For example, in order to assign a password for a resulting SAS data set, use the ALTER=, PW=, READ=, or WRITE= data set option. To import only data that meets a specified condition, use the WHERE= data set option.

The MEANS and SUMMARY Procedures

The new THREADS|NOTHREADS option enables or prevents the activation of multi-threaded processing.

When you format class variables with user-defined formats that are created with the MULTILABEL and NOTSORTED options, specifying the MLF, PRELOADFMT, and ORDER=DATA options together in the CLASS statement now orders the procedure output according to the label order in the format definition.

The MIGRATE Procedure

The new MIGRATE procedure is available specifically for migrating a SAS data library from a previous release to the most recent release. For migration, PROC MIGRATE offers benefits that PROC COPY does not. For PROC MIGRATE documentation, see the Migration Community at **http://support.sas.com/rnd/migration**.

The PROTO Procedure

The PROTO procedure, which has been available in SAS Risk Dimensions software, is now a Base SAS procedure. The PROTO procedure enables you to register, in batch, external functions that are written in the C or C++ programming languages for use in SAS programs and C-language structures and types. For PROC PROTO documentation, go to **http://support.sas.com/documentation/onlinedoc**. Select **Base SAS** from the Product-Specific Documentation list.

The PRTDEF Procedure

There are 15 new variables to control the default printer settings.

The PRTEXP Procedure

The new PRTEXP procedure enables you to write attributes, which are used by PROC PRTDEF to define a printer, either to a SAS data set or to the SAS log. With this capability you can replicate and modify those attributes easily.

The PWENCODE Procedure

The new PWENCODE procedure enables you to encode a password. You can use the encoded password in place of plain-text passwords in SAS programs that access relational database management systems (RDBMSs) and SAS servers (such as the SAS Metadata Server).

The REGISTRY Procedure

The REGISTRY procedure has three new options:

- The LISTREG option lists the contents of the registry in the log.
- The COMPAREREG1 and COMPAREREG2 options are used together to compare two registries. The results appear in the log.

The REPORT Procedure

The REPORT procedure has the following new features:

- The new THREADS|NOTHREADS option enables or prevents the activation of multi-threaded processing.
- Numeric class variables that do not have a format assigned to them are automatically formatted with the BEST12. format.
- PROC REPORT now writes the value **_PAGE_** for the _BREAK_ variable in the output data set for observations that are derived from a COMPUTE BEFORE _PAGE_ or COMPUTE AFTER _PAGE_ statement.

The SORT Procedure

The SORT procedure has the following new options:

- The DATECOPY option copies to the output data set the SAS internal date and time when the input data set was created, and the SAS internal date and time when it was last modified prior to the sort.
- The DUPOUT= option specifies an output data set that contains duplicate observations.

- The OVERWRITE option deletes the input data set before the replacement output data set is populated with observations.
- The THREADS|NOTHREADS option enables or prevents the activation of multi-threaded sorting.

The SQL Procedure

The SQL procedure has the following new features:

- The PROC SQL statement now has a THREADS | NOTHREADS option. THREADS enables PROC SQL to take advantage of the new parallel processing capabilities in SAS when performing sorting operations.
- There are new DICTIONARY tables, new columns in existing DICTIONARY tables, and SASHELP views of the new tables. For DICTIONARY.TABLES and SASHELP.VTABLE, if a table is read-protected with a password, the only information that is listed for that table is the library name, member name, member type, and type of password protection; all other information is set to missing.
- You can now reference a permanent SAS data set by its physical filename.
- When using the INTO clause to assign values to a range of macro variables, you can now specify leading zeroes in the macro variable names.
- PROC SQL now supports TRANSCODE=YES|NO as a column modifier.

The SYLK Procedure (Experimental)

The new SYLK procedure enables you to read an external SYLK-formatted spreadsheet into SAS, including data, formulas, and formats. You can also use PROC SYLK as a batch spreadsheet, using programming statements to manipulate data, perform calculations, generate summaries, and format the output.

For more information about the SYLK procedure, go to **http://support.sas.com/documentation/onlinedoc**. Select **Base SAS** from the Product-Specific Documentation list.

The TABULATE Procedure

The TABULATE procedure has the following new features:

- The new THREADS|NOTHREADS option enables or prevents the activation of multi-threaded processing.
- Available statistics include upper and lower confidence limits, skewness, and kurtosis. PROC TABULATE now supports the ALPHA= option, which enables you to specify a confidence level.
- Numeric class variables that do not have a format assigned to them are automatically formatted with the BEST12. format.
- The new FORMAT_PRECEDENCE and STYLE_PRECEDENCE options in the TABLE statement enable you to specify which formats and styles (defined for the column, row, or page dimensions) are applied.

Additionally, when you format class variables with user-defined formats that are created with the MULTILABEL and NOTSORTED options, specifying the MLF, PRELOADFMT, and ORDER=DATA options together in the CLASS statement now orders the procedure output according to the label order in the format definition.

The TEMPLATE Procedure

The TEMPLATE procedure now enables you to customize or create your own markup language for your output. For complete information, see *SAS Output Delivery System: User's Guide*.

The TIMEPLOT Procedure

The TIMEPLOT procedure now supports the SPLIT= option, which enables you to specify a character at which labels will be split into multiple lines.

The UNIVARIATE Procedure

The UNIVARIATE procedure has the following new features:

- The LOWER= and NOUPPER= suboptions in the KERNEL option in the HISTOGRAM statement specify the lower and upper bounds for fitted kernel density curves.
- The FRONTREF option in the HISTOGRAM statement draws reference lines in front of the histogram bars instead of behind them.

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	SUMMARY*
CHART*	PRINT	TABULATE*
FREQ*	REPORT*	TIMEPLOT
MEANS*	SQL*	

* 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, cross-tabulations, 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 basic utility operations. They create, edit, sort, and transpose data sets, create and restore transport data sets, create user-defined formats, and provide basic file maintenance such as to copy, append, and compare data sets:

APPEND	EXPORT	PWENCODE
BMDP*	FONTREG	PRTEXP
CATALOG	FORMAT	REGISTRY
CIMPORT	FSLIST	RELEASE*
COMPARE	IMPORT	SORT
CONTENTS	OPTIONS	SOURCE*
CONVERT*	OPTLOAD	SQL
COPY	OPTSAVE	TAPECOPY*
CPORT	PDS*	TAPELABEL*
CV2VIEW@	PDSCOPY*	TEMPLATE+
DATASETS	PMENU	TRANSPOSE
DBCSTAB#	PRINTTO	TRANTAB#
DOCUMENT+	PRTDEF	

* See the SAS documentation for your operating environment for a description of these procedures.
\+ See *SAS Output Delivery System: User's Guide* for a description of these procedures.
@ See *SAS/ACCESS for Relational Databases: Reference* for a description of this procedure.
\# See *SAS National Language Support (NLS): User's Guide* for a description of these procedures.

Report-Writing Procedures

Table 1.1 on page 5 lists report-writing procedures according to the type of report.

Table 1.1 Report-Writing Procedures by Task

To produce...	Use this procedure...	Which...
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.
	REPORT	combines features of the PRINT, MEANS, and TABULATE procedures with features of the DATA step in a single report writing tool that can produce a variety of reports; can also create an output data set.
	SQL	computes descriptive statistics for one or more SAS data sets or DBMS tables; can produce a printed report or create a SAS data set.
	TABULATE	produces descriptive statistics in a tabular format; can produce stub-and-banner reports (multidimensional tables with descriptive statistics); can also create an output data set.
Miscellaneous highly formatted reports		
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

Table 1.2 on page 6 lists statistical procedures according to task. Table A1.1 on page 1341 lists the most common statistics and the procedures that compute them.

Table 1.2 Elementary Statistical Procedures by Task

To produce...	Use this procedure...	Which...
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.
	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 cross-tabulation tables	FREQ	produces one-way to *n*-way tables; reports frequency counts; computes chi-square tests; computes tests and measures of association and agreement for two-way to *n*-way cross-tabulation tables; can compute exact tests and asymptotic tests; can create output data sets.
	TABULATE	produces one-way and two-way cross-tabulation 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 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.

To produce...	Use this procedure...	Which...
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, box plot, and normal probability plot.

* 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 may 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. See “Quantiles” on page 555 for more information.

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 may require substantial computer resources. A more efficient way to compute statistics within groups without sorting is to use a CLASS statement with one of the following procedures: MEANS, SUMMARY, or TABULATE.

Additional Information about the Statistical Procedures

Appendix 1, “SAS Elementary Statistics Procedures,” on page 1339 lists standard keywords, statistical notation, and formulas for the statistics that base SAS procedures compute frequently. The individual statistical procedures discuss the statistical concepts that are useful to interpret the output of a procedure.

Utility Procedures

Table 1.3 on page 8 groups utility procedures according to task.

Table 1.3 Utility Procedures by Task

To perform these utility tasks...	Use this procedure...	Which...
Supply information	COMPARE	compares the contents of two SAS data sets.
	CONTENTS	describes the contents of a SAS data library or specific library members.
	OPTIONS	lists the current values of all SAS system options.
	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	saves 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	FSLIST	browses external files such as files that contain SAS source lines or SAS procedure output.
	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.
	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.
Manage SAS files	APPEND	appends one SAS data set to the end of another.
	BMDP[*]	invokes a BMDP program to analyze data in a SAS data set.

To perform these utility tasks...	Use this procedure...	Which...
	CATALOG	manages SAS catalog entries.
	CIMPORT	restores a transport sequential file that PROC CPORT creates (usually under 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 data 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 under 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.
	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.
	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 under the z/OS environment.
Control windows	PMENU	creates customized pull-down menus for SAS applications.
Miscellaneous	PWENCODE	encodes passwords for use in SAS programs.

* See the SAS documentation for your operating environment for a description of these procedures.
\+ See *SAS Output Delivery System: User's Guide* for a description of these procedures.
@ See *SAS/ACCESS for Relational Databases: Reference* for a description of this procedure.
\# See *SAS National Language Support (NLS): User's Guide* for a description of these procedures.

Brief Descriptions of Base SAS Procedures

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

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

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, nonwindowing 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 data 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 data 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 data library.

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

COPY procedure
copies an entire SAS data 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 data 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. See *SAS/ACCESS for Relational Databases: Reference* for details.

DATASETS procedure
: lists, copies, renames, and deletes SAS files and SAS generation groups, manages indexes, and appends SAS data sets in a SAS data 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.

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. See *SAS Output Delivery System: User's Guide* for details.

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

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, tests and measures of association and of agreement for two-way to *n*-way cross-tabulation 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.

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

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.

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. See the SAS documentation for your operating environment for more information.

PDSCOPY procedure
: copies partitioned data sets from disk to tape, disk to disk, tape to tape, or tape to disk. See the SAS documentation for your operating environment for more information.

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.

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.

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. See the SAS documentation for this operating environment for more information.

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.

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. See the SAS documentation for your operating environment for more information.

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 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. See the SAS documentation for your operating environment for more information.

TAPELABEL procedure
: lists the label information of an IBM standard-labeled tape volume under the z/OS environment. See the SAS documentation for this operating environment for more information.

TEMPLATE procedure
: customizes ODS output for an entire SAS job or a single ODS output object. See *SAS Output Delivery System: User's Guide* for details.

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.

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.

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 data library. The WORK data library is defined automatically at the beginning of the SAS session and is automatically deleted 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 data library, unless you specify a USER data library (see "USER Data Library" on page 17). For example, the following PROC PRINT steps are equivalent. The second PROC PRINT step assumes that the DEBATE data set is in the WORK data 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. See *SAS Language Reference: Dictionary* for complete documentation.

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 data library*. A SAS data library is an external storage location that stores SAS data sets in your operating environment. A LIBNAME statement associates the libref with the SAS data library. In the following PROC PRINT step, PROCLIB is the libref and EMP is the SAS data set within the library:

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

USER Data Library

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

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

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

SAS System Options

Some SAS system option settings affect procedure output. The following are the SAS system 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 *SAS Language Reference: Dictionary*.

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

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=	SORTSEQ=
IDXNAME=	SPILL=
IDXWHERE=	TOBSNO=
IN=	TYPE=
INDEX=	WHERE=
KEEP=	WHEREUP=
LABEL=	WRITE=

For a complete description of SAS data set options, see *SAS Language Reference: Dictionary*.

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 but the ODS statement, refer to *SAS Language Reference: Dictionary*. For information about the ODS statement, refer to "Output Delivery System" on page 32 and to *SAS Output Delivery System: User's Guide*.

Procedure Concepts

Input Data Sets

Many base 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 *SAS Language Reference: Dictionary*.

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 eliminate 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 base SAS 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

#BY-specification<.suffix>

BY-specification
: is one of the following:

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:

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:

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

1. creates a data set, GROC, that contains data for stores from four regions. Each store has four departments. See "GROC" on page 1388 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; ❶
   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;  ❷
   by region department;
run;
options nobyline nodate pageno=1
        linesize=64 pagesize=20;  ❸
proc chart data=groc;  ❹
   by region department;
   vbar manager / type=sum sumvar=sales;
   title1 'This chart shows #byval2 sales';
   title2 'in the #byval(region)..';
run;
options byline;   ❺
```

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

```
                This chart shows Canned sales                1
                     in the Northwest.

        Sales Sum

        400 +        *****                   *****
            |        *****                   *****
        300 +        *****                   *****
            |        *****       *****       *****
        200 +        *****       *****       *****
            |        *****       *****       *****
        100 +        *****       *****       *****
            |        *****       *****       *****
            --------------------------------------------
                    Aikmann     Duncan     Jeffreys

                                Manager
```

```
                 This chart shows Meat sales                 2
                     in the Northwest.

        Sales Sum

        75 +        *****       *****
           |        *****       *****
        60 +        *****       *****
           |        *****       *****
        45 +        *****       *****
           |        *****       *****
        30 +        *****       *****       *****
           |        *****       *****       *****
        15 +        *****       *****       *****
           |        *****       *****       *****
           --------------------------------------------
                   Aikmann     Duncan     Jeffreys

                               Manager
```

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

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;  ❶
proc chart data=groc;  ❷
   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:

```
                      Regional Analysis                        1
                      for the Northwest

       Sales Mean

       300 +                              *****
           |                              *****
       200 +       *****                  *****
       100 +       *****       *****      *****
           |       *****       *****      *****
           -------------------------------------------
                  Aikmann     Duncan     Jeffreys

                              Manager
```

Example: Inserting the Complete BY Line into a Title

This example inserts the complete BY line into the title. The program

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:

```
        Information for Region=Northwest Department=Canned      1

         Sales Sum

         400 +       *****                   *****
             |       *****                   *****
         300 +       *****                   *****
             |       *****       *****       *****
         200 +       *****       *****       *****
             |       *****       *****       *****
         100 +       *****       *****       *****
             |       *****       *****       *****
             -------------------------------------------
                    Aikmann     Duncan     Jeffreys

                                Manager
```

```
         Information for Region=Northwest Department=Meat       2

         Sales Sum

         75 +       *****       *****
            |       *****       *****
         60 +       *****       *****
            |       *****       *****
         45 +       *****       *****
            |       *****       *****
         30 +       *****       *****       *****
            |       *****       *****       *****
         15 +       *****       *****       *****
            |       *****       *****       *****
            -------------------------------------------
                   Aikmann     Duncan     Jeffreys

                               Manager
```

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 simply 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.

Notation	Meaning
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. △

See *SAS Language Reference: Concepts* for complete documentation.

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 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. (See "PROCLIB.PAYROLL" on page 1395 for the DATA step that creates this data set.) 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.

```
libname proclib 'SAS-data-library';


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;
```

This is a partial printing of PROCLIB.PAYROLL:

```
                         PROCLIB.PAYROLL                         1
                    First 10 Observations Only

   Id
 Number   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                   1
                        Each Job Code

                     The MEANS Procedure

                 Analysis Variable : Salary

                                N
Jobcode                       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               1
           (Using a Format that Groups the Job Codes)

                      The MEANS Procedure

                  Analysis Variable : Salary

                          N
   Jobcode              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                1
                  with the Variable Salary

  Id
Number   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                         1

                    The MEANS Procedure

                 Analysis Variable : GPA

                          N
          Year          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 see if 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 variable(s) have the same formatted value, then the values are grouped into different BY groups. This results in two BY groups with the same formatted value. Further, if different and *consecutive* internal values of the BY variable(s) 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 456. △

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 base SAS software.

Example 9 on page 767 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. Appendix 2, "Operating Environment-Specific Procedures," on page 1375 contains a table with additional information. These procedures are described in more detail in the SAS documentation for operating environments.

Statistic Descriptions

Table 2.1 on page 31 identifies common descriptive statistics that are available in several Base SAS procedures. See “Keywords and Formulas” on page 1340 for more detailed information about available statistics and theoretical information.

Table 2.1 Common Descriptive Statistics That Base Procedures Calculate

Statistic	Description	Procedures
confidence intervals		FREQ, MEANS/SUMMARY, TABULATE, UNIVARIATE
CSS	corrected sum of squares	CORR, MEANS/SUMMARY, REPORT, SQL, TABULATE, UNIVARIATE
CV	coefficient of variation	MEANS/SUMMARY, REPORT, SQL, TABULATE, UNIVARIATE
goodness-of-fit tests		FREQ, UNIVARIATE
KURTOSIS	kurtosis	MEANS/SUMMARY, TABULATE, UNIVARIATE
MAX	largest (maximum) value	CORR, MEANS/SUMMARY, REPORT, SQL, TABULATE, UNIVARIATE
MEAN	mean	CORR, MEANS/SUMMARY, REPORT, SQL, TABULATE, UNIVARIATE
MEDIAN	median (50^{th} percentile)	CORR (for nonparametric correlation measures), MEANS/SUMMARY, TABULATE, UNIVARIATE
MIN	smallest (minimum) value	CORR, MEANS/SUMMARY, REPORT, SQL, TABULATE, UNIVARIATE
MODE	most frequent value (if not unique, the smallest mode is used)	UNIVARIATE
N	number of observations on which calculations are based	CORR, FREQ, MEANS/SUMMARY, REPORT, SQL, TABULATE, UNIVARIATE
NMISS	number of missing values	FREQ, MEANS/SUMMARY, REPORT, SQL, TABULATE, UNIVARIATE
NOBS	number of observations	MEANS/SUMMARY, UNIVARIATE
PCTN	the percentage of a cell or row frequency to a total frequency	REPORT, TABULATE
PCTSUM	the percentage of a cell or row sum to a total sum	REPORT, TABULATE
Pearson correlation		CORR
percentiles		FREQ, MEANS/SUMMARY, REPORT, TABULATE, UNIVARIATE
RANGE	range	CORR, MEANS/SUMMARY, REPORT, SQL, TABULATE, UNIVARIATE

Statistic	Description	Procedures
robust statistics	trimmed means, Winsorized means	UNIVARIATE
SKEWNESS	skewness	MEANS/SUMMARY, TABULATE, UNIVARIATE
Spearman correlation		CORR
STD	standard deviation	CORR, MEANS/SUMMARY, REPORT, SQL, TABULATE, UNIVARIATE
STDERR	the standard error of the mean	MEANS/SUMMARY, REPORT, SQL, TABULATE, UNIVARIATE
SUM	sum	CORR, MEANS/SUMMARY, REPORT, SQL, TABULATE, UNIVARIATE
SUMWGT	sum of weights	CORR, MEANS/SUMMARY, REPORT, SQL, TABULATE, UNIVARIATE
tests of location		UNIVARIATE
USS	uncorrected sum of squares	CORR, MEANS/SUMMARY, REPORT, SQL, TABULATE, UNIVARIATE
VAR	variance	CORR, MEANS/SUMMARY, REPORT, SQL, TABULATE, UNIVARIATE

Computational Requirements for Statistics

The following requirements are computational requirements for the statistics that are listed in Table 2.1 on page 31. 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

What Is the 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.

Prior to Version 7, most SAS procedures generated output that was designed for a traditional line-printer. This type of output has limitations that prevents 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. Prior to 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.

Gallery of ODS Samples

Introduction to the ODS Samples

This section shows you samples of the different kinds of formatted output that you can produce with ODS. The input file contains sales records for TruBlend Coffee Makers, a company that distributes coffee machines.

Listing Output

Traditional SAS output is Listing output. You do not need to change your SAS programs to create listing output. By default, you continue to create this kind of output even if you also create a type of output that contains more formatting.

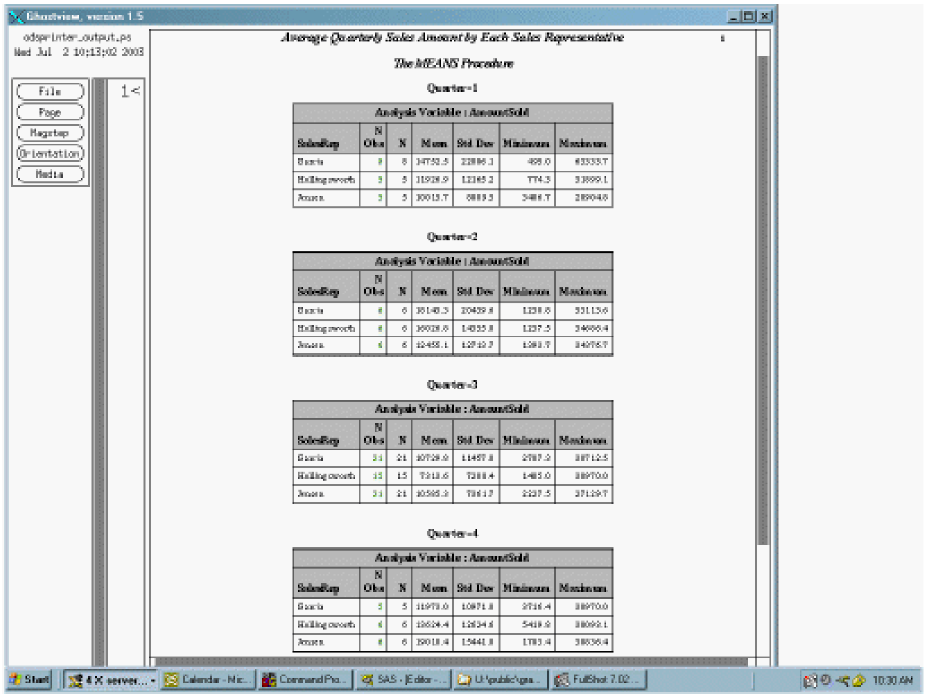
Output 2.1 Listing Output

```
        Average Quarterly Sales Amount by Each Sales Representative        1

--------------------------------- Quarter=1 ----------------------------------

                            The MEANS Procedure

                      Analysis Variable : AmountSold

                     N
SalesRep           Obs      N         Mean      Std Dev      Minimum      Maximum
___________________________________________________________________________________
Garcia               8      8      14752.5      22806.1        495.0      63333.7

Hollingsworth        5      5      11926.9      12165.2        774.3      31899.1

Jensen               5      5      10015.7       8009.5       3406.7      20904.8
___________________________________________________________________________________

        Average Quarterly Sales Amount by Each Sales Representative        2

--------------------------------- Quarter=2 ----------------------------------

                            The MEANS Procedure

                      Analysis Variable : AmountSold

                     N
SalesRep           Obs      N         Mean      Std Dev      Minimum      Maximum
___________________________________________________________________________________
Garcia               6      6      18143.3      20439.6       1238.8      53113.6

Hollingsworth        6      6      16026.8      14355.0       1237.5      34686.4

Jensen               6      6      12455.1      12713.7       1393.7      34376.7
___________________________________________________________________________________

        Average Quarterly Sales Amount by Each Sales Representative        3

--------------------------------- Quarter=3 ----------------------------------

                            The MEANS Procedure

                      Analysis Variable : AmountSold

                     N
SalesRep           Obs      N         Mean      Std Dev      Minimum      Maximum
___________________________________________________________________________________
Garcia              21     21      10729.8      11457.0       2787.3      38712.5

Hollingsworth       15     15       7313.6       7280.4       1485.0      30970.0

Jensen              21     21      10585.3       7361.7       2227.5      27129.7
___________________________________________________________________________________

        Average Quarterly Sales Amount by Each Sales Representative        4

--------------------------------- Quarter=4 ----------------------------------

                            The MEANS Procedure

                      Analysis Variable : AmountSold

                     N
SalesRep           Obs      N         Mean      Std Dev      Minimum      Maximum
___________________________________________________________________________________
Garcia               5      5      11973.0      10971.8       3716.4      30970.0

Hollingsworth        6      6      13624.4      12624.6       5419.8      38093.1

Jensen               6      6      19010.4      15441.0       1703.4      38836.4
___________________________________________________________________________________
```

PostScript Output

With ODS, you can produce output in PostScript format.

Display 2.1 PostScript Output Viewed with Ghostview



HTML Output

With ODS, you can produce output in HTML (Hypertext Markup Language.) You can browse these files with Internet Explorer, Netscape, or any other browser that fully supports the HTML 3.2 tagset.

Note: To create HTML 4.0 tagsets, use the ODS HTML4 statement. In SAS 9, the ODS HTML statement generates HTML 3.2 tagsets. In future releases of SAS, the ODS HTML statement will support the most current HTML tagsets available. △

Display 2.2 HTML Output Viewed with Microsoft Internet Explorer

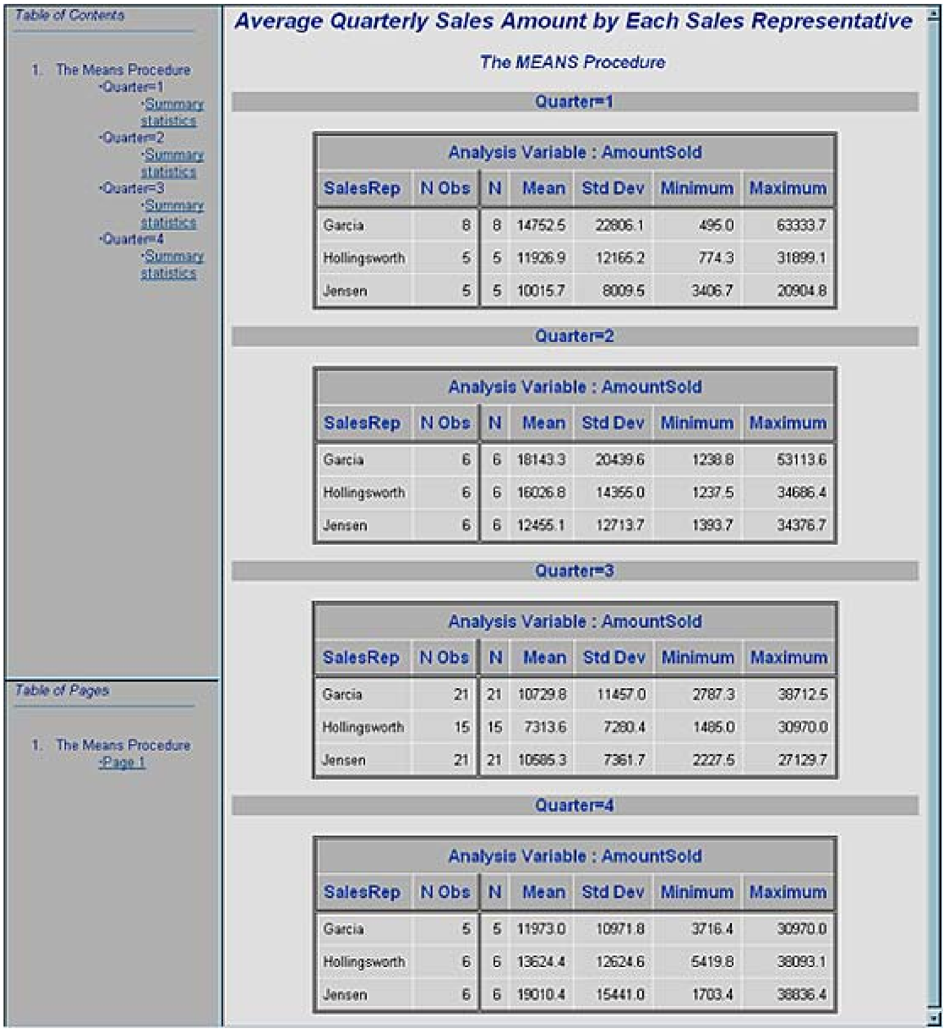


Table of Contents

1. The Means Procedure
   - Quarter=1
     - Summary statistics
   - Quarter=2
     - Summary statistics
   - Quarter=3
     - Summary statistics
   - Quarter=4
     - Summary statistics

Table of Pages

1. The Means Procedure
   - Page 1

Average Quarterly Sales Amount by Each Sales Representative

The MEANS Procedure

Quarter=1

Analysis Variable : AmountSold						
SalesRep	N Obs	N	Mean	Std Dev	Minimum	Maximum
Garcia	8	8	14752.5	22806.1	495.0	63333.7
Hollingsworth	5	5	11926.9	12165.2	774.3	31899.1
Jensen	5	5	10015.7	8009.5	3406.7	20904.8

Quarter=2

Analysis Variable : AmountSold						
SalesRep	N Obs	N	Mean	Std Dev	Minimum	Maximum
Garcia	6	6	18143.3	20439.6	1238.8	53113.6
Hollingsworth	6	6	16026.8	14355.0	1237.5	34686.4
Jensen	6	6	12455.1	12713.7	1393.7	34376.7

Quarter=3

Analysis Variable : AmountSold						
SalesRep	N Obs	N	Mean	Std Dev	Minimum	Maximum
Garcia	21	21	10729.8	11457.0	2787.3	38712.5
Hollingsworth	15	15	7313.6	7280.4	1485.0	30970.0
Jensen	21	21	10585.3	7361.7	2227.5	27129.7

Quarter=4

Analysis Variable : AmountSold						
SalesRep	N Obs	N	Mean	Std Dev	Minimum	Maximum
Garcia	5	5	11973.0	10971.8	3716.4	30970.0
Hollingsworth	6	6	13624.4	12624.6	5419.8	38093.1
Jensen	6	6	19010.4	15441.0	1703.4	38836.4

RTF Output

With ODS, you can produce RTF (Rich Text Format) output which is used with Microsoft Word.

Display 2.3 RTF Output Viewed with Microsoft Word

Average Quarterly Sales Amount by Each Sales Representative

The MEANS Procedure

Quarter=1

Analysis Variable : AmountSold						
Sales Rep	N Obs	N	Mean	Std Dev	Minimum	Maximum
Garcia	8	8	14752.49	22806.09	495.0000000	63333.65
Hollingsworth	5	5	11926.94	12165.18	774.2500000	31899.10
Jensen	5	5	10015.70	8009.46	3406.70	20904.75

Quarter=2

Analysis Variable : AmountSold						
Sales Rep	N Obs	N	Mean	Std Dev	Minimum	Maximum
Garcia	6	6	18143.26	20439.58	1238.80	53113.55
Hollingsworth	6	6	16026.76	14355.04	1237.50	34686.40
Jensen	6	6	12455.10	12713.73	1393.65	34376.70

Quarter=3

Analysis Variable : AmountSold						
Sales Rep	N Obs	N	Mean	Std Dev	Minimum	Maximum
Garcia	21	21	10729.82	11457.05	2787.30	38712.50
Hollingsworth	15	15	7313.62	7280.44	1485.00	30970.00
Jensen	21	21	10585.29	7361.68	2227.50	27129.72

Quarter=4

Analysis Variable : AmountSold						
Sales Rep	N Obs	N	Mean	Std Dev	Minimum	Maximum
Garcia	5	5	11973.00	10971.77	3716.40	30970.00
Hollingsworth	6	6	13624.42	12624.61	5419.75	38093.10
Jensen	6	6	19010.42	15440.98	1703.35	38836.38

PDF Output

With ODS, you can produce output in PDF (Portable Document Format), which can be viewed with the Adobe Acrobat Reader.

Display 2.4 PDF Output Viewed with Adobe Acrobat Reader

Average Quarterly Sales Amount by Each Sales Representative 1

The MEANS Procedure

Quarter=1

Analysis Variable : AmountSold						
SalesRep	**N Obs**	**N**	**Mean**	**Std Dev**	**Minimum**	**Maximum**
Garcia	8	8	14752.49	22806.09	495.0000000	63333.65
Hollingsworth	5	5	11926.94	12165.18	774.2500000	31899.10
Jensen	5	5	10015.70	8009.46	3406.70	20904.75

Quarter=2

Analysis Variable : AmountSold						
SalesRep	**N Obs**	**N**	**Mean**	**Std Dev**	**Minimum**	**Maximum**
Garcia	6	6	18143.26	20439.58	1238.80	53113.55
Hollingsworth	6	6	16026.76	14355.04	1237.50	34686.40
Jensen	6	6	12455.10	12713.73	1393.65	34376.70

Quarter=3

Analysis Variable : AmountSold						
SalesRep	**N Obs**	**N**	**Mean**	**Std Dev**	**Minimum**	**Maximum**
Garcia	21	21	10729.82	11457.05	2787.30	38712.50
Hollingsworth	15	15	7313.62	7280.44	1485.00	30970.00
Jensen	21	21	10585.29	7361.68	2227.50	27129.72

Quarter=4

Analysis Variable : AmountSold						
SalesRep	**N Obs**	**N**	**Mean**	**Std Dev**	**Minimum**	**Maximum**
Garcia	5	5	11973.00	10971.77	3716.40	30970.00
Hollingsworth	6	6	13624.42	12624.61	5419.75	38093.10
Jensen	6	6	19010.42	15440.98	1703.35	38836.38

XML Output

With ODS, you can produce output that is tagged with XML (Extensible Markup Language) tags.

Output 2.2 XML Output file

```
<?xml version="1.0" encoding="windows-1252"?>

<odsxml>
<head>
<meta operator="user"/>
</head>
<body>
<proc name="Print">
<label name="IDX"/>
<title class="SystemTitle" toc-level="1">US Census of Population and Housing</title>
<branch name="Print" label="The Print Procedure" class="ContentProcName" toc-level="1">
<leaf name="Print" label="Data Set SASHELP.CLASS" class="ContentItem" toc-level="2">
<output name="Print" label="Data Set SASHELP.CLASS" clabel="Data Set SASHELP.CLASS">
<output-object type="table" class="Table">

  <style>
    <border spacing="1" padding="7" rules="groups" frame="box"/>
  </style>
<colspecs columns="6">
<colgroup>
<colspec name="1" width="2" align="right" type="int"/>
</colgroup>
<colgroup>
<colspec name="2" width="7" type="string"/>
<colspec name="3" width="1" type="string"/>
<colspec name="4" width="2" align="decimal" type="double"/>
<colspec name="5" width="4" align="decimal" type="double"/>
<colspec name="6" width="5" align="decimal" type="double"/>
</colgroup>
</colspecs>
<output-head>
<row>
<header type="string" class="Header" row="1" column="1">
<value>Obs</value>
</header>
<header type="string" class="Header" row="1" column="2">
<value>Name</value>
</header>
<header type="string" class="Header" row="1" column="3">
<value>Sex</value>
</header>
<header type="string" class="Header" row="1" column="4">
<value>Age</value>
</header>
<header type="string" class="Header" row="1" column="5">
<value>Height</value>
</header>
<header type="string" class="Header" row="1" column="6">
<value>Weight</value>
</header>
</row>
</output-head>
<output-body>
<row>
<header type="double" class="RowHeader" row="2" column="1">
<value> 1</value>
</header>
<data type="string" class="Data" row="2" column="2">
<value>Alfred</value>
</data>
... more xml tagged output...
<
/odsxml>
```

Commonly Used ODS Terminology

data component
is a form, similar to a SAS data set, that contains the results (numbers and characters) of a DATA step or PROC step that supports ODS.

table definition
is a set of instructions that describes how to format the data. This description includes but is not limited to

- the order of the columns
- text and order of column headings
- formats for data
- font sizes and font faces.

output object
is an object that contains both the results of a DATA step or PROC step and information about how to format the results. An output object has a name, label, and path. For example, the Basic Statistical Measurement table generated from the UNIVARIATE procedure is an output object. It contains the data component and formatted presentation of the mean, median, mode, standard deviation, variance, range, and interquartile range.

Note: Although many output objects include formatting instructions, not all of them do. In some cases the output object consists of only the data component. △

ODS destinations
are designations that produce specific types of output. ODS supports a number of destinations, including the following:

LISTING
produces traditional SAS output (monospace format).

Markup Languages
produce SAS output that is formatted using one of many different markup languages such as HTML (Hypertext Markup Language), XML (Extensible Markup Language), and LaTeX that you can access with a web browser. SAS supplies many markup languages for you to use ranging from DOCBOOK to TROFF. You can specify a markup language that SAS supplies or create one of your own and store it as a user-defined markup language.

DOCUMENT
produces a hierarchy of output objects that enables you to produce multiple ODS output formats without rerunning a PROC or DATA step and gives you more control over the structure of the output.

OUTPUT
produces a SAS data set.

Printer Family
produces output that is formatted for a high-resolution printer such as a PostScript (PS), PDF, or PCL file.

RTF
produces output that is formatted for use with Microsoft Word.

ODS output
ODS output consists of formatted output from any of the ODS destinations. For example, the OUTPUT destination produces SAS data sets; the LISTING destination produces listing output; the HTML destination produces output that is formatted in Hypertext Markup Language.

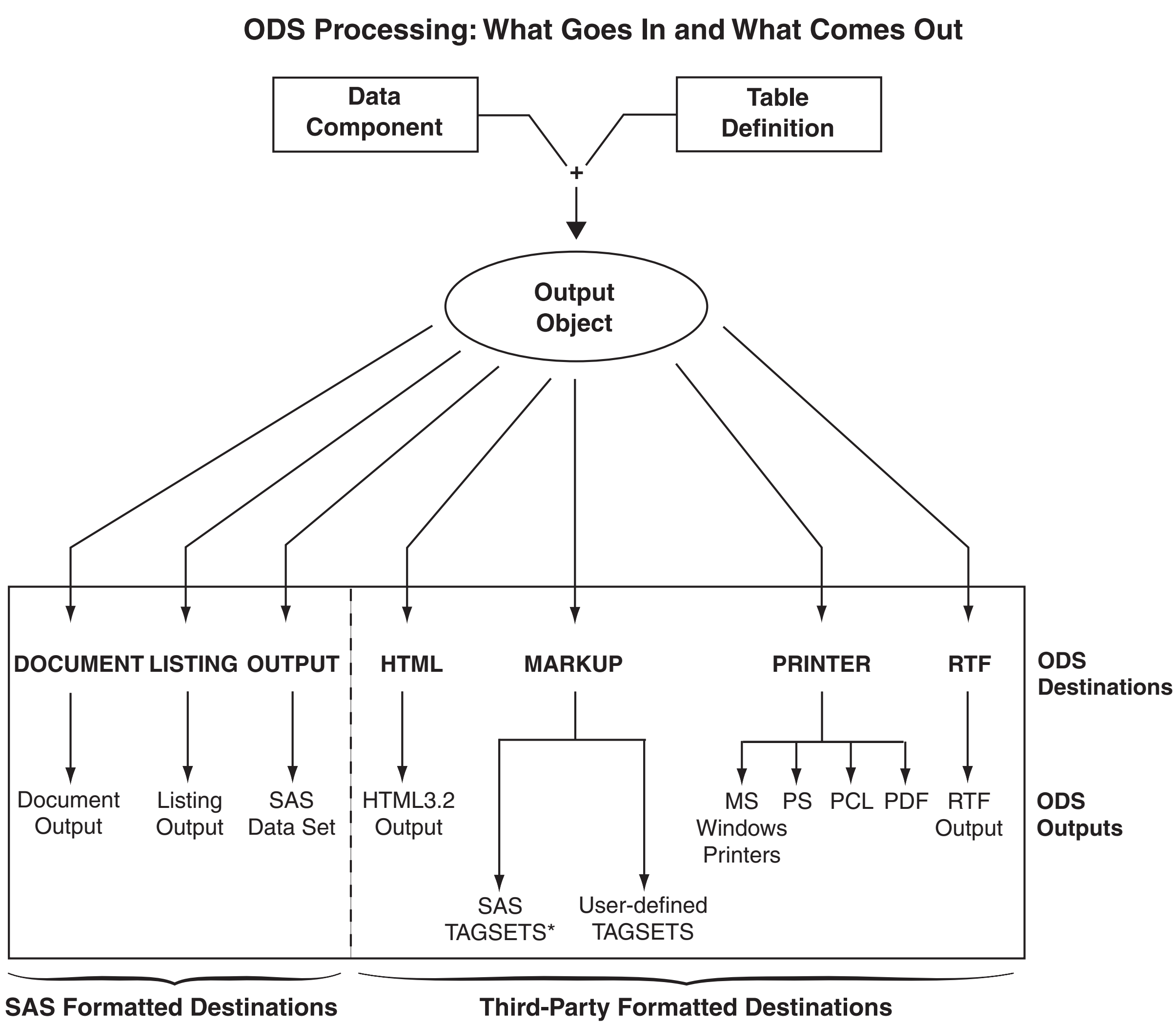
How Does ODS Work?

Components of SAS Output

The PROC or DATA step supplies raw data and the name of the table definition that contains the formatting instructions, and ODS formats the output. You can use the Output Delivery System to format output from individual procedures and from the DATA step in many different forms other than the default SAS listing output.

The following figure shows how SAS produces ODS output.

Figure 2.1 ODS Processing: What Goes in and What Comes Out



* List of Tagsets that SAS Supplies and Supports

Table 2.2 * List of Tagsets that SAS Supplies and Supports

CHTML	HTML4	SASIOXML	SASXMOH
CSVALL	HTMLCSS	SASREPORT	SASXMOIM
DEFAULT	IMODE	SASXML	SASXMOR
DOCBOOK	PHTML	SASXMOG	WML
EVENT_MAP			

* List of Tagsets that SAS Supplies but Does Not Support

Table 2.3 Additional Tagsets that SAS Supplies but Does Not Support

COLORLATEX	LATEX	SHORT_MAP	TPL_STYLE_MAP
CSV	LATEX2	STYLE_DISPLAY	TROFF
CSVBYLINE	NAMEDHTML	STYLE_POPUP	WMLOLIST
GRAPH	ODSSTYLE	TEXT_MAP	
GTABLEAPPLET	PYX	TPL_STYLE_LIST	

CAUTION:

These tagsets are experimental tagsets. Do not use these tagsets in production jobs. △

Features of ODS

ODS is designed to overcome the limitations of traditional SAS output and to make it easy to access and create the new formatting options. ODS provides a method of delivering output in a variety of formats, and makes the formatted output easy to access.

Important features of ODS include the following:

- ODS combines raw data with one or more table definitions to produce one or more output objects. These objects can be sent to any or all ODS destinations. You control the specific type of output from ODS by selecting an ODS destination. The currently available ODS destinations can produce
  - traditional monospace output
  - an output data set
  - an ODS document that contains a hierarchy file of the output objects
  - output that is formatted for a high-resolution printer such as PostScript and PDF
  - output that is formatted in various markup languages such as HTML
  - RTF output that is formatted for use with Microsoft Word.
- ODS provides table definitions that define the structure of the output from SAS procedures and from the DATA step. You can customize the output by modifying these definitions, or by creating your own.
- ODS provides a way for you to choose individual output objects to send to ODS destinations. For example, PROC UNIVARIATE produces five output objects. You can easily create HTML output, an output data set, traditional listing output, or printer output from any or all of these output objects. You can send different output objects to different destinations.

- In the SAS windowing environment, ODS stores a link to each output object in the Results folder in the Results window.
- Because formatting is now centralized in ODS, the addition of a new ODS destination does not affect any procedures or the DATA step. As future destinations are added to ODS, they will automatically become available to the DATA step and all procedures that support ODS.
- With ODS, you can produce output for numerous destinations from a single source, but you do not need to maintain separate sources for each destination. This feature saves you time and system resources by enabling you to produce multiple kinds of output with a single run of your procedure or data query.

What Are the ODS Destinations?

Overview of ODS Destination Categories

ODS enables you to produce SAS procedure and DATA step output to many different destinations. ODS destinations are organized into two categories.

SAS Formatted destinations
: produce output that is controlled and interpreted by SAS, such as a SAS data set, SAS output listing, or an ODS document.

Third-Party Formatted destinations
: produce output which enables you to apply styles, markup languages, or enables you to print to physical printers using page description languages. For example, you can produce output in PostScript, HTML, XML, or a style or markup language that you created.

The following table lists the ODS destination categories, the destination that each category includes, and the formatted output that results from each destination.

Table 2.4 Destination Category Table

Category	Destinations	Results
SAS Formatted	DOCUMENT	ODS document
	LISTING	SAS output listing
	OUTPUT	SAS data set
Third-Party Formatted	HTML	HTML file for online viewing
	MARKUP	markup language tagsets
	PRINTER	printable output in one of three different formats: PCL, PDF, or PS (PostScript)
	RTF	output written in Rich Text Format for use with Microsoft Word 2000

As future destinations are added to ODS, they automatically will become available to the DATA step and to all procedures that support ODS.

Definition of Destination-Independent Input

Destination-independent input means that one destination can support a feature even though another destination does not support it. In this case, the request is ignored by the destination that does not support it. Otherwise, ODS would support a small subset of features that are only common to all destinations. If this was true, then it would be difficult to move your reports from one output format to another output format. ODS provides many output formatting options, so that you can use the appropriate format for the output that you want. It is best to use the appropriate destination suited for your purpose.

The SAS Formatted Destinations

The SAS formatted destinations create SAS entities such as a SAS data set, a SAS output listing, or an ODS document. The statements in the ODS SAS Formatted category create the SAS entities.

The three SAS formatted destinations are:

DOCUMENT Destination

The DOCUMENT destination enables you to restructure, navigate, and replay your data in different ways and to different destinations as you like without needing to rerun your analysis or repeat 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. Unlike other ODS destinations, the DOCUMENT destination has a GUI interface. However, everything that you can do through the GUI, you can also do with batch commands using the ODS DOCUMENT statement and the DOCUMENT procedure.

Prior to SAS 9, each procedure or DATA step produced output that was sent to each destination that you specified. While you could always send your output to as many destinations as you wanted, you needed to rerun your procedure or data query if you decided to use a destination that you had not originally designated. The DOCUMENT destination eliminates the need to rerun procedures or repeat data queries by enabling you to store your output objects and replay them to different destinations.

LISTING Destination

The LISTING destination produces output that looks the same as the traditional SAS output. The LISTING destination is the default destination that opens when you start your SAS session. Thus ODS is always being used, even when you do not explicitly invoke ODS.

The LISTING destination enables you to produce traditional SAS output with the same look and presentation as it had in previous versions of SAS.

Because most procedures share some of the same table definitions, the output is more consistent. For example, if you have two different procedures producing an ANOVA table, they will both produce it in the same way because each procedure uses the same template to describe the table. However, there are four procedures that do not use a default table definition to produce their output: PRINT procedure, REPORT procedure, TABULATE procedure, and FREQ procedure's n-way tables. These procedures use the structure that you specified in your program code to define their tables.

OUTPUT Destination

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. The output data sets can be used for further analysis, or for sophisticated reports in which you want to combine similar statistics across different data sets into a single table. You can easily access and process your output data sets using all of the SAS data set features. For example, you can access your output data using variable names and perform WHERE-expression processing just as you would process data from any other SAS data set.

The Third-Party Formatted Destinations

The third-party formatted destinations enable you to apply styles to the output objects that are used by applications other than SAS. For example, these destinations support attributes such as "font" and "color."

Note: For a list of style elements and valid values, see the style elements table in the *SAS Output Delivery System: User's Guide*. △

The four categories of third-party formatted destinations are:

- *HTML (Hypertext Markup Language)*

  The HTML destination produces HTML 3.2-compatible output. You can, however, produce (HTML 4 stylesheet) output using the HTML4 tagsets.

  The HTML destination can create some or all of the following:

  - an HTML file (called the *body file*) that contains the results from the procedure
  - a table of contents that links to the body file
  - a table of pages that links to the body file
  - a frame that displays the table of contents, the table of pages, and the body file.

  The body file is required with all ODS HTML output. If you do not want to link to your output, then you do not have to create a table of contents, a table of pages, or a frame file. However, if your output is very large, you might want to create a table of contents and a table of pages for easier reading and transversing through your file.

  The HTML destination is intended only for on-line use, not for printing. To print hard-copies of the output objects, use the PRINTER destination.

- *Markup Languages (MARKUP) Family*

  Just as table definitions describe how to lay out a table, and style attributes describe the style of the output, *tagsets* describe how to produce a markup language output. You can use a tagset that SAS supplies or you can create your own using the TEMPLATE procedure. Like table definitions and style attributes, tagsets enable you to modify your markup language output. For example, each variety of XML can be specified as a new tagset. SAS supplies you with a collection of XML tagsets and enables you to produce a customized variety of XML. The important point is that you can implement a tagset that SAS supplies or a customized tagset that you created without having to wait for the next release of SAS. With the addition of modifying and creating your own tagsets by using PROC TEMPLATE, now you have greater flexibility in customizing your output.

  Because the MARKUP destination is so flexible, you can use either the SAS tagsets or a tagset that you created. For a complete listing of the markup language tagsets that SAS supplies, see the section on listing tagset names in the *SAS Output Delivery System: User's Guide*. To learn how to define your own tagsets, see the section on methods to create your own tagsets in the *SAS Output Delivery System: User's Guide*.

  The MARKUP destination cannot replace ODS PRINTER or ODS RTF destinations because it cannot do text measurement. Therefore, it cannot produce output for a page description language or a hybrid language like RTF which requires all of the text to be measured and placed at a specific position on the page.

- *PRINTER Family*

  The PRINTER destination produces output for

  - printing to physical printers such as Windows printers under Windows, PCL, and PostScript printers on other operating systems
  - producing portable PostScript, PCL, and PDF files.

  The PRINTER destinations produce ODS output that contain page description languages: they describe precise positions where each line of text, each rule, and each graphical element are to be placed on the page. In general, you cannot edit or alter these formats. Therefore, the output from ODS PRINTER is intended to be the final form of the report.

- *Rich Text Format (RTF)*

  RTF produces output for Microsoft Word. While there are other applications that can read RTF files, the RTF output might not work successfully with them.

  The RTF destination enables you to view and edit the RTF output. ODS does not define the "vertical measurement," meaning that SAS does not determine the optimal place to position each item on the page. For example, page breaks are not always fixed, so when you edit your text, you do not want your RTF output tables to split at inappropriate places. Your tables can remain whole and intact on one page or can have logical breaks where you specified.

  However, because Microsoft Word needs to know the widths of table columns and it cannot adjust tables if they are too wide for the page, ODS measures the width of the text and tables (horizontal measurement). Therefore, all the column widths can be set properly by SAS and the table can be divided into panels if it is too wide to fit on a single page.

  In short, when producing RTF output for input to Microsoft Word, SAS determines the horizontal measurement and Microsoft Word controls the vertical measurement. Because Microsoft Word can determine how much room there is on the page, your tables will display consistently as you specified even after you modified your RTF file.

What Controls the Formatting Features of Third-Party Formats?

All of the formatting features that control the appearance of the third-party formatted destinations beyond what the LISTING destination can do are controlled by two mechanisms:

- ODS statement options
- ODS style attributes

The ODS statement options control three features:

1. Features that are specific to a given destination, such as stylesheets for HTML.
2. Features that are global to the document, such as AUTHOR and table of contents generation.
3. Features that we expect users to change on each document, such as the output file name.

The ODS style attributes control the way that individual elements are created. Attributes are aspects of a given style, such as type face, weight, font size, and color. The values of the attributes collectively determine the appearance of each part of the document to which the style is applied. With style attributes, it is unnecessary to insert destination-specific code (such as raw HTML) into the document. Each output destination will interpret the attributes that are necessary to generate the presentation of the document. Because not all destinations are the same, not all attributes can be interpreted by all destinations. Style attributes that are incompatible with a selected destination are ignored. For example, PostScript does not support active links, so the URL= attribute is ignored when producing PostScript output.

ODS Destinations and System Resources

ODS destinations can be open or closed. You open and close a destination with the appropriate ODS statement. When a destination is open, ODS sends the output objects to it. An open destination uses system resources even if you use the selection and exclusion features of ODS to select or exclude all objects from the destination. Therefore, to conserve resources, close unnecessary destinations. For more information about using each destination, see the topic on ODS statements in the *SAS Output Delivery System: User's Guide*.

By default, the LISTING destination is open and all other destinations are closed. Consequently, if you do nothing, your SAS programs run and produce listing output looking just as they did in previous releases of SAS before ODS was available.

What Are Table Definitions, Table Elements, and Table Attributes?

A *table definition* describes how to generate the output for a tabular output object. (Most ODS output is tabular.) A table definition determines the order of column headers and the order of variables, as well the overall look of the output object that uses it. For information about customizing the table definition, see the topic on the TEMPLATE procedure in the *SAS Output Delivery System: User's Guide*.

In addition to the parts of the table definition that order the headers and columns, each table definition contains or references *table elements*. A table element is a collection of table attributes that apply to a particular header, footer, or column. Typically, a *table attribute* specifies something about the data rather than about its presentation. For example, FORMAT specifies the SAS format, such as the number of decimal places. However, some table attributes describe presentation aspects of the data, such as how many blank characters to place between columns.

Note: The attributes of table definitions that control the presentation of the data have no effect on output objects that go to the LISTING or OUTPUT destination. However, the attributes that control the structure of the table and the data values do affect listing output. △

For information on table attributes, see the section on table attributes in the *SAS Output Delivery System: User's Guide*.

What Are Style Definitions, Style Elements, and Style Attributes?

To customize the output at the level of your entire output stream in a SAS session, you specify a style definition. A *style definition* describes how to generate the presentation aspects (color, font face, font size, and so on) of the entire SAS output. A style definition determines the overall look of the documents that use it.

Each style definition is composed of *style elements*. A style element is a collection of style attributes that apply to a particular part of the output. For example, a style element may contain instructions for the presentation of column headers, or for the presentation of the data inside the cells. Style elements may also specify default colors and fonts for output that uses the style definition.

Each *style attribute* specifies a value for one aspect of the presentation. For example, the BACKGROUND= attribute specifies the color for the background of an HTML table or for a colored table in printed output. The FONT_STYLE= attribute specifies whether to use a Roman or an italic font. For information on style attributes, see the section on style attributes in the *SAS Output Delivery System: User's Guide*.

Note: Because style definitions control the presentation of the data, they have no effect on output objects that go to the LISTING or OUTPUT destination. △

What Style Definitions Are Shipped with SAS Software?

Base SAS software is shipped with many style definitions. To see a list of these styles, you can view them in the SAS Explorer Window, use the TEMPLATE procedure, or use the SQL procedure.

- *SAS Explorer Window:*

  To display a list of the available styles using the SAS Explorer Window, follow these steps:

  1. From any window in an interactive SAS session, select

     View ► Results

  2. In the Results window, select

     View ► Templates

  3. In the Templates window, select and open **`Sashelp.tmplmst`**.
  4. Select and open the **`Styles`** folder, which contains a list of available style definitions. If you want to view the underlying SAS code for a style definition, then select the style and open it.

     Operating Environment Information: For information on navigating in the Explorer window without a mouse, see the section on "Window Controls and General Navigation" in the SAS documentation for your operating environment. △

- *TEMPLATE Procedure:*

  You can also display a list of the available styles by submitting the following PROC TEMPLATE statements:

  ```
  proc template;
     list styles;
  run;
  ```

- *SQL Procedure:*

  You can also display a list of the available styles by submitting the following PROC SQL statements:

  ```
  proc sql;
  select * from styles.style-name;
  ```

  The *style–name* is the name of any style from the template store (for example, **`styles.default`** or **`styles.beige`**).

For more information on how ODS destinations use styles and how you can customize styles, see the section on the DEFINE STYLE statement in the *SAS Output Delivery System: User's Guide*.

How Do I Use Style Definitions with Base SAS Procedures?

- Most Base SAS Procedures

Most Base SAS procedures that support ODS use one or more table definitions to produce output objects. These table definitions include definitions for table elements: columns, headers, and footers. Each table element can specify the use of one or more style elements for various parts of the output. These style elements cannot be specified within the syntax of the procedure, but you can use customized styles for the ODS destinations that you use. For more information about customizing tables and styles, see the TEMPLATE procedure in the *SAS Output Delivery System: User's Guide*.

□ The PRINT, REPORT and TABULATE Procedures

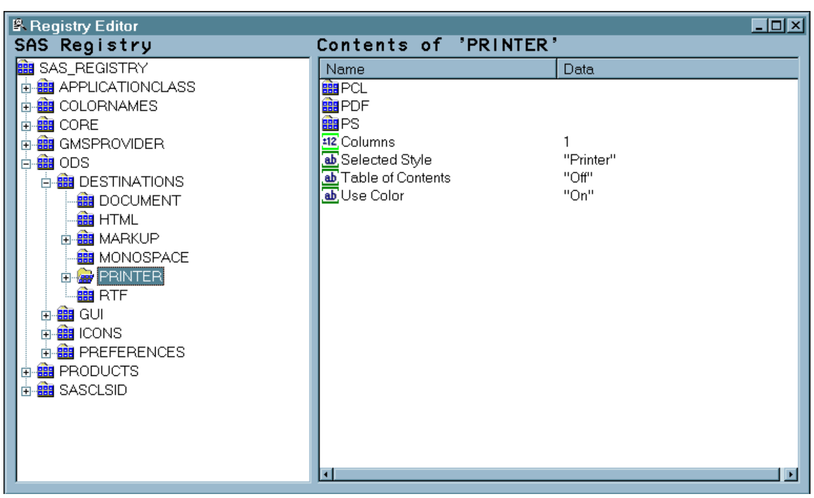
The PRINT, REPORT and TABULATE procedures provide a way for you to access table elements from the procedure step itself. Accessing the table elements enables you to do such things as specify background colors for specific cells, change the font face for column headers, and more. The PRINT, REPORT, and TABULATE procedures provide a way for you to customize the markup language and printed output directly from the procedure statements that create the report. For more information about customizing the styles for these procedures, see the *Base SAS Procedures Guide*.

Changing SAS Registry Settings for ODS

Overview of ODS and the SAS Registry

The SAS registry is the central storage area for configuration data that ODS uses. This configuration data is stored in a hierarchical form, which works in a similar manner to the way directory-based file structures work under UNIX, Windows, VMS, and the z/OS UNIX system. However, the SAS registry uses keys and subkeys as the basis for its structure, instead of using directories and subdirectories, like similar file systems in DOS or UNIX. A key is a word or a text string that refers to a particular aspect of SAS. Each key may be a place holder without values or subkeys associated with it, or it may have many subkeys with associated values. For example, the ODS key has DESTINATIONS, GUI, ICONS, and PREFERENCES subkeys. A subkey is a key inside another key. For example, PRINTER is a subkey of the DESTINATIONS subkey.

Display 2.5 SAS Registry of ODS Subkeys



Changing Your Default HTML Version Setting

By default, the SAS registry is configured to generate HTML4 output when you specify the ODS HTML statement. To permanently change the default HTML version, you can change the setting of the HTML version in the SAS registry.

CAUTION:

If you make a mistake when you modify the SAS registry, then your system might become unstable or unusable. You will not be warned if an entry is incorrect. Incorrect entries can cause errors, and can even prevent you from bringing up a SAS session. For more information about how to configure the SAS registry, see the SAS registry section in *SAS Language Reference: Concepts*. △

To change the default setting of the HTML version in the SAS registry:

1 Select

Solutions ▶ Accessories ▶ Registry Editor

or

Issue the command **REGEDIT**.

2 Select

ODS ▶ Default HMTL Version

3 Select

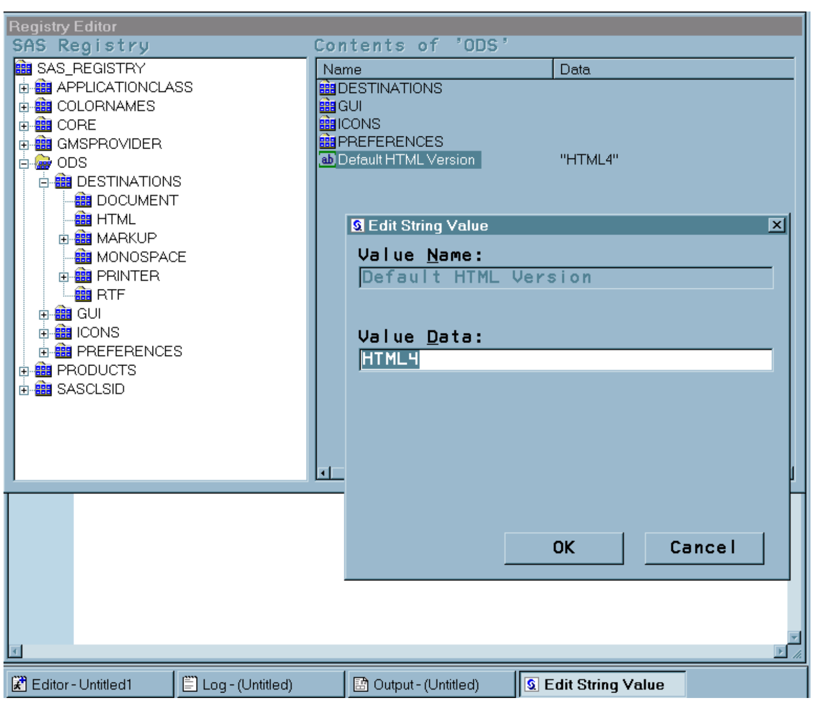
Edit ▶ Modify

or

Click the right mouse button and select **MODIFY**. The Edit String Value window appears.

4 Type the HTML version in the **Value Data** text box and select **OK**.

Display 2.6 SAS Registry Showing HTML Version Setting



Changing ODS Destination Default Settings

ODS destination subkeys are stored in the SAS registry. To change the values for these destinations subkeys:

1. Select

   ODS ▶ Destinations

2. Select a destination subkey
3. Select a subkey in the Contents of window
4. Select

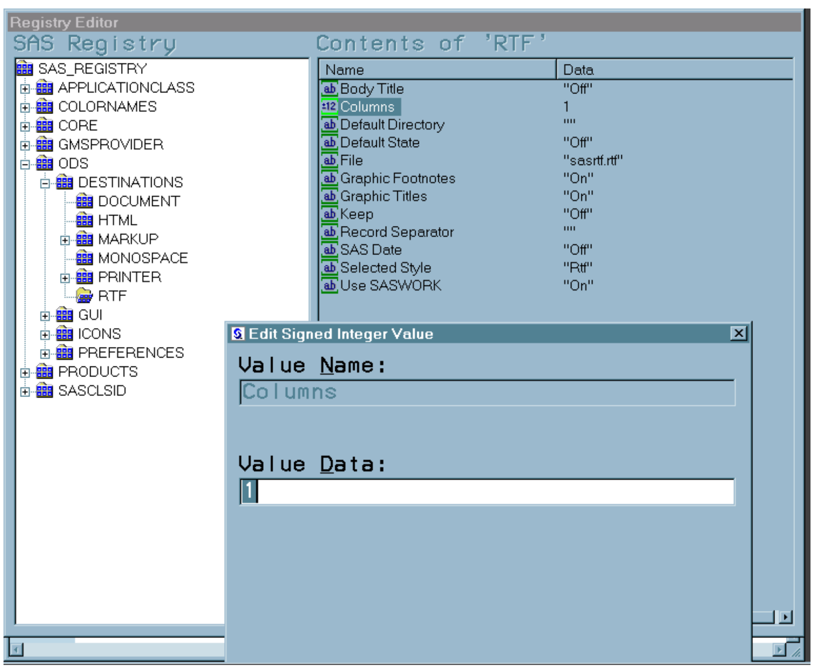
   Edit ▶ Modify

   or

   Click the right mouse button and select **MODIFY**.

5. Type in the Value Data entry into the Edit Value String or Edit Signed Integer Value window and select **OK**.

Display 2.7 Registry Editor Window



Customized ODS Output

SAS Output

By default, ODS output is formatted according to instructions that a PROC step or DATA step defines. However, ODS provides ways for you to customize the output. You can customize the output for an entire SAS job, or you can customize the output for a single output object.

Selection and Exclusion Lists

You can specify which output objects that you want to produce by selecting or excluding them in a list. For each ODS destination, ODS maintains either a selection list or an exclusion list. A selection list is a list of output objects that are sent to the destination. An exclusion list is a list of output objects that are excluded from the destination. ODS also maintains an overall selection list or an overall exclusion list. You can use these lists to control which output objects go to the specified ODS destinations.

To see the contents of the lists use the ODS SHOW statement. The lists are written to the SAS log. The following table shows the default lists:

Table 2.5 Default List for Each ODS Destination

ODS Destination	Default List
OUTPUT	EXCLUDE ALL
All others	SELECT ALL

How Does ODS Determine the Destinations for an Output Object?

To specify an output object, you need to know which output objects your SAS program produces. The ODS TRACE statement writes to the SAS log a trace record that includes the path, the label, and other information about each output object that is produced. For more information, about the ODS TRACE statement see *SAS Output Delivery System: User's Guide*. You can specify an output object as any of the following:

- a full path. For example,

  ```
  Univariate.City_Pop_90.TestsForLocation
  ```

  is the full path of the output object.

- a partial path. A partial path consists of any part of the full path that begins immediately after a period (.) and continues to the end of the full path. For example, if the full path is

  ```
  Univariate.City_Pop_90.TestsForLocation
  ```

  then the partial paths are:

  ```
   City_Pop_90.TestsForLocation
   TestsForLocation
  ```

- a label that is enclosed in quotation marks.

  For example,

  ```
  "Tests For Location"
  ```

- a label path. For example, the label path for the output object is

  "The UNIVARIATE Procedure"."CityPop_90"."Tests For Location"

  Note: The trace record shows the label path only if you specify the LABEL option in the ODS TRACE statement. △

- a partial label path. A partial label path consists of any part of the label that begins immediately after a period (.) and continues to the end of the label. For example, if the label path is

  ```
  "The UNIVARIATE Procedure"."CityPop_90"."Tests For Location"
  ```

then the partial label paths are:

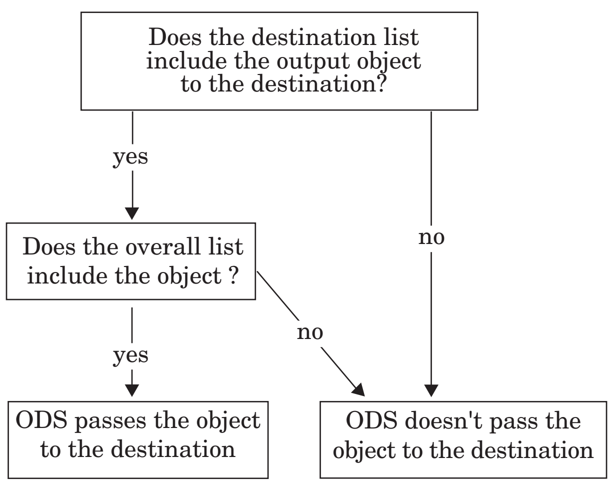
```
"CityPop_90"."Tests For Location"
"Tests For Location"
```

- a mixture of labels and paths.
- any of the partial path specifications, followed by a pound sign (#) and a number. For example, TestsForLocation#3 refers to the third output object that is named TestsForLocation.

As each output object is produced, ODS uses the selection and exclusion lists to determine which destination or destinations the output object will be sent to. The following figure illustrates this process:

Figure 2.2 Directing an Output Object to a Destination

For each destination, ODS first asks if the list for that destination includes the object. If it does not, ODS does not send the output object to that destination. If the list for that destination does include the object, ODS reads the overall list. If the overall list includes the object, ODS sends it to the destination. If the overall list does not include the object, ODS does not send it to the destination.



Note: Although you can maintain a selection list for one destination and an exclusion list for another, it is easier to understand the results if you maintain the same types of lists for all the destinations where you route output. △

Customized Output for an Output Object

For a procedure, the name of the table definition that is used for an output object comes from the procedure code. The DATA step uses a default table definition unless you specify an alternative with the TEMPLATE= suboption in the ODS option in the FILE statement. For more information, see the section on the TEMPLATE= suboption in the *SAS Output Delivery System: User's Guide*.

To find out which table definitions a procedure or the DATA step uses for the output objects, you must look at a trace record. To produce a trace record in your SAS log, submit the following SAS statements:

```
ods trace on;
your-proc-or-DATA-step
ods trace off;
```

Remember that not all procedures use table definitions. If you produce a trace record for one of these procedures, no definition appears in the trace record. Conversely, some procedures use multiple table definitions to produce their output. If you produce a trace record for one of these procedures, more than one definition appears in the trace record.

The trace record refers to the table definition as a template. For a detailed explanation of the trace record, see the section on the ODS TRACE statement in the *SAS Output Delivery System: User's Guide*.

You can use PROC TEMPLATE to modify an entire table definition. When a procedure or DATA step uses a table definition, it uses the elements that are defined or referenced in its table definition. In general, you cannot directly specify a table element for your procedure or DATA step to use without modifying the definition itself.

Note: Three Base SAS procedures, PROC PRINT, PROC REPORT and PROC TABULATE, do provide a way for you to access table elements from the procedure step itself. Accessing the table elements enables you to customize your report. For more information about these procedures, see the *Base SAS Procedures Guide* △

Summary of ODS

In the past, the term "output" has generally referred to the outcome of a SAS procedure and DATA step. With the advent of the Output Delivery System, "output" takes on a much broader meaning. ODS is designed to optimize output from SAS procedures and the DATA step. It provides a wide range of formatting options and greater flexibility in generating, storing, and reproducing SAS output.

Important features of ODS include the following:

- ODS combines raw data with one or more table definitions to produce one or more *output objects*. An output object tells ODS how to format the results of a procedure or DATA step.
- ODS provides table definitions that define the structure of the output from SAS procedures and from the DATA step. You can customize the output by modifying these definitions, or by creating your own definitions.
- ODS provides a way for you to choose individual output objects to send to ODS destinations.
- ODS stores a link to each output object in the Results folder for easy retrieval and access.
- As future destinations are added to ODS, they will automatically become available to the DATA step and all procedures that support ODS.

One of the main goals of ODS is to enable you to produce output for numerous destinations from a single source, without requiring separate sources for each destination. ODS supports many destinations:

DOCUMENT
: enables you to capture output objects from single run of the analysis and produce multiple reports in various formats whenever you want without re-running your SAS programs.

LISTING
: produces output that looks the same as the traditional SAS output.

HTML
: produces output for online viewing.

MARKUP
: produces output for markup language tagsets.

OUTPUT
: produces SAS output data sets, thereby eliminating the need to parse PROC PRINTTO output.

PRINTER
: produces presentation-ready printed reports.

RTF
: produces output suitable for Microsoft Word reports.

By default, ODS output is formatted according to instructions that the procedure or DATA step defines. However, ODS provides ways for you to customize the presentation of your output. You can customize the presentation of your SAS output, or you can customize the look of a single output object. ODS gives you greater flexibility in generating, storing, and reproducing SAS procedure and DATA step output with a wide range of formatting options.

CHAPTER 3

Statements with the Same Function in Multiple Procedures

Overview

Several statements are available and have the same function in a number of base SAS procedures. Some of the statements are fully documented in *SAS Language Reference: Dictionary*, 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. See *SAS Language Reference: Dictionary* for complete documentation.

BY
: orders the output according to the BY groups. See "BY" on page 58.

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. See *SAS Language Reference: Dictionary* for complete documentation.

FREQ
: treats observations as if they appear multiple times in the input data set. See "FREQ" on page 61.

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. See *SAS Language Reference: Dictionary* for complete documentation.

QUIT
: executes any statements that have not executed and ends the procedure. See "QUIT" on page 63.

WEIGHT
specifies weights for analysis variables in the statistical calculations. See "WEIGHT" on page 63.

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 68.

Statements

BY

Orders the output according to the BY groups.

See also: "Creating Titles That Contain BY-Group Information" on page 20

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 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*.

Options

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.

Note: 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 may need to use the NOTSORTED or DESCENDING option in the BY statement in the PROC step.

For more information on

- the BY statement, see *SAS Language Reference: Dictionary*.
- PROC SORT, see Chapter 43, “The SORT Procedure,” on page 1003.
- creating indexes, see “INDEX CREATE Statement” on page 345.

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(s) change.

This process can have unexpected results if, for instance, 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.

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. With the other procedures, the BY statement specifies how the data are 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 29.

```
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                   1
                             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

Treats observations as if they appear multiple times in the input data set.

Tip: 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/SUMMARY
- REPORT
- STANDARD
- TABULATE
- UNIVARIATE

Example

The data in this example represent 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                    1

                     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, and the total number of observations is 141 (the sum of the values of the frequency variable).

```
                    Average Course and Speed                   1

                      The MEANS Procedure

         Variable                   N            Mean
         ----------------------------------------
         Course                   141           49.28
         Speed                    141            8.06
         ----------------------------------------
```

QUIT

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

QUIT;

Procedures That Support the QUIT Statement

- CATALOG
- DATASETS
- PLOT
- PMENU
- SQL

WEIGHT

Specifies weights for analysis variables in the statistical calculations.

Tip: 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:

Weight value ...	The 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

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

Prior to 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.

The procedure substitutes the value of the WEIGHT variable for w_i, which appears in "Keywords and Formulas" on page 1340.

Procedures That Support the WEIGHT Statement

- CORR
- FREQ
- MEANS/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. See the PROC FREQ documentation in Volume 3 of this book for more information. △

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 come 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 *SAS/STAT User's Guide*. △

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          1

                      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;
```

In the first PROC MEANS step, the variance is an estimate of σ^2, where the variance of the ith observation is assumed to be $var\,(x_i) = \sigma^2/w_i$ and w_i is the weight for the ith observation. In the second PROC MEANS step, the computed variance is an estimate of $(n-1/n)\ \sigma^2/\overline{w}$, where $\overline{w}$ is the average weight. For large n, this is an approximate estimate of the variance of an observation with average weight.

```
           Weighted Analysis Using Default VARDEF=DF          1

                      The MEANS Procedure

                 Analysis Variable : ObjectSize

         N            Mean        Variance         Std Dev
        --------------------------------------------------
        20          31.088          20.678           4.547
        --------------------------------------------------
```

```
            Weighted Analysis Using VARDEF=WEIGHT            2

                     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                            4

------------------------- Distance=1.5 -------------------------

  Object                      Est_         Est_      Est_       Est_
   Size      Precision      SigmaSq       Sigma     VarObs     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 --------------------------

  Object                      Est_         Est_      Est_       Est_
   Size      Precision      SigmaSq       Sigma     VarObs     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 -------------------------

  Object                      Est_         Est_      Est_       Est_
   Size      Precision      SigmaSq       Sigma     VarObs     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 --------------------------

  Object                      Est_         Est_      Est_       Est_
   Size      Precision      SigmaSq       Sigma     VarObs     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 -------------------------

  Object                      Est_         Est_      Est_       Est_
   Size      Precision      SigmaSq       Sigma     VarObs     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

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 Language Reference: Dictionary* for more information on 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/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 on WHERE=, see *SAS Language Reference: Dictionary*.

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 29.

```
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                   1
                     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
```

PART 2

Procedures

PART 2

Procedures

CHAPTER
4

The 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

Reminder: You can use the ATTRIB, FORMAT, LABEL, and WHERE statements. See Chapter 3, "Statements with the Same Function in Multiple Procedures," on page 57 for details. You can also use any global statements. See "Global Statements" on page 18 for a list.

Reminder: You can use data set options with the BASE= and DATA= options. See "Data Set Options" on page 18 for a list.

Reminder: Complete documentation for the APPEND statement and the APPEND procedure is in "APPEND Statement" on page 313 .

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

CHAPTER

5

The 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

- 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

Output 5.1 illustrates the simplest kind of schedule calendar that you can produce. This calendar output displays activities that are planned by a banking executive. The following statements produce Output 5.1.

```
options nodate pageno=1 linesize=132 pagesize=60;

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

For the activities data set shown that is in this calendar, see Example 1 on page 114.

Output 5.1 Simple Schedule Calendar

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

```
                                                            The SAS System                                                            1

 -------------------------------------------------------------------------------------------------------------------------------
 |                                                                                                                             |
 |                                                         July  1996                                                          |
 |                                                                                                                             |
 |-----------------------------------------------------------------------------------------------------------------------------|
 |     Sunday      |     Monday      |     Tuesday     |    Wednesday    |    Thursday     |     Friday      |    Saturday     |
 |-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------|
 |                 |        1        |        2        |        3        |        4        |        5        |        6        |
 |                 |                 |                 |                 |                 |                 |                 |
 |                 |                 |                 |                 |                 |                 |                 |
 |                 |                 |                 |                 |                 |                 |                 |
 |                 |                 |                 |                 |                 |                 |                 |
 |                 |                 |                 |+=Interview/JW==+|                 |                 |                 |
 |                 |+Dist. Mtg./All=+|+====Mgrs. Meeting/District 6=====+|                 |+VIP Banquet/JW=+|                 |
 |-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------|
 |        7        |        8        |        9        |       10        |       11        |       12        |       13        |
 |                 |                 |                 |                 |                 |                 |                 |
 |                 |                 |                 |                 |                 |                 |                 |
 |                 |                 |                 |                 |                 |                 |                 |
 |                 |                 |                 |                 |+Planning Counci+|+=Seminar/White=+|                 |
 |                 |+=================Trade Show/Knox===================+|+====Mgrs. Meeting/District 7=====+|                 |
 |                 |+===============================Sales Drive/District 6================================+|                 |
 |-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------|
 |       14        |       15        |       16        |       17        |       18        |       19        |       20        |
 |                 |                 |                 |                 |                 |                 |                 |
 |                 |                 |                 |                 |                 |                 |                 |
 |                 |                 |                 |                 |                 |                 |                 |
 |                 |                 |                 |                 |+NewsLetter Dead+|+Co. Picnic/All=+|                 |
 |                 |                 |+==Dentist/JW===+|+Bank Meeting/1s+|+Planning Counci+|+=Seminar/White=+|                 |
 |                 |+===============================Sales Drive/District 7================================+|                 |
 |-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------|
 |       21        |       22        |       23        |       24        |       25        |       26        |       27        |
 |                 |                 |                 |                 |                 |                 |                 |
 |                 |                 |                 |                 |                 |                 |                 |
 |                 |                 |                 |                 |                 |                 |                 |
 |                 |                 |                 |                 |                 |                 |                 |
 |                 |                 |                 |+=Birthday/Mary=+|+======Close Sale/WYGIX Co.=======+|                 |
 |                 |+===============Inventors Show/Melvin===============+|+Planning Counci+|                 |                 |
 |-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------|
 |       28        |       29        |       30        |       31        |                 |                 |                 |
 |                 |                 |                 |                 |                 |                 |                 |
 |                 |                 |                 |                 |                 |                 |                 |
 |                 |                 |                 |                 |                 |                 |                 |
 |                 |                 |                 |                 |                 |                 |                 |
 |                 |                 |                 |                 |                 |                 |                 |
 |                 |                 |                 |                 |                 |                 |                 |
 -------------------------------------------------------------------------------------------------------------------------------
```

Advanced Schedule Calendar

Output 5.2 is an advanced schedule calendar produced by PROC CALENDAR. The statements that create this calendar

- 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 on page 127.

Output 5.2 Advanced Schedule Calendar

```
                                          Well Drilling Work Schedule: Combined Calendars                                          1

          ------------------------------------------------------------------------------------------------------------------
          |                                                                                                                |
          |                                                July  1996                                                      |
          |                                                                                                                |
          |----------------------------------------------------------------------------------------------------------------|
          |     Sunday     |     Monday     |    Tuesday     |   Wednesday    |    Thursday    |     Friday     |    Saturday    |
----------+----------------+----------------+----------------+----------------+----------------+----------------+----------------|
|         |                |       1        |       2        |       3        |       4        |       5        |       6        |
|.........|................|................|................|................|................|................|................|
| CAL1    |                |                |                |                |**Independence**|+Assemble Tank/>|                |
|         |                |                |                |                |                |+Lay Power Line>|                |
|         |                |+=============Drill Well/$1,000.00==============>|                |<Drill Well/$1,+|                |
|.........|................|................|................|................|................|................|................|
| 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/$2,000.00====+|                |+===Pour Foundation/$1,500.00===>|                |
|.........|................|................|................|................|................|................|................|
| CAL2    |                |<Excavate/$3,50>|****Vacation****|<Excavate/$3,50+|                |                |                |
|---------+----------------+----------------+----------------+----------------+----------------+----------------+----------------|
|         |       14       |       15       |       16       |       17       |       18       |       19       |       20       |
|.........|................|................|................|................|................|................|................|
| CAL1    |                |+==============================Install Pump/$500.00==============================+|                |
|         |                |<===========Pour Foundation/$1,500.00============+|                |+Install Pipe/$>|                |
|         |                |                |                |                |                |                |                |
|         |                |                |                |                |                |                |                |
|         |                |                |                |                |                |                |                |
|---------+----------------+----------------+----------------+----------------+----------------+----------------+----------------|
|         |       21       |       22       |       23       |       24       |       25       |       26       |       27       |
|.........|................|................|................|................|................|................|................|
| CAL1    |                |+=============================Erect Tower/$2,500.00==============================>|                |
|         |                |<====Install Pipe/$1,000.00=====+|                |                |                |                |
|         |                |                |                |                |                |                |                |
|         |                |                |                |                |                |                |                |
|         |                |                |                |                |                |                |                |
|---------+----------------+----------------+----------------+----------------+----------------+----------------+----------------|
|         |       28       |       29       |       30       |       31       |                |                |                |
|.........|................|................|................|................|................|................|................|
| CAL1    |                |<Erect Tower/$2+|                |                |                |                |                |
|         |                |                |                |                |                |                |                |
|         |                |                |                |                |                |                |                |
|         |                |                |                |                |                |                |                |
|         |                |                |                |                |                |                |                |
----------------------------------------------------------------------------------------------------------------------------------
```

Simple Summary Calendar

Output 5.3 shows a simple summary calendar that displays the number of meals served daily in a hospital cafeteria:

```
options nodate pageno=1 linesize=132 pagesize=60;

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 Output 5.3. The data set that is shown in this calendar is created in Example 7 on page 143.

Output 5.3 Simple Summary Calendar

```
                                        The SAS System                                         1

          -----------------------------------------------------------------------------------------
          |                                                                                       |
          |                                    December  1996                                     |
          |                                                                                       |
          |---------------------------------------------------------------------------------------|
          |   Sunday    |   Monday    |   Tuesday   |  Wednesday  |  Thursday   |   Friday    |  Saturday   |
          |-------------+-------------+-------------+-------------+-------------+-------------+-------------|
          |      1      |      2      |      3      |      4      |      5      |      6      |      7      |
          |             |             |             |             |             |             |             |
          |             |         123 |         188 |         123 |         200 |         176 |             |
          |             |         234 |         188 |         183 |         267 |         165 |             |
          |             |         238 |         198 |         176 |         243 |         177 |             |
          |-------------+-------------+-------------+-------------+-------------+-------------+-------------|
          |      8      |      9      |     10      |     11      |     12      |     13      |     14      |
          |             |             |             |             |             |             |             |
          |             |         178 |         165 |         187 |         176 |         187 |             |
          |             |         198 |         176 |         176 |         187 |         187 |             |
          |             |         187 |         187 |         231 |         222 |         123 |             |
          |-------------+-------------+-------------+-------------+-------------+-------------+-------------|
          |     15      |     16      |     17      |     18      |     19      |     20      |     21      |
          |             |             |             |             |             |             |             |
          |             |         176 |         156 |         198 |         178 |         165 |             |
          |             |         165 |           . |         143 |         198 |         176 |             |
          |             |         177 |         167 |         167 |         187 |         187 |             |
          |-------------+-------------+-------------+-------------+-------------+-------------+-------------|
          |     22      |     23      |     24      |     25      |     26      |     27      |     28      |
          |             |             |             |             |             |             |             |
          |             |         187 |             |             |             |             |             |
          |             |         187 |             |             |             |             |             |
          |             |         123 |             |             |             |             |             |
          |-------------+-------------+-------------+-------------+-------------+-------------+-------------|
          |     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 on page 137.

Syntax: CALENDAR Procedure

Required: You must use a START statement.

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

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

Tip: Supports the Output Delivery System. See “Output Delivery System” on page 32 for details.

ODS Table Name: Calendar

Reminder: 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)*;

The following table lists the statements and options available in the CALENDAR procedure according to function.

To do this	Use this statement
Create summary calendar	MEAN
	SUM
Create schedule calendar	DUR or FIN
Create multiple calendars	CALID
Specify holidays	HOLISTART
	HOLIDUR
	HOLIFIN
	HOLIVAR

To do this	Use this statement
Control display	OUTSTART
	OUTDUR
	OUTFIN
Specify grouping	BY
	CALID

PROC CALENDAR Statement

PROC CALENDAR <*option(s)*>;

To do this	Use this option
Specify data sets containing	
weekly work schedules	CALEDATA=
activities	DATA=
holidays	HOLIDATA=
unique shift patterns	WORKDATA=
Control printing	
display all months, even if no activities exist	FILL
define characters used for outlines, dividers, and so on	FORMCHAR=
specify the type of heading for months	HEADER=
display month and weekday names in local language (experimental)	LOCALE
specify how to show missing values	MISSING
suppress the display of Saturdays and Sundays	WEEKDAYS
Specify time or duration	
specify that START and FIN variables are in DATETIME format	DATETIME
specify the number of hours in a standard work day	DAYLENGTH=
specify the units of the DUR and HOLIDUR variables	INTERVAL=
Control summary information	
identify variables in the calendar	LEGEND
specify the type of mean to calculate	MEANTYPE=

Options

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, as described in “The Default Calendars” on page 103.

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

See also: “Calendar Data Set” on page 109

Featured in: Example 3 on page 122

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 also: “Activities Data Set” on page 107

Featured in: All examples. See “Examples: CALENDAR Procedure” on page 114

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.

Featured in: Example 3 on page 122

DAYLENGTH=*hours*
gives the number of hours in a standard working day. 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.

Interaction: 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.

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

Tip: The DAYLENGTH= option is useful when you use the DUR statement and your work schedule contains days of varying lengths, for example, a 5 half-day work week. 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.

Tip: 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 also: “Calendar Data Set” on page 109 for more information on 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.)

Featured in: Example 5 on page 134

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. Table 5.1 on page 87 shows the formatting characters that PROC CALENDAR uses. Figure 5.1 on page 88 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 that they are listed. For instance, the following option assigns an asterisk (*) to the twelfth position, assigns a single dash (-) to the thirteenth, and does not alter remaining characters:

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

These new settings change the activity line from this:

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

to this:

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

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 instance, 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
```

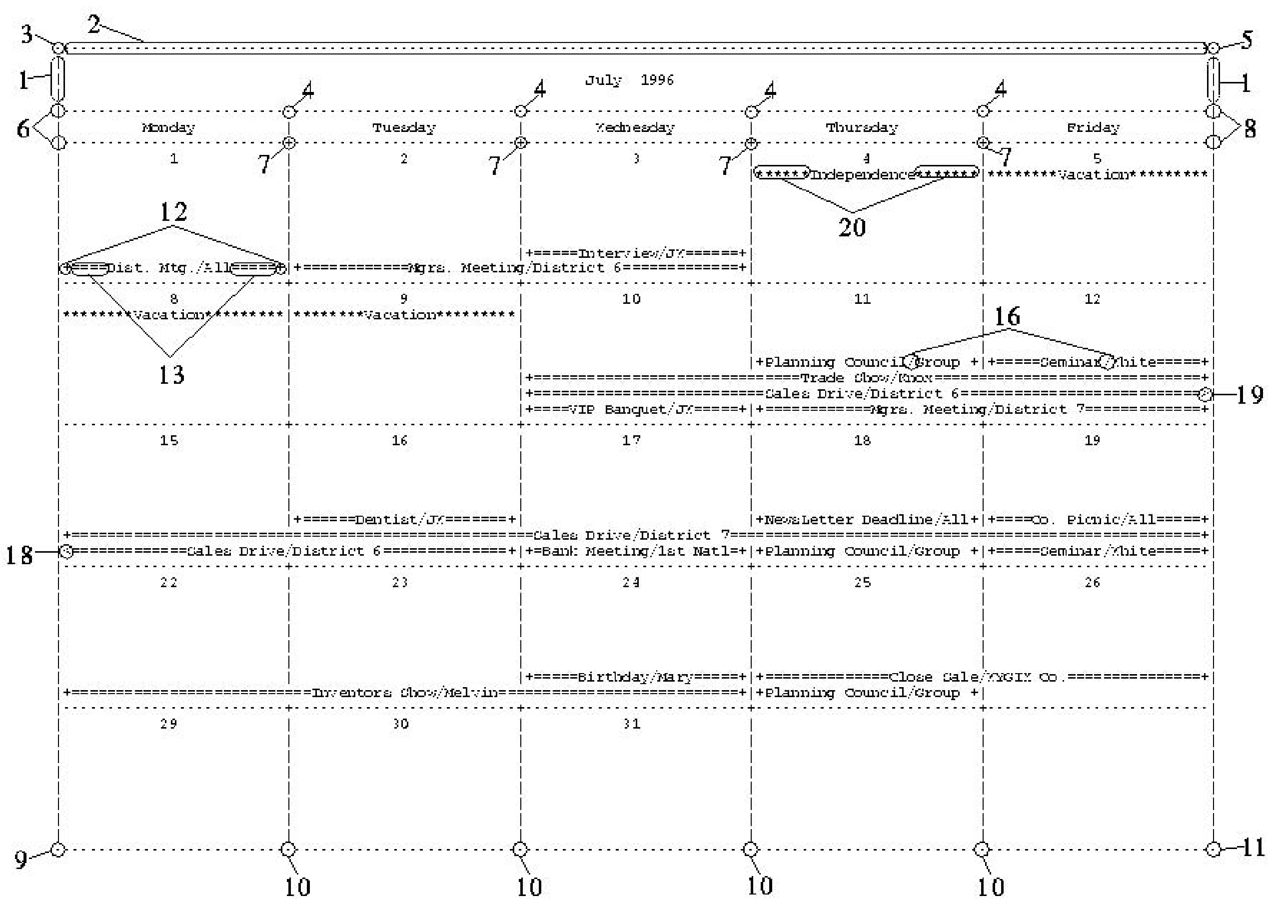
See also: For information on which hexadecimal codes to use for which characters, consult the documentation for your hardware.

Table 5.1 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

Position	Default	Used to draw
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
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

Figure 5.1 Formatting Characters in PROC CALENDAR Output



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 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 also: “Holidays Data Set” on page 108

Featured in: All examples. See “Examples: CALENDAR Procedure” on page 114

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 instance, 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 and 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 8 hours long, then a DUR value of 3.0 is treated as 24 hours.

Default: DAY

Interaction: In the absence of a CALEDATA= data set, PROC CALENDAR uses the work schedule defined in a default calendar.

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

See also: “Calendars and Multiple Calendars” on page 104 and “Calendar Data Set” on page 109 for more information on the INTERVAL= option and the specification of working days; “The Default Calendars” on page 103

Featured in: Example 5 on page 134

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.

Featured in: Example 8 on page 147

LOCALE (Experimental)

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.

CAUTION:

LOCALE is an experimental option that is available in SAS 9.1. Do not use this option in production jobs. △

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 may omit some days if you use the OUTSTART statement with the OUTDUR or OUTFIN statement.

Featured in: Example 7 on page 143

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 also: "Missing Values in Input Data Sets" on page 111 for more information on 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.

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 Code 5.1 Illustration of Formatting Characters in PROC CALENDAR Output

```
proc calendar weekdays;
   start date;
run;

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

Featured in: Example 1 on page 114

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 workshift pattern during one working day.

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

See also: "Workdays Data Set" on page 110 and "Calendar Data Set" on page 109

Featured in: Example 3 on page 122

BY Statement

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

Calendar type: Summary and schedule

Main discussion: "BY" on page 58

See also: "CALID Statement" on page 92

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, 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*.

Options

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.

Showing Multiple Calendars in Related Groups

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.

Calendar type: Summary and schedule

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

See also: "Calendar Data Set" on page 109

Featured in: Example 2 on page 118, Example 3 on page 122, and Example 6 on page 137

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

Required Arguments

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 also: "Calendar Data Set" on page 109

Options

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.

Featured in: Example 2 on page 118 and Example 4 on page 127

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.

Featured in: Example 4 on page 127

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.

Featured in: Example 3 on page 122 and Example 8 on page 147

Default: COMBINE

DUR Statement

Specifies the variable that contains the duration of each activity.

Alias: DURATION

Calendar type: Schedule

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

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

Featured in: All schedule calendars (see "Examples: CALENDAR Procedure" on page 114)

DUR *variable*;

Required Arguments

variable

contains the duration of each activity in a schedule calendar.

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

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

See also: For more information on activity durations, see "Activities Data Set" on page 107 and "Calendar Data Set" on page 109

Duration

- 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:

If INTERVAL= …	Then the default length of the duration unit is …
DAY (the default)	24 hours
WORKDAY	8 hours

- You can override the default length of a duration unit by using
  - 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

Calendar type: Schedule

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

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

Featured in: Example 6 on page 137

FIN *variable*;

Required Arguments

variable

contains the finishing date of each activity.

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

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

Restriction: 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

Calendar type: Schedule

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

Restriction: Cannot use with a HOLIFIN statement.

Featured in: Example 1 on page 114 through Example 5 on page 134

HOLIDUR *variable*;

Required Arguments

variable
contains the duration of each holiday.

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

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

Featured in: Example 3 on page 122 and Example 8 on page 147

Holiday Duration

- 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

Calendar type: Schedule

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

HOLIFIN *variable*;

Required Arguments

variable
contains the finishing date of each holiday.

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

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

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

Holiday Duration

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

Calendar type: Summary and schedule

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

Featured in: Example 1 on page 114 through Example 5 on page 134

HOLISTART *variable*;

Required Arguments

variable
contains the starting date of each holiday.

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

Restriction: 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

Calendar type: Summary and schedule

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

Featured in: Example 1 on page 114 through Example 5 on page 134

HOLIVAR *variable*;

Required Arguments

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

Range: character or numeric.

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

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.

Calendar type: Summary

Tip: You can use multiple MEAN statements.

Featured in: Example 7 on page 143

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

Required Arguments

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

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

Options

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

Alias: F=

Default: BEST. format

Featured in: Example 7 on page 143

What Is Displayed and How

- 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.

OUTDUR *number-of-days*;

Required Arguments

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

Length of Week

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.

Featured in: Example 3 on page 122 and Example 8 on page 147

OUTFIN *day-of-week*;

Required Arguments

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

```
outfin friday;
```

Length of Week

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.

Featured in: Example 3 on page 122 and Example 8 on page 147

OUTSTART *day-of-week*;

Required Arguments

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

```
outstart monday;
```

Interaction with OUTDUR and OUTFIN

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

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

Featured in: All examples

START *variable*;

Required Arguments

variable
contains the starting date of each activity.

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

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

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

Restriction: 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.

Calendar type: Summary

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

Featured in: Example 7 on page 143 and Example 8 on page 147

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

Required Arguments

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

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

Options

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

Alias: F=

Default: BEST. format

Featured in: Example 7 on page 143 and Example 8 on page 147

What Is Displayed and How

- 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

VAR *variable(s)*;

Required Arguments

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 that 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, even if you do not name that variable in a VAR statement.

Concepts: CALENDAR Procedure

Type of Calendars

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

Use a …	if you want to …	and can accept this 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.

Use this statement …	to specify a variable whose value indicates the …
START	starting date of an activity
DUR*	duration of an activity
FIN*	ending date of an activity

* Choose one of these. If you do not use a DUR or FIN statement, then PROC CALENDAR assumes that you want to create a summary calendar report.

Examples

See “Simple Schedule Calendar” on page 79, “Advanced Schedule Calendar” on page 80, as well as Example 1 on page 114, Example 2 on page 118, Example 3 on page 122, Example 4 on page 127, Example 5 on page 134, and Example 6 on page 137

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

See "Simple Summary Calendar" on page 81, Example 7 on page 143, and Example 8 on page 147

The Default Calendars

Description

PROC CALENDAR provides two default calendars for simple applications. You can produce calendars without having to specify detailed workshifts and weekly work patterns if your application can use one of two simple work patterns. Consider using a default calendar if

- your application uses a 5-day work week with 8-hour days or a 7-day work week with 24-hour days. See Table 5.2 on page 103.
- you want to print all activities on the same calendar.
- you do not need to identify separate calendars.

Table 5.2 Default Calendar Settings and Examples

If scheduled work days are	Then set INTERVAL=	By default DAYLENGTH=	So work periods are	Shown in 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 5.1 and the 5-day default calendar in Example 1 on page 114

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

- 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.

As an 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 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 on page 137.

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 on page 118, Example 3 on page 122, Example 4 on page 127, and Example 8 on page 147

Input Data Sets

You may 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. See Table 5.3 on page 106.

Table 5.3 Four Possible Input Data Sets for PROC CALENDAR

Data Set	Description	Specify with the …
activities	Each *observation* contains information about a single activity.	DATA= option
holidays	Each *observation* contains information about a holiday	HOLIDATA= option

Data Set	Description	Specify with the …
calendar	Each *observation* defines one weekly work schedule.	CALEDATA= option
workdays	Each *variable* represents one daily schedule of alternating work and nonwork periods.	WORKDATA= option

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.

If a variable contains an activity's …	Then specify it with the …	For this type of calendar…
starting date	START statement	Schedule Summary
duration	DUR statement	Schedule
finishing date	FIN statement	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 holidays on your calendar output
- identify 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.

If a variable contains a holiday's …	Then specify it with this statement …
starting date	HOLISTART
name	HOLIVAR
duration	HOLIDUR
finishing date	HOLIFIN

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
;
```

The variables in this calendar data set consist of

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 workshifts. Valid values for workshifts are

- **`WORKDAY`** (the default workshift)
- **`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. See “The Default Calendars” on page 103 for further details.

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 8 hours if INTERVAL=WORKDAY.

Using Default Workshifts 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:

If INTERVAL=	Then the work-shift begins at …	And the day length is …
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

Example 3 on page 122, Example 4 on page 127, and Example 7 on page 143 feature a calendar data set.

Workdays Data Set

Purpose

You can use a workdays data set, specified with the WORKDATA= option, to define the daily workshifts 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:

If INTERVAL=	Then the work-shift begins at …	And the day length is …
DAY	00:00	24 hours
WORKDAY	9:00	8 hours

See the INTERVAL= option on page 89.

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 on page 122

Missing Values in Input Data Sets

Table 5.4 on page 111 summarizes the treatment of missing values for variables in the data sets used by PROC CALENDAR.

Table 5.4 Treatment of Missing Values in PROC CALENDAR

Data set	Variable	Treatment of missing values
Activities (DATA=)	CALID	default calendar value is used
	START	observation is not used
	DUR	1.0 is used
	FIN	START value + daylength is used

Data set	Variable	Treatment of missing values
	VAR	if a summary calendar or the MISSING option is specified, then the missing value is used; otherwise, no value is used
	SUM, MEAN	0
Calendar (CALEDATA=)	CALID	default calendar value is used
	SUN *through* _SAT_	corresponding shift for default calendar is used
	D_LENGTH	if available, DAYLENGTH= value is used; otherwise, if INTERVAL=DAY, 24:00 is used; otherwise 8:00 is used
	SUM, MEAN	0
Holiday (HOLIDATA=)	CALID	all holidays apply to all calendars
	HOLISTART	observation is not used
	HOLIDUR	if available, HOLIFIN value is used instead of HOLIDUR value; otherwise 1.0 is used
	HOLIFIN	if available, HOLIDUR value is used instead of HOLIFIN value; otherwise, HOLISTART value + day length is used
	HOLIVAR	no value is used
Workdays (WORKDATA=)	any	for the first observation, 00:00 is used; otherwise, 24:00 is used

Results: CALENDAR Procedure

What Affects the Quantity of PROC CALENDAR Output

The quantity of printed calendar output depends on

- 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 this by specifying

- a larger linesize 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 Table 5.1 on page 87 and Figure 5.1 on page 88.

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 on which hexadecimal codes to use for which characters, consult the documentation for your hardware. For an example of assigning hexadecimal values, see FORMCHAR= on page 87.

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

Procedure features:

PROC CALENDAR statement options:

- DATA=
- HOLIDATA=
- WEEKDAYS

DUR statement

HOLISTART statement

HOLIVAR statement

HOLIDUR statement

START statement

Other features:

PROC SORT statement

BY statement

5-day default calendar

This example

- 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

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;
01JUL96 Dist. Mtg.                  All          1
17JUL96 Bank Meeting                1st Natl     1
02JUL96 Mgrs. Meeting               District 6   2
11JUL96 Mgrs. Meeting               District 7   2
03JUL96 Interview                   JW           1
08JUL96 Sales Drive                 District 6   5
15JUL96 Sales Drive                 District 7   5
08JUL96 Trade Show                  Knox         3
```

```
22JUL96 Inventors Show          Melvin       3
11JUL96 Planning Council        Group II     1
18JUL96 Planning Council        Group III    1
25JUL96 Planning Council        Group IV     1
12JUL96 Seminar                 White        1
19JUL96 Seminar                 White        1
18JUL96 NewsLetter Deadline     All          1
05JUL96 VIP Banquet             JW           1
19JUL96 Co. Picnic              All          1
16JUL96 Dentist                 JW           1
24JUL96 Birthday                Mary         1
25JUL96 Close Sale              WYGIX Co.    2
;
```

Create the holidays data set.

```
data hol;
   input date : date7. holiday $ 11-25 holilong @27;
   datalines;
05jul96   Vacation        3
04jul96   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 LINESIZE= appropriately. If the line size is not long enough to print the variable values, then PROC CALENDAR either truncates the values or produces no calendar output.

```
options nodate pageno=1 linesize=132 pagesize=60;
```

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

Output 5.4 Schedule Calendar: 5-Day Week with Holidays

```
                                       Summer Planning Calendar:  Julia Cho                                              1
                                           President, Community Bank

 ------------------------------------------------------------------------------------------------------------------------
 |                                                                                                                      |
 |                                                      July  1996                                                      |
 |                                                                                                                      |
 |----------------------------------------------------------------------------------------------------------------------|
 |         Monday         |        Tuesday         |       Wednesday        |        Thursday        |         Friday         |
 |------------------------+------------------------+------------------------+------------------------+------------------------|
 |           1            |           2            |           3            |           4            |           5            |
 |                        |                        |                        |******Independence*******|********Vacation*********|
 |                        |                        |                        |                        |                        |
 |                        |                        |                        |                        |                        |
 |                        |                        |                        |                        |                        |
 |                        |                        |                        |                        |                        |
 |                        |                        |+=====Interview/JW======+|                        |                        |
 |+====Dist. Mtg./All=====+|+============Mgrs. Meeting/District 6=============+|                        |                        |
 |------------------------+------------------------+------------------------+------------------------+------------------------|
 |           8            |           9            |          10            |          11            |          12            |
 |********Vacation*********|********Vacation*********|                        |                        |                        |
 |                        |                        |                        |                        |                        |
 |                        |                        |                        |                        |                        |
 |                        |                        |                        |+Planning Council/Group +|+=====Seminar/White=====+|
 |                        |                        |+============================Trade Show/Knox=============================+|
 |                        |                        |+=========================Sales Drive/District 6==========================>|
 |                        |                        |+====VIP Banquet/JW=====+|+============Mgrs. Meeting/District 7=============+|
 |------------------------+------------------------+------------------------+------------------------+------------------------|
 |          15            |          16            |          17            |          18            |          19            |
 |                        |                        |                        |                        |                        |
 |                        |                        |                        |                        |                        |
 |                        |                        |                        |                        |                        |
 |                        |                        |                        |                        |                        |
 |                        |+======Dentist/JW=======+|                        |+NewsLetter Deadline/All+|+====Co. Picnic/All=====+|
 |+=================================================Sales Drive/District 7=================================================+|
 |<=============Sales Drive/District 6==============+|+=Bank Meeting/1st Natl=+|+Planning Council/Group +|+=====Seminar/White=====+|
 |------------------------+------------------------+------------------------+------------------------+------------------------|
 |          22            |          23            |          24            |          25            |          26            |
 |                        |                        |                        |                        |                        |
 |                        |                        |                        |                        |                        |
 |                        |                        |                        |                        |                        |
 |                        |                        |                        |                        |                        |
 |                        |                        |                        |                        |                        |
 |                        |                        |+=====Birthday/Mary=====+|+==============Close Sale/WYGIX Co.===============+|
 |+==========================Inventors Show/Melvin===========================+|+Planning Council/Group +|                        |
 |------------------------+------------------------+------------------------+------------------------+------------------------|
 |          29            |          30            |          31            |                        |                        |
 |                        |                        |                        |                        |                        |
 |                        |                        |                        |                        |                        |
 |                        |                        |                        |                        |                        |
 |                        |                        |                        |                        |                        |
 |                        |                        |                        |                        |                        |
 |                        |                        |                        |                        |                        |
 |                        |                        |                        |                        |                        |
 ------------------------------------------------------------------------------------------------------------------------
```

Example 2: Schedule Calendar Containing Multiple Calendars

Procedure features:

CALID statement:

CAL variable

OUTPUT=COMBINE option

DUR statement

24-hour day, 7-day week

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

- 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

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;
01JUL96  Dist. Mtg.               All           CAL1   1
02JUL96  Mgrs. Meeting            District 6    CAL1   2
03JUL96  Interview                JW            CAL1   1
05JUL96  VIP Banquet              JW            CAL1   1
06JUL96  Beach trip               family        CAL2   2
08JUL96  Sales Drive              District 6    CAL1   5
08JUL96  Trade Show               Knox          CAL1   3
09JUL96  Orthodontist             Meagan        CAL2   1
11JUL96  Mgrs. Meeting            District 7    CAL1   2
11JUL96  Planning Council         Group II      CAL1   1
12JUL96  Seminar                  White         CAL1   1
14JUL96  Co. Picnic               All           CAL1   1
14JUL96  Business trip            Fred          CAL2   2
15JUL96  Sales Drive              District 7    CAL1   5
16JUL96  Dentist                  JW            CAL1   1
17JUL96  Bank Meeting             1st Natl      CAL1   1
17JUL96  Real estate agent        Family        CAL2   1
18JUL96  NewsLetter Deadline      All           CAL1   1
18JUL96  Planning Council         Group III     CAL1   1
19JUL96  Seminar                  White         CAL1   1
22JUL96  Inventors Show           Melvin        CAL1   3
24JUL96  Birthday                 Mary          CAL1   1
```

```
25JUL96  Planning Council         Group IV     CAL1   1
25JUL96  Close Sale               WYGIX Co.    CAL1   2
27JUL96  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;
29JUL96   vacation                CAL2
04JUL96   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 LINESIZE= appropriately. If the linesize is not long enough to print the variable values, then PROC CALENDAR either truncates the values or produces no calendar output.

```
options nodate pageno=1 pagesize=60 linesize=132;
```

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

Output 5.5 Schedule Calendar Containing Multiple Calendars

```
                                   Summer Planning Calendar:  Julia Cho                                  1
                                       President, Community Bank
                                        Work and Home Schedule

          -------------------------------------------------------------------------------------------------------------------
          |                                                                                                                 |
          |                                                   July  1996                                                    |
          |                                                                                                                 |
          |-----------------------------------------------------------------------------------------------------------------|
          |     Sunday     |     Monday     |    Tuesday     |   Wednesday    |    Thursday    |     Friday     |    Saturday    |
----------+----------------+----------------+----------------+----------------+----------------+----------------+----------------|
|         |                |       1        |       2        |       3        |       4        |       5        |       6        |
|.........|................|................|................|................|................|................|................|
| CAL2    |                |                |                |                |                |                |+Beach trip/fam>|
|.........|................|................|................|................|................|................|................|
| CAL1    |                |                |                |+=Interview/JW=+|**Independence**|                |                |
|         |                |+Dist. Mtg./All+|+===Mgrs. Meeting/District 6====+|                |+VIP Banquet/JW+|                |
|         |                |                |                |                |                |                |                |
|---------+----------------+----------------+----------------+----------------+----------------+----------------+----------------|
|         |       7        |       8        |       9        |       10       |       11       |       12       |       13       |
|.........|................|................|................|................|................|................|................|
| CAL2    |<Beach trip/fam+|                |+Orthodontist/M+|                |                |                |                |
|.........|................|................|................|................|................|................|................|
| CAL1    |                |                |                |                |+Planning Counc+|+Seminar/White=+|                |
|         |                |+================Trade Show/Knox=================+|+===Mgrs. Meeting/District 7====+|                |
|         |                |+=============================Sales Drive/District 6=============================+|                |
|---------+----------------+----------------+----------------+----------------+----------------+----------------+----------------|
|         |       14       |       15       |       16       |       17       |       18       |       19       |       20       |
|.........|................|................|................|................|................|................|................|
| CAL2    |+======Business trip/Fred=======+|                |+Real estate ag+|                |                |                |
|.........|................|................|................|................|................|................|................|
| CAL1    |                |                |                |                |+Planning Counc+|                |                |
|         |                |                |+==Dentist/JW==+|+Bank Meeting/1+|+NewsLetter Dea+|+Seminar/White=+|                |
|         |+Co. Picnic/All+|+=============================Sales Drive/District 7=============================+|                |
|---------+----------------+----------------+----------------+----------------+----------------+----------------+----------------|
|         |       21       |       22       |       23       |       24       |       25       |       26       |       27       |
|.........|................|................|................|................|................|................|................|
| CAL2    |                |                |                |                |                |                |+Ballgame/Famil+|
|.........|................|................|................|................|................|................|................|
| CAL1    |                |                |                |+Birthday/Mary=+|+=====Close Sale/WYGIX Co.======+|                |
|         |                |+=============Inventors Show/Melvin==============+|+Planning Counc+|                |                |
|         |                |                |                |                |                |                |                |
|---------+----------------+----------------+----------------+----------------+----------------+----------------+----------------|
|         |       28       |       29       |       30       |       31       |                |                |                |
|.........|................|................|................|................|................|................|................|
| CAL2    |                |****vacation****|                |                |                |                |                |
|         |                |                |                |                |                |                |                |
|         |                |                |                |                |                |                |                |
|         |                |                |                |                |                |                |                |
|         |                |                |                |                |                |                |                |
----------------------------------------------------------------------------------------------------------------------------------
```

Example 3: Multiple Schedule Calendars with Atypical Workshifts (Separated Output)

Procedure features:

PROC CALENDAR statement options:
- CALEDATA=
- DATETIME
- WORKDATA=

CALID statement:
- _CAL_ variable
- OUTPUT=SEPARATE option

DUR statement

OUTSTART statement

OUTFIN statement

This example

- □ 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 on page 127 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.

To print …	Sort the activities data set by …	And set OUTPUT= to	See Example
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

Specify a library so that you can permanently store the activities data set.

```
libname well 'SAS-data-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  01JUL96:12:00:00  CAL1   1000
Lay Power Line      3.00  04JUL96:12:00:00  CAL1   2000
Assemble Tank       4.00  05JUL96:08:00:00  CAL1   1000
Build Pump House    3.00  08JUL96:12:00:00  CAL1   2000
Pour Foundation     4.00  11JUL96:08:00:00  CAL1   1500
Install Pump        4.00  15JUL96:14:00:00  CAL1    500
Install Pipe        2.00  19JUL96:08:00:00  CAL1   1000
Erect Tower         6.00  20JUL96:08:00:00  CAL1   2500
Deliver Material    2.00  01JUL96:12:00:00  CAL2    500
Excavate            4.75  03JUL96:08:00:00  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;
09JUL96   Vacation            CAL2
04JUL96   Independence        CAL1
;
```

Create the calendar data set. Each observation defines the workshifts for an entire week. The _CAL_ variable identifies to which calendar the workshifts apply. CAL1 uses the default 8-hour workshifts for Monday through Friday. CAL2 uses a half day on Saturday and the default 8-hour workshift 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 workshifts that are named in the calendar data set. Each variable (not observation) contains one daily schedule of alternating work and nonwork periods. The HALFDAY workshift 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 LINESIZE= appropriately. If the linesize is not long enough to print the variable values, then PROC CALENDAR either truncates the values or produces no calendar output.

```
options nodate pageno=1 linesize=132 pagesize=60;
```

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

Output 5.6 Separate Output for Multiple Schedule Calendars

```
                                    Well Drilling Work Schedule: Separate Calendars                                   1

........................................................... _cal_=CAL1 ...........................................................

  ----------------------------------------------------------------------------------------------------------------------------
  |                                                                                                                          |
  |                                                      July  1996                                                          |
  |                                                                                                                          |
  |--------------------------------------------------------------------------------------------------------------------------|
  |      Monday        |      Tuesday       |     Wednesday      |      Thursday      |       Friday       |      Saturday      |
  |--------------------+--------------------+--------------------+--------------------+--------------------+--------------------|
  |          1         |          2         |          3         |          4         |          5         |          6         |
  |                    |                    |                    |****Independence****|                    |                    |
  |                    |                    |                    |                    |                    |                    |
  |                    |                    |                    |                    |                    |                    |
  |                    |                    |                    |                    |+Assemble Tank/$1,0>|                    |
  |                    |                    |                    |                    |+Lay Power Line/$2,>|                    |
  |+===================Drill Well/$1,000.00===================>|                    |<Drill Well/$1,000.+|                    |
  |--------------------+--------------------+--------------------+--------------------+--------------------+--------------------|
  |          8         |          9         |         10         |         11         |         12         |         13         |
  |                    |                    |                    |                    |                    |                    |
  |                    |                    |                    |                    |                    |                    |
  |                    |                    |                    |                    |                    |                    |
  |+==========================Build Pump House/$2,000.00===========================+|                    |                    |
  |<===========================Assemble Tank/$1,000.00============================+|                    |                    |
  |<=======Lay Power Line/$2,000.00========+|                    |+=======Pour Foundation/$1,500.00=======>|                    |
  |--------------------+--------------------+--------------------+--------------------+--------------------+--------------------|
  |         15         |         16         |         17         |         18         |         19         |         20         |
  |                    |                    |                    |                    |                    |                    |
  |                    |                    |                    |                    |                    |                    |
  |                    |                    |                    |                    |                    |                    |
  |                    |                    |                    |                    |                    |                    |
  |+=======================================Install Pump/$500.00=======================================+|                    |
  |<=================Pour Foundation/$1,500.00==================+|                    |+Install Pipe/$1,00>|                    |
  |--------------------+--------------------+--------------------+--------------------+--------------------+--------------------|
  |         22         |         23         |         24         |         25         |         26         |         27         |
  |                    |                    |                    |                    |                    |                    |
  |                    |                    |                    |                    |                    |                    |
  |                    |                    |                    |                    |                    |                    |
  |                    |                    |                    |                    |                    |                    |
  |+======================================Erect Tower/$2,500.00=======================================>|                    |
  |<========Install Pipe/$1,000.00=========+|                    |                    |                    |                    |
  |--------------------+--------------------+--------------------+--------------------+--------------------+--------------------|
  |         29         |         30         |         31         |                    |                    |                    |
  |                    |                    |                    |                    |                    |                    |
  |                    |                    |                    |                    |                    |                    |
  |                    |                    |                    |                    |                    |                    |
  |                    |                    |                    |                    |                    |                    |
  |<Erect Tower/$2,500+|                    |                    |                    |                    |                    |
  ----------------------------------------------------------------------------------------------------------------------------
```

```
                                   Well Drilling Work Schedule: Separate Calendars                                    2

.......................................................... _cal_=CAL2 ..........................................................

  -----------------------------------------------------------------------------------------------------------------------------
  |                                                                                                                           |
  |                                                        July  1996                                                         |
  |                                                                                                                           |
  |---------------------------------------------------------------------------------------------------------------------------|
  |      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/$3,500.00>|                    |<Excavate/$3,500.00+|                    |                    |                    |
  |--------------------+--------------------+--------------------+--------------------+--------------------+--------------------|
  |        15          |        16          |        17          |        18          |        19          |        20          |
  |                    |                    |                    |                    |                    |                    |
  |                    |                    |                    |                    |                    |                    |
  |                    |                    |                    |                    |                    |                    |
  |                    |                    |                    |                    |                    |                    |
  |                    |                    |                    |                    |                    |                    |
  |--------------------+--------------------+--------------------+--------------------+--------------------+--------------------|
  |        22          |        23          |        24          |        25          |        26          |        27          |
  |                    |                    |                    |                    |                    |                    |
  |                    |                    |                    |                    |                    |                    |
  |                    |                    |                    |                    |                    |                    |
  |                    |                    |                    |                    |                    |                    |
  |                    |                    |                    |                    |                    |                    |
  |--------------------+--------------------+--------------------+--------------------+--------------------+--------------------|
  |        29          |        30          |        31          |                    |                    |                    |
  |                    |                    |                    |                    |                    |                    |
  |                    |                    |                    |                    |                    |                    |
  |                    |                    |                    |                    |                    |                    |
  |                    |                    |                    |                    |                    |                    |
  |                    |                    |                    |                    |                    |                    |
  -----------------------------------------------------------------------------------------------------------------------------
```

Example 4: Multiple Schedule Calendars with Atypical Workshifts (Combined and Mixed Output)

Procedure features:

PROC CALENDAR statement options:

- CALEDATA=
- DATETIME
- WORKDATA=

CALID statement:

- _CAL_ variable

OUTPUT=COMBINE option
OUTPUT=MIXED option
DUR statement
OUTSTART statement
OUTFIN statement

Data sets:
WELL.ACT on page 123, WELL.HOL on page 123, WELL.CAL on page 123, WEL.WOR on page 123.

This example

- □ 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).

Two Programs and Two Pieces of 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.

Producing Different Output for Multiple Calendars

This example and Example 3 on page 122 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.

To print …	Sort the activities data set by …	And set OUTPUT= to	See Example
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 Calendars

Specify the SAS data library where the activities data set is stored.

```
libname well 'SAS-data-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 PAGESIZE= and LINESIZE= appropriately. When you combine calendars, check the value of PAGESIZE= to ensure that there is enough room to print the activities from multiple calendars. If LINESIZE= is too small for the variable values to print, then PROC CALENDAR either truncates the values or produces no calendar output.

```
options nodate pageno=1 linesize=132 pagesize=60;
```

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

Output 5.7 Multiple Schedule Calendars with Atypical Workshifts (Combined Output)

```
                                 Well Drilling Work Schedule: Combined Calendars                                              1

          ---------------------------------------------------------------------------------------------------------------------
          |                                                                                                                   |
          |                                                    July  1996                                                     |
          |                                                                                                                   |
          |-------------------------------------------------------------------------------------------------------------------|
          |     Sunday     |     Monday     |    Tuesday     |   Wednesday    |    Thursday    |     Friday     |    Saturday    |
----------+----------------+----------------+----------------+----------------+----------------+----------------+----------------|
|         |                |       1        |       2        |       3        |       4        |       5        |       6        |
|.........|................|................|................|................|................|................|................|
| CAL1    |                |                |                |                |**Independence**|+Assemble Tank/>|                |
|         |                |                |                |                |                |+Lay Power Line>|                |
|         |                |+==============Drill Well/$1,000.00==============>|                |<Drill Well/$1,+|                |
|.........|................|................|................|................|................|................|................|
| 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/$2,000.00====+|                |+===Pour Foundation/$1,500.00===>|                |
|.........|................|................|................|................|................|................|................|
| CAL2    |                |<Excavate/$3,50>|****Vacation****|<Excavate/$3,50+|                |                |                |
|---------+----------------+----------------+----------------+----------------+----------------+----------------+----------------|
|         |       14       |       15       |       16       |       17       |       18       |       19       |       20       |
|.........|................|................|................|................|................|................|................|
| CAL1    |                |+=============================Install Pump/$500.00==============================+|                |
|         |                |<===========Pour Foundation/$1,500.00============+|                |+Install Pipe/$>|                |
|         |                |                |                |                |                |                |                |
|         |                |                |                |                |                |                |                |
|         |                |                |                |                |                |                |                |
|---------+----------------+----------------+----------------+----------------+----------------+----------------+----------------|
|         |       21       |       22       |       23       |       24       |       25       |       26       |       27       |
|.........|................|................|................|................|................|................|................|
| CAL1    |                |+============================Erect Tower/$2,500.00==============================>|                |
|         |                |<====Install Pipe/$1,000.00=====+|                |                |                |                |
|         |                |                |                |                |                |                |                |
|         |                |                |                |                |                |                |                |
|         |                |                |                |                |                |                |                |
|---------+----------------+----------------+----------------+----------------+----------------+----------------+----------------|
|         |       28       |       29       |       30       |       31       |                |                |                |
|.........|................|................|................|................|................|................|................|
| CAL1    |                |<Erect Tower/$2+|                |                |                |                |                |
|         |                |                |                |                |                |                |                |
|         |                |                |                |                |                |                |                |
|         |                |                |                |                |                |                |                |
|         |                |                |                |                |                |                |                |
----------------------------------------------------------------------------------------------------------------------------------
```

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

Output 5.8 Multiple Schedule Calendar with Atypical Workshifts (Mixed Output)

```
                                         Well Drilling Work Schedule: Mixed Calendars                                           1

-------------------------------------------------------------------------------------------------------------------------------
|                                                                                                                             |
|                                                         July  1996                                                          |
|                                                                                                                             |
|-----------------------------------------------------------------------------------------------------------------------------|
|       Monday       |      Tuesday       |     Wednesday      |      Thursday      |       Friday       |      Saturday      |
|--------------------+--------------------+--------------------+--------------------+--------------------+--------------------|
|         1          |         2          |         3          |         4          |         5          |         6          |
|                    |                    |                    |                    |                    |                    |
|                    |                    |                    |                    |                    |                    |
|                    |                    |                    |                    |+Assemble Tank/$1,0>|                    |
|                    |                    |+==============================Excavate/$3,500.00=================================>|
|+==================Deliver Material/$500.00==================+|****Independence****|+Lay Power Line/$2,>|                    |
|+====================Drill Well/$1,000.00====================>|****Independence****|<Drill Well/$1,000.+|                    |
|--------------------+--------------------+--------------------+--------------------+--------------------+--------------------|
|         8          |         9          |        10          |        11          |        12          |        13          |
|                    |                    |                    |                    |                    |                    |
|                    |                    |                    |                    |                    |                    |
|+===========================Build Pump House/$2,000.00============================+|                    |                    |
|<=============================Assemble Tank/$1,000.00=============================+|                    |                    |
|<=======Lay Power Line/$2,000.00========+|                    |                    |                    |                    |
|<Excavate/$3,500.00>|******Vacation******|<Excavate/$3,500.00+|+=======Pour Foundation/$1,500.00=======>|                    |
|--------------------+--------------------+--------------------+--------------------+--------------------+--------------------|
|        15          |        16          |        17          |        18          |        19          |        20          |
|                    |                    |                    |                    |                    |                    |
|                    |                    |                    |                    |                    |                    |
|                    |                    |                    |                    |                    |                    |
|                    |                    |                    |                    |                    |                    |
|+=========================================Install Pump/$500.00=========================================+|                    |
|<=================Pour Foundation/$1,500.00==================+|                    |+Install Pipe/$1,00>|                    |
|--------------------+--------------------+--------------------+--------------------+--------------------+--------------------|
|        22          |        23          |        24          |        25          |        26          |        27          |
|                    |                    |                    |                    |                    |                    |
|                    |                    |                    |                    |                    |                    |
|                    |                    |                    |                    |                    |                    |
|                    |                    |                    |                    |                    |                    |
|+========================================Erect Tower/$2,500.00=========================================>|                    |
|<========Install Pipe/$1,000.00=========+|                    |                    |                    |                    |
|--------------------+--------------------+--------------------+--------------------+--------------------+--------------------|
|        29          |        30          |        31          |                    |                    |                    |
|                    |                    |                    |                    |                    |                    |
|                    |                    |                    |                    |                    |                    |
|                    |                    |                    |                    |                    |                    |
|                    |                    |                    |                    |                    |                    |
|                    |                    |                    |                    |                    |                    |
|<Erect Tower/$2,500+|                    |                    |                    |                    |                    |
-------------------------------------------------------------------------------------------------------------------------------
```

Example 5: Schedule Calendar, Blank or with Holidays

Procedure features:

PROC CALENDAR statement options:

- FILL
- HOLIDATA=
- INTERVAL=WORKDAY

DUR statement

HOLIDUR statement

HOLISTART statement

HOLIVAR statement

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

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;
01JAN97   Start                0
31DEC97   Finish               0
;
```

Create the holidays data set.

```
data holidays;
   input sta : date7. act $ 11-30 dur;
   datalines;
01JAN97   New Year's           1
28MAR97   Good Friday          1
30MAY97   Memorial Day         1
04JUL97   Independence Day     1
01SEP97   Labor Day            1
27NOV97   Thanksgiving         2
25DEC97   Christmas Break      5
;
```

Set PAGESIZE= and LINESIZE= appropriately. To create larger boxes for each day in the calendar output, increase the value of PAGESIZE=.

```
options nodate pageno=1 linesize=132 pagesize=30;
```

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

Output 5.9 Schedule Calendars with Holidays Only (Partial Output).

Without INTERVAL=WORKDAY, the 5-day Christmas break would be scheduled through the weekend.

```
                                     Calendar of Holidays Only                                        1

 ----------------------------------------------------------------------------------------------------------------------------
 |                                                                                                                          |
 |                                                      January  1997                                                       |
 |                                                                                                                          |
 |--------------------------------------------------------------------------------------------------------------------------|
 |     Sunday      |     Monday      |     Tuesday     |    Wednesday    |    Thursday     |     Friday      |    Saturday     |
 |-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------|
 |                 |                 |                 |        1        |        2        |        3        |        4        |
 |                 |                 |                 |***New Year's****|                 |                 |                 |
 |-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------|
 |        5        |        6        |        7        |        8        |        9        |       10        |       11        |
 |                 |                 |                 |                 |                 |                 |                 |
 |-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------|
 |       12        |       13        |       14        |       15        |       16        |       17        |       18        |
 |                 |                 |                 |                 |                 |                 |                 |
 |-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------|
 |       19        |       20        |       21        |       22        |       23        |       24        |       25        |
 |                 |                 |                 |                 |                 |                 |                 |
 |-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------|
 |       26        |       27        |       28        |       29        |       30        |       31        |                 |
 |                 |                 |                 |                 |                 |                 |                 |
 ----------------------------------------------------------------------------------------------------------------------------
```

```
                                     Calendar of Holidays Only                                        2

 ----------------------------------------------------------------------------------------------------------------------------
 |                                                                                                                          |
 |                                                      February  1997                                                      |
 |                                                                                                                          |
 |--------------------------------------------------------------------------------------------------------------------------|
 |     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        |       26        |       27        |       28        |                 |
 |                 |                 |                 |                 |                 |                 |                 |
 ----------------------------------------------------------------------------------------------------------------------------
```

```
                              Calendar of Holidays Only                                              12

 -------------------------------------------------------------------------------------------------------
 |                                                                                                     |
 |                                            December  1997                                           |
 |                                                                                                     |
 |-----------------------------------------------------------------------------------------------------|
 |     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        |      26        |      27        |
 |                |                |                |                |*Christmas Break*|*Christmas Break*|               |
 |----------------+----------------+----------------+----------------+----------------+----------------+----------------|
 |      28        |      29        |      30        |      31        |                |                |                |
 |                |*Christmas Break*|*Christmas Break*|*Christmas Break*|               |                |                |
 -------------------------------------------------------------------------------------------------------
```

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

Procedure features:

PROC CALENDAR statement

CALID statement

FIN statement

VAR statement

Other features:

PROC CPM step

PROC SORT step

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

1. create an activities data set that contains activities with durations. (You can indicate nonwork days, weekly work schedules, and workshifts with holidays, calendar, and workshift 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, workshifts, 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.

Highlights of This Example

This example

- 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

- 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).

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 User's Guide: Project Management*.

Program

Set appropriate options. If the linesize is not long enough to print the variable values, then PROC CALENDAR either truncates the values or produces no calendar output. A longer linesize also makes it easier to view a listing of a PROC CPM output data set.

```
options nodate pageno=1 linesize=132 pagesize=60;
```

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 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                   18jul96 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;
04jul96 04jul96 Independence Day Prof.
02sep96 02sep96 Labor Day        Prof.
04jul96 04jul96 Independence Day Student
02sep96 02sep96 Labor Day        Student
15jul96 16jul96 PROF Vacation    Prof.
15aug96 16aug96 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='01jul96'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. See Output 5.10.

```
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. See Output 5.11.

```
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

Output 5.10 The Data Set GCPM1

PROC PRINT displays the observations in GCPM1, showing the scheduling calculations created by PROC CPM.

Data Set GCPM1, Created with PROC CPM 1

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	01JUL96	16JUL96	01JUL96	16JUL96	0	0
2	Analyze Exp 1	Send Report 1	5	Prof.	17JUL96	23JUL96	17JUL96	23JUL96	0	0
3	Send Report 1	Run Exp 2	0	Prof.	24JUL96	24JUL96	24JUL96	24JUL96	0	0
4	Run Exp 2	Analyze Exp 2	11	Student	24JUL96	07AUG96	24JUL96	07AUG96	0	0
5	Analyze Exp 2	Send Report 2	4	Prof.	08AUG96	13AUG96	08AUG96	13AUG96	0	0
6	Send Report 2	Write Final Report	0	Prof.	14AUG96	14AUG96	14AUG96	14AUG96	0	0
7	Write Final Report	Send Final Report	4	Prof.	14AUG96	19AUG96	14AUG96	19AUG96	0	0
8	Send Final Report		0	Student	20AUG96	20AUG96	20AUG96	20AUG96	0	0
9	Site Visit		1	Prof.	18JUL96	18JUL96	18JUL96	18JUL96	0	0

Output 5.11 Schedule Calendar Based on Output from PROC CPM

PROC CALENDAR created this schedule calendar by using the S_START and S_FINISH dates that were calculated by PROC CPM. The activities on July 24th and August 14th, 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                                            2
                                           Professor and Student Schedule

          -----------------------------------------------------------------------------------------------------------------------
          |                                                                                                                      |
          |                                               July  1996                                                             |
          |                                                                                                                      |
          |----------------------------------------------------------------------------------------------------------------------|
          |     Sunday     |     Monday     |    Tuesday     |   Wednesday    |    Thursday    |     Friday     |    Saturday    |
----------+----------------+----------------+----------------+----------------+----------------+----------------+----------------|
|         |                |       1        |       2        |       3        |       4        |       5        |       6        |
|.........|................|................|................|................|................|................|................|
| PROF.   |                |                |                |                |Independence Day|                |                |
|.........|................|................|................|................|................|................|................|
| STUDENT |                |+===================Run Exp 1====================>|Independence Day|<==Run Exp 1===>|                |
|         |                |                |                |                |                |                |                |
|         |                |                |                |                |                |                |                |
|---------+----------------+----------------+----------------+----------------+----------------+----------------+----------------|
|         |       7        |       8        |       9        |       10       |       11       |       12       |       13       |
|.........|................|................|................|................|................|................|................|
| STUDENT |                |<====================================Run Exp 1=====================================>|                |
|         |                |                |                |                |                |                |                |
|         |                |                |                |                |                |                |                |
|         |                |                |                |                |                |                |                |
|         |                |                |                |                |                |                |                |
|---------+----------------+----------------+----------------+----------------+----------------+----------------+----------------|
|         |       14       |       15       |       16       |       17       |       18       |       19       |       20       |
|.........|................|................|................|................|................|................|................|
| PROF.   |                |*PROF Vacation**|*PROF Vacation**|                |+==Site Visit==+|                |                |
|         |                |                |                |+=================Analyze Exp 1==================>|                |
|.........|................|................|................|................|................|................|................|
| STUDENT |                |<===========Run Exp 1===========+|                |                |                |                |
|         |                |                |                |                |                |                |                |
|---------+----------------+----------------+----------------+----------------+----------------+----------------+----------------|
|         |       21       |       22       |       23       |       24       |       25       |       26       |       27       |
|.........|................|................|................|................|................|................|................|
| PROF.   |                |<=========Analyze Exp 1=========+|+Send Report 1=+|                |                |                |
|.........|................|................|................|................|................|................|................|
| STUDENT |                |                |                |+===================Run Exp 2====================>|                |
|         |                |                |                |                |                |                |                |
|         |                |                |                |                |                |                |                |
|---------+----------------+----------------+----------------+----------------+----------------+----------------+----------------|
|         |       28       |       29       |       30       |       31       |                |                |                |
|.........|................|................|................|................|................|................|................|
| STUDENT |                |<===================Run Exp 2====================>|                |                |                |
|         |                |                |                |                |                |                |                |
|         |                |                |                |                |                |                |                |
|         |                |                |                |                |                |                |                |
|         |                |                |                |                |                |                |                |
----------------------------------------------------------------------------------------------------------------------------------
```

```
                                     Schedule for Experiment X-15                                                              3
                                     Professor and Student Schedule

          -----------------------------------------------------------------------------------------------------------------------
          |                                                                                                                     |
          |                                                   August  1996                                                      |
          |                                                                                                                     |
          |---------------------------------------------------------------------------------------------------------------------|
          |     Sunday     |     Monday     |    Tuesday     |   Wednesday    |    Thursday    |     Friday     |    Saturday    |
----------+----------------+----------------+----------------+----------------+----------------+----------------+----------------|
|         |                |                |                |                |       1        |       2        |       3        |
|.........|................|................|................|................|................|................|................|
| STUDENT |                |                |                |                |<===========Run Exp 2===========>|                |
|         |                |                |                |                |                |                |                |
|         |                |                |                |                |                |                |                |
|         |                |                |                |                |                |                |                |
|         |                |                |                |                |                |                |                |
|---------+----------------+----------------+----------------+----------------+----------------+----------------+----------------|
|         |       4        |       5        |       6        |       7        |       8        |       9        |       10       |
|.........|................|................|................|................|................|................|................|
| PROF.   |                |                |                |                |+=========Analyze Exp 2=========>|                |
|.........|................|................|................|................|................|................|................|
| STUDENT |                |<===================Run Exp 2====================+|                |                |                |
|         |                |                |                |                |                |                |                |
|         |                |                |                |                |                |                |                |
|---------+----------------+----------------+----------------+----------------+----------------+----------------+----------------|
|         |       11       |       12       |       13       |       14       |       15       |       16       |       17       |
|.........|................|................|................|................|................|................|................|
| PROF.   |                |                |                |+===============Write Final Report===============>|                |
|         |                |<=========Analyze Exp 2=========+|+Send Report 2=+|                |                |                |
|.........|................|................|................|................|................|................|................|
| STUDENT |                |                |                |                |STUDENT Vacation|STUDENT Vacation|                |
|         |                |                |                |                |                |                |                |
|---------+----------------+----------------+----------------+----------------+----------------+----------------+----------------|
|         |       18       |       19       |       20       |       21       |       22       |       23       |       24       |
|.........|................|................|................|................|................|................|................|
| PROF.   |                |<Write Final Re+|                |                |                |                |                |
|.........|................|................|................|................|................|................|................|
| STUDENT |                |                |+Send Final Rep+|                |                |                |                |
|         |                |                |                |                |                |                |                |
|         |                |                |                |                |                |                |                |
|---------+----------------+----------------+----------------+----------------+----------------+----------------+----------------|
|         |       25       |       26       |       27       |       28       |       29       |       30       |       31       |
|         |                |                |                |                |                |                |                |
|         |                |                |                |                |                |                |                |
|         |                |                |                |                |                |                |                |
|         |                |                |                |                |                |                |                |
|         |                |                |                |                |                |                |                |
|         |                |                |                |                |                |                |                |
----------------------------------------------------------------------------------------------------------------------------------
```

Example 7: Summary Calendar with MEAN Values By Observation

Procedure features:

- CALID statement:
  - _CAL_ variable
  - OUTPUT=SEPARATE option
- FORMAT statement
- LABEL statement

MEAN statement
SUM statement

Other features:

PROC FORMAT:
PICTURE statement

This example

- □ 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.

MEAN Values by Number of Days

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

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;
02Dec96       123 234 238
03Dec96       188 188 198
04Dec96       123 183 176
05Dec96       200 267 243
06Dec96       176 165 177
09Dec96       178 198 187
10Dec96       165 176 187
11Dec96       187 176 231
12Dec96       176 187 222
13Dec96       187 187 123
16Dec96       176 165 177
17Dec96       156   . 167
18Dec96       198 143 167
19Dec96       178 198 187
20Dec96       165 176 187
23Dec96       187 187 123
;
```

Create the holidays data set.

```
data closed;
   input date date. holiday $ 11-25;
```

```
   datalines;
26DEC96   Repairs
27DEC96   Repairs
30DEC96   Repairs
31DEC96   Repairs
24DEC96   Christmas Eve
25DEC96   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 PAGESIZE= and LINESIZE= appropriately. The legend box prints on the next page if PAGESIZE= is not set large enough. LINESIZE= controls the width of the cells in the calendar.

```
options nodate pageno=1 linesize=132 pagesize=60;
```

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;
```

Output

Output 5.12 Summary Calendar with MEAN Values by Observation

```
                             Meals Served in Company Cafeteria                                   1
                               Mean Number by Business Day

   ---------------------------------------------------------------------------------------------
   |                                                                                           |
   |                                      December  1996                                       |
   |                                                                                           |
   |-------------------------------------------------------------------------------------------|
   |   Sunday   |   Monday   |  Tuesday   | Wednesday  |  Thursday  |   Friday   |  Saturday  |
   |------------+------------+------------+------------+------------+------------+------------|
   |     1      |     2      |     3      |     4      |     5      |     6      |     7      |
   |            |            |            |            |            |            |            |
   |            | 123 Brkfst | 188 Brkfst | 123 Brkfst | 200 Brkfst | 176 Brkfst |            |
   |            | 234 Lunch  | 188 Lunch  | 183 Lunch  | 267 Lunch  | 165 Lunch  |            |
   |            | 238 Dinner | 198 Dinner | 176 Dinner | 243 Dinner | 177 Dinner |            |
   |------------+------------+------------+------------+------------+------------+------------|
   |     8      |     9      |     10     |     11     |     12     |     13     |     14     |
   |            |            |            |            |            |            |            |
   |            | 178 Brkfst | 165 Brkfst | 187 Brkfst | 176 Brkfst | 187 Brkfst |            |
   |            | 198 Lunch  | 176 Lunch  | 176 Lunch  | 187 Lunch  | 187 Lunch  |            |
   |            | 187 Dinner | 187 Dinner | 231 Dinner | 222 Dinner | 123 Dinner |            |
   |------------+------------+------------+------------+------------+------------+------------|
   |     15     |     16     |     17     |     18     |     19     |     20     |     21     |
   |            |            |            |            |            |            |            |
   |            | 176 Brkfst | 156 Brkfst | 198 Brkfst | 178 Brkfst | 165 Brkfst |            |
   |            | 165 Lunch  |          . | 143 Lunch  | 198 Lunch  | 176 Lunch  |            |
   |            | 177 Dinner | 167 Dinner | 167 Dinner | 187 Dinner | 187 Dinner |            |
   |------------+------------+------------+------------+------------+------------+------------|
   |     22     |     23     |     24     |     25     |     26     |     27     |     28     |
   |            |            |Christmas Ev|*Christmas**|**Repairs***|**Repairs***|            |
   |            | 187 Brkfst |            |            |            |            |            |
   |            | 187 Lunch  |            |            |            |            |            |
   |            | 123 Dinner |            |            |            |            |            |
   |------------+------------+------------+------------+------------+------------+------------|
   |     29     |     30     |     31     |            |            |            |            |
   |            |**Repairs***|**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 Workshifts (Separated Output)

Procedure features:

PROC CALENDAR statement options:

DATETIME

LEGEND

CALID statement:

CAL variable
OUTPUT=SEPARATE option

OUTSTART statement

OUTFIN statement

SUM statement

Data sets:

WELL.ACT on page 123 and WELL.HOL on page 123.

This example

- 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 two things:

- how the activities data set is sorted
- how the OUTPUT= option is set.

To print …	Sort the activities data set by …	And set OUTPUT= to	See Example
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

Specify the SAS data library where the activities data set is stored.

```
libname well 'SAS-data-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 PAGESIZE= and LINESIZE= appropriately. The legend box prints on the next page if PAGESIZE= is not set large enough. LINESIZE= controls the width of the boxes.

```
options nodate pageno=1 linesize=132 pagesize=60;
```

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

Output 5.13 Separated Output for Multiple Summary Calendars

```
                                     Well Drilling Cost Summary                                                  1
                                         Separate Calendars

............................................................ _cal_=CAL1 ............................................................

        ---------------------------------------------------------------------------------------------------------------------
        |                                                                                                                   |
        |                                                  July  1996                                                       |
        |                                                                                                                   |
        |-------------------------------------------------------------------------------------------------------------------|
        |      Monday       |      Tuesday      |     Wednesday     |     Thursday      |      Friday       |     Saturday      |
        |-------------------+-------------------+-------------------+-------------------+-------------------+-------------------|
        |         1         |         2         |         3         |         4         |         5         |         6         |
        |                   |                   |                   |***Independence***|                   |                   |
        | Drill Well        |                   |                   | Lay Power Line    | Assemble Tank     |                   |
        |              3.5  |                   |                   |                3  |                4  |                   |
        |        $1,000.00  |                   |                   |        $2,000.00  |        $1,000.00  |                   |
        |-------------------+-------------------+-------------------+-------------------+-------------------+-------------------|
        |         8         |         9         |        10         |        11         |        12         |        13         |
        |                   |                   |                   |                   |                   |                   |
        | Build Pump House  |                   |                   | Pour Foundation   |                   |                   |
        |                3  |                   |                   |                4  |                   |                   |
        |        $2,000.00  |                   |                   |        $1,500.00  |                   |                   |
        |-------------------+-------------------+-------------------+-------------------+-------------------+-------------------|
        |        15         |        16         |        17         |        18         |        19         |        20         |
        |                   |                   |                   |                   |                   |                   |
        | Install Pump      |                   |                   |                   | Install Pipe      | Erect Tower       |
        |                4  |                   |                   |                   |                2  |                6  |
        |          $500.00  |                   |                   |                   |        $1,000.00  |        $2,500.00  |
        |-------------------+-------------------+-------------------+-------------------+-------------------+-------------------|
        |        22         |        23         |        24         |        25         |        26         |        27         |
        |                   |                   |                   |                   |                   |                   |
        |                   |                   |                   |                   |                   |                   |
        |                   |                   |                   |                   |                   |                   |
        |-------------------+-------------------+-------------------+-------------------+-------------------+-------------------|
        |        29         |        30         |        31         |                   |                   |                   |
        |                   |                   |                   |                   |                   |                   |
        |                   |                   |                   |                   |                   |                   |
        |                   |                   |                   |                   |                   |                   |
        |                   |                   |                   |                   |                   |                   |
        ---------------------------------------------------------------------------------------------------------------------

                                        -------------------------
                                        |  Legend  |    Sum     |
                                        |          |            |
                                        | task     |            |
                                        | dur      |            |
                                        | cost     | $11,500.00 |
                                        -------------------------
```

```
                                 Well Drilling Cost Summary                                         2
                                    Separate Calendars

............................................................ _cal_=CAL2 ............................................................

        ------------------------------------------------------------------------------------------------------------------
        |                                                                                                                  |
        |                                                    July  1996                                                    |
        |                                                                                                                  |
        |------------------------------------------------------------------------------------------------------------------|
        |      Monday      |     Tuesday      |    Wednesday     |     Thursday     |      Friday      |     Saturday     |
        |------------------+------------------+------------------+------------------+------------------+------------------|
        |        1         |        2         |        3         |        4         |        5         |        6         |
        |                  |                  |                  |                  |                  |                  |
        | Deliver Material |                  | Excavate         |                  |                  |                  |
        |                2 |                  |             4.75 |                  |                  |                  |
        |          $500.00 |                  |        $3,500.00 |                  |                  |                  |
        |------------------+------------------+------------------+------------------+------------------+------------------|
        |        8         |        9         |       10         |       11         |       12         |       13         |
        |                  |*****Vacation*****|                  |                  |                  |                  |
        |                  |                  |                  |                  |                  |                  |
        |                  |                  |                  |                  |                  |                  |
        |                  |                  |                  |                  |                  |                  |
        |------------------+------------------+------------------+------------------+------------------+------------------|
        |       15         |       16         |       17         |       18         |       19         |       20         |
        |                  |                  |                  |                  |                  |                  |
        |                  |                  |                  |                  |                  |                  |
        |                  |                  |                  |                  |                  |                  |
        |                  |                  |                  |                  |                  |                  |
        |------------------+------------------+------------------+------------------+------------------+------------------|
        |       22         |       23         |       24         |       25         |       26         |       27         |
        |                  |                  |                  |                  |                  |                  |
        |                  |                  |                  |                  |                  |                  |
        |                  |                  |                  |                  |                  |                  |
        |                  |                  |                  |                  |                  |                  |
        |------------------+------------------+------------------+------------------+------------------+------------------|
        |       29         |       30         |       31         |                  |                  |                  |
        |                  |                  |                  |                  |                  |                  |
        |                  |                  |                  |                  |                  |                  |
        |                  |                  |                  |                  |                  |                  |
        |                  |                  |                  |                  |                  |                  |
        ------------------------------------------------------------------------------------------------------------------

                                                 -------------------------
                                                 |  Legend  |    Sum     |
                                                 |          |            |
                                                 | task     |            |
                                                 | dur      |            |
                                                 | cost     |  $4,000.00 |
                                                 -------------------------
```

CHAPTER 6

The 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

- 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 on SAS data 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 may prefer to use the Explorer window instead of using PROC CATALOG. The window can do most of what the procedure does.

Syntax: CATALOG Procedure

Tip: Supports RUN-group processing.

Tip: Supports the Output Delivery System. See “Output Delivery System” on page 32 for details.

ODS Table Name: See: “Results: CATALOG Procedure” on page 169

Reminder: You can perform similar functions with the SAS Explorer window and with dictionary tables in the SQL procedure. For information on the Explorer window, see the online Help. For information on PROC SQL, see Chapter 44, “The SQL Procedure,” on page 1027.

See: CATALOG Procedure in the documentation for your operating environment.

PROC CATALOG CATALOG=<*libref.*>*catalog* <ENTRYTYPE=*etype*> <FORCE>
<KILL>;

CONTENTS <OUT=*SAS-data-set*> <FILE=*fileref*>;

COPY OUT=<*libref.*>*catalog* <*options*>;
SELECT *entry(s)* </ ENTRYTYPE=*etype*>;
EXCLUDE *entry(s)* </ 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(s)* </ ENTRYTYPE=*etype*>;

MODIFY *entry* (DESCRIPTION=<<’>*entry-description*<’>>)</ ENTRYTYPE=*etype*>;

SAVE *entry(s)* </ ENTRYTYPE=*etype*>;

To do this	Use this statement
Copy entries from one SAS catalog to another	
Copy or move all entries	COPY (with MOVE option)
Copy entries to a new catalog (overwriting the catalog if it already exists)	COPY (with NEW option)
Copy only selected entries	COPY, SELECT
Copy all *except* the entries specified	COPY, EXCLUDE
Delete entries from a SAS catalog	

To do this	Use this statement
Delete *all* entries	PROC CATALOG (with KILL option)
Delete *all* entries in catalog opened by another resource environment	PROC CATALOG (with FORCE and KILL options)
Delete all *except* the entries specified	SAVE
Alter names and descriptions	
Change the names of catalog entries	CHANGE
Switch the names of two catalog entries	EXCHANGE
Change the description of a catalog entry	MODIFY
Print	
Print the contents of a catalog	CONTENTS

PROC CATALOG Statement

PROC CATALOG CATALOG=<*libref.*>*catalog* <ENTRYTYPE=*etype*> <FORCE> <KILL>;

To do this	Use this option
Restrict processing to one entry type	ENTRYTYPE=
Delete all catalog entries	KILL
Force certain statements to execute on a catalog opened by another resource environment	FORCE

Required Arguments

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.

Options

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.

Interaction: 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.

Interaction: 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 also: "Specifying an Entry Type" on page 166.

Featured in: Example 1 on page 170 and Example 2 on page 174

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. The statements and the catalogs that are affected by FORCE are

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.

Featured in: Example 3 on page 176

KILL
deletes all entries in a SAS catalog.

Interaction: The KILL option deletes all catalog entries even when ENTRYTYPE= is specified.

Interaction: 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 data library. You must use another method, such as PROC DATASETS or the DIR window to delete an empty SAS catalog.

Featured in: Example 3 on page 176

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. △

CHANGE Statement

Renames one or more catalog entries.

Tip: You can change multiple names in a single CHANGE statement or use multiple CHANGE statements.

Featured in: Example 2 on page 174

CHANGE *old-name-1=new-name-1*
<*…old-name-n=new-name-n*>
</ ENTRYTYPE=*etype*>;

Required Arguments

old-name=new-name
specifies the current name of a catalog entry and the new name 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.

Options

ENTRYTYPE=*etype*
restricts processing to one entry type.

See: "The ENTRYTYPE= Option" on page 167

See also: "Specifying an Entry Type" on page 166

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.

Featured in: Example 2 on page 174

CONTENTS <OUT=*SAS-data-set*> <FILE=*fileref*>;

Without Options

The output is sent to the procedure output.

Options

Note: The ENTRYTYPE= (ET=) option is not available for the CONTENTS statement. △

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 this 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.

Tip: Use SELECT or EXCLUDE statements, but not both, after the COPY statement to limit which entries are copied.

Tip: You can copy entries from multiple catalogs in a single PROC step, not just the one specified in the PROC CATALOG statement.

Tip: The ENTRYTYPE= option does not require a forward slash (/) in this statement.

Featured in: Example 1 on page 170

COPY OUT=<*libref.*>*catalog* <*options*>;

To do this	Use this option
Restrict processing to one type of entry	ENTRYTYPE=
Copy from a different catalog in the same step	IN=
Move (copy and then delete) a catalog entry	MOVE
Copy entries to a new catalog (overwriting the catalog if it already exists)	NEW
Protect several types of SAS/AF entries from being edited with PROC BUILD	NOEDIT
Not copy source lines from a PROGRAM, FRAME, or SCL entry	NOSOURCE

Required Arguments

OUT=*<libref.>catalog*
names the catalog to which entries are copied.

Options

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 167

See also: "Specifying an Entry Type" on page 166

IN=*<libref.>catalog*
specifies the catalog to copy.

Interaction: The IN= option overrides a CATALOG= argument that was specified in the PROC CATALOG statement.

Featured in: Example 1 on page 170

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. For information about using the NEW option with concatenated catalogs, see "Catalog Concatenation" on page 168.

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.

Featured in: Example 1 on page 170

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.

Tip: Use DELETE to delete only a few entries; use SAVE when it is more convenient to specify which entries *not* to delete.

Tip: You can specify multiple entries. You can also use multiple DELETE statements.

See also: "SAVE Statement" on page 164

Featured in: Example 1 on page 170

DELETE *entry(s)* </ ENTRYTYPE=*etype*>;

Required Arguments

entry(s)
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.

Options

ENTRYTYPE=*etype*
restricts processing to one entry type.

See: "The ENTRYTYPE= Option" on page 167

See also: "Specifying an Entry Type" on page 166

EXCHANGE Statement

Switches the name of two catalog entries.

Restriction: The catalog entries must be of the same type.

EXCHANGE *name-1=other-name-1*
<*…name-n=other-name-n*>
</ ENTRYTYPE=*etype*>;

Required Arguments

name=other-name
specifies two catalog entry names that the procedure will switch.

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 also: "Specifying an Entry Type" on page 166

Options

ENTRYTYPE=*etype*
restricts processing to one entry type.

See: "The ENTRYTYPE= Option" on page 167

See also: "Specifying an Entry Type" on page 166

EXCLUDE Statement

Specifies entries that the COPY statement does *not* copy.

Restriction: Requires the COPY statement.

Restriction: Do not use the EXCLUDE statement with the SELECT statement.

Tip: You can specify multiple entries in a single EXCLUDE statement.

Tip: You can use multiple EXCLUDE statements with a single COPY statement within a RUN group.

See also: “COPY Statement” on page 158 and “SELECT Statement” on page 165

Featured in: Example 1 on page 170

EXCLUDE *entry(s)* </ ENTRYTYPE=*etype*>;

Required Arguments

entry(s)
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 also: “Specifying an Entry Type” on page 166

Options

ENTRYTYPE=*etype*
restricts processing to one entry type.

See: “The ENTRYTYPE= Option” on page 167

See also: “Specifying an Entry Type” on page 166

MODIFY Statement

Changes the description of a catalog entry.

Featured in: Example 2 on page 174

MODIFY *entry* (DESCRIPTION=<<'>*entry-description*<'>>) </ ENTRYTYPE=*etype*>;

Required Arguments

entry

specifies the name of one SAS catalog entry. Optionally, 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 also: "Specifying an Entry Type" on page 166

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. Optionally, you can enclose the description in single or double quotes.

Alias: DESC

Tip: Use DESCRIPTION= with no text to remove the current description.

Options

ENTRYTYPE=*etype*
restricts processing to one entry type.

See: “The ENTRYTYPE= Option” on page 167

See also: “Specifying an Entry Type” on page 166

SAVE Statement

Specify entries *not* to delete from a SAS catalog.

Restriction: Cannot limit the effects of the KILL option.

Tip: 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.

Tip: You can specify multiple entries and use multiple SAVE statements.

See also: “DELETE Statement” on page 160

SAVE *entry(s)* </ ENTRYTYPE=*etype*>;

Required Arguments

entry(s)
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.

Options

ENTRYTYPE=*etype*
restricts processing to one entry type.

See: “The ENTRYTYPE= Option” on page 167

See also: “Specifying an Entry Type” on page 166

SELECT Statement

Specifies entries that the COPY statement will copy.

Restriction: Requires the COPY statement.

Restriction: Cannot be used with an EXCLUDE statement.

Tip: You can specify multiple entries in a single SELECT statement.

Tip: You can use multiple SELECT statements with a single COPY statement within a RUN group.

See also: "COPY Statement" on page 158 and "EXCLUDE Statement" on page 162

Featured in: Example 1 on page 170

SELECT *entry(s)* </ ENTRYTYPE=*etype*>;

Required Arguments

entry(s)
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.

Options

ENTRYTYPE=*etype*
restricts processing to one entry type.

See: "The ENTRYTYPE= Option" on page 167.

See also: "Specifying an Entry Type" on page 166.

Concepts: CATALOG Procedure

Interactive Processing with RUN Groups

Definition

The CATALOG procedure is interactive. Once you submit a PROC CATALOG statement, you can continue to submit and execute statements or groups of statements without repeating the PROC CATALOG statement.

A set of procedure statements ending with a RUN statement is called a *RUN group*. The changes specified in a given group of statements take effect when a RUN statement is encountered.

How to End a PROC CATALOG Step

In the DATA step and most SAS procedures, a RUN statement is a step boundary and ends the step. A simple RUN statement does not, however, end an interactive procedure. To terminate a PROC CATALOG step, you can

- 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 on page 174.

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. See Table 6.1 on page 167.

Table 6.1 Supplying an Entry Type

You can supply an entry type with...	Example
the entry name	`delete test1.program test1.log test2.log;`
ET= in parentheses	`delete test1 (et=program);`
ET= *after* a slash[1]	`delete test1 (et=program) test1 test2 / et=log;`
ENTRYTYPE= *without* a slash[2]	`proc catalog catalog=mycat et=log; delete test1 test2;`

1 in a subordinate statement
2 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.

If many entries are of one type, but a few are 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.

Alias: ET=

Default: If you omit ENTRYTYPE=, PROC CATALOG processes all entries in the catalog.

Interaction: If you specify ENTRYTYPE= in the PROC CATALOG statement, do not specify either ENTRYTYPE= or (ENTRYTYPE=) in a subordinate statement.

Interaction: (ENTRYTYPE=*etype*) *in parentheses* immediately following an entry name overrides ENTRYTYPE= *in that same statement*.

Tip: On all statements *except* the PROC CATALOG and COPY statements, this option follows a slash.

Tip: To process multiple entry types in a single PROC CATALOG step, use ENTRYTYPE= in a subordinate statement, not in the PROC CATALOG statement.

See also: "Specifying an Entry Type" on page 166.

Featured in: Example 1 on page 170

(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*.

Tip: 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;
```

Tip: For the CHANGE and EXCHANGE statements, specify (ENTRYTYPE=) *in parentheses* only once for each pair of names following the second name in the pair. For example,

```
change old1=new1 (et=log)
       old1=new2 (et=help);
```

See also: "Specifying an Entry Type" on page 166

Featured in: Example 1 on page 170 and Example 2 on page 174

Catalog Concatenation

The CATALOG procedure supports both implicit and explicit concatenation of catalogs. 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 implicit concatenation of A and B, and catalog D is an implicit 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

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 6.2 ODS Tables Produced by the CATALOG Procedure

Table Name	The CATALOG procedure generates this table:
Catalog_Random	when the catalog is in a random-access data library.
Catalog_Sequential	when the catalog is in a sequential data library.

Examples: CATALOG Procedure

Example 1: Copying, Deleting, and Moving Catalog Entries from Multiple Catalogs

Procedure 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=)

QUIT statement

RUN statement

SELECT statement options:

ENTRYTYPE= or ET=

This example

- 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.

Input Catalogs

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
PRTINFO     KPRINTER   Printing Parameters
```

The SAS catalog PERM.FORMATS contains the following entries:

```
REVENUE    FORMAT     FORMAT:MAXLEN=16,16,12
DEPT       FORMATC    FORMAT:MAXLEN=1,1,14
```

Program

Set the SAS system options. Write the source code to the log by specifying the SOURCE SAS system option.

```
options nodate pageno=1 linesize=80 pagesize=60 source;
```

Assign a library reference to a SAS data library. The LIBNAME statement assigns the libref PERM to the SAS data library that contains a permanent SAS catalog.

```
libname perm 'SAS-data-library';
```

Delete two entries from the PERM.SAMPLE catalog.

```
proc catalog cat=perm.sample;
   delete credit.program credit.log;
run;
```

Copy all entries in the PERM.SAMPLE catalog to the WORK.TCATALL catalog.

```
   copy out=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;
```

Log

```
1    libname perm 'SAS-data-library';
NOTE: Directory for library PERM contains files of mixed engine types.
NOTE: Libref PERM was successfully assigned as follows:
      Engine:        V9
      Physical Name: 'SAS-data-library'
2    options nodate pageno=1 linesize=80 pagesize=60 source;
3   proc catalog cat=perm.sample;
4      delete credit.program credit.log;
5   run;
NOTE: Deleting entry CREDIT.PROGRAM in catalog PERM.SAMPLE.
NOTE: Deleting entry CREDIT.LOG in catalog PERM.SAMPLE.
6      copy out=tcatall;
7   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.
NOTE: Copying entry PRTINFO.XPRINTER from catalog PERM.SAMPLE to catalog
      WORK.TCATALL.
```

```
8      copy out=testcat;
9         exclude test1 test2 test3  passist (et=slist) / et=log;
10   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.
NOTE: Copying entry PRTINFO.XPRINTER from catalog PERM.SAMPLE to catalog
      WORK.TESTCAT.
11      copy out=logcat move;
12         select test1 test2 test3 / et=log;
13   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.
14      copy out=perm.finance noedit;
15         select loan.frame loan.help loan.keys loan.pmenu;
16   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.
17      copy in=perm.formats out=perm.finance;
18         select revenue.format dept.formatc;
19   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

Procedure features:

- PROC CATALOG statement
- CHANGE statement options:
  - (ENTRYTYPE=) or (ET=)
- CONTENTS statement options:
  - FILE=
- MODIFY statement
- RUN statement
- QUIT statement

This example

- □ 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

Set the SAS system options. The system option SOURCE writes the source code to the log.

```
options nodate pageno=1 linesize=80 pagesize=60 source;
```

Assign a library reference. The LIBNAME statement assigns a libref to the SAS data library that contains a permanent SAS catalog.

```
libname perm 'SAS-data-library';
```

List the entries in a catalog and route the output to a file. The CONTENTS statement creates a listing of the contents of the SAS catalog PERM.FINANCE and routes the output to a file.

```
proc catalog catalog=perm.finance;
   contents;
title1 'Contents of PERM.FINANCE before changes are made';
run;
```

Change entry names. The CHANGE statement changes the name of an entry that contains a user-written character format. (ET=) specifies the entry type.

```
   change dept=deptcode (et=formatc);
run;
```

Process entries in multiple run groups. The MODIFY statement changes the description of an entry. The CONTENTS statement creates a listing of the contents of PERM.FINANCE after all the changes have been applied. QUIT ends the procedure.

```
   modify loan.frame (description='Loan analysis app. - ver1');
   contents;
title1 'Contents of PERM.FINANCE after changes are made';
run;
quit;
```

Output

```
                   Contents of PERM.FINANCE before changes are made              1

                           Contents of Catalog PERM.FINANCE

# Name    Type            Create Date       Modified Date Description
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
1 REVENUE FORMAT   16OCT1996:13:48:11  16OCT1996:13:48:11 FORMAT:MAXLEN=16,16,12
2 DEPT    FORMATC  30OCT1996:13:40:42  30OCT1996:13:40:42 FORMAT:MAXLEN=1,1,14
3 LOAN    FRAME    30OCT1996:13:40:43  30OCT1996:13:40:43 Loan analysis
                                                          application
4 LOAN    HELP     16OCT1996:13:48:10  16OCT1996:13:48:10 Information about
                                                          the application
5 LOAN    KEYS     16OCT1996:13:48:10  16OCT1996:13:48:10 Custom key definitions
                                                          for application
6 LOAN    PMENU    16OCT1996:13:48:10  16OCT1996:13:48:10 Custom menu
                                                          definitions for
                                                          application
7 LOAN    SCL      16OCT1996:13:48:10  16OCT1996:13:48:10 SCL code for loan
                                                          analysis application
```

```
                   Contents of PERM.FINANCE after changes are made               2

                           Contents of Catalog PERM.FINANCE

# Name     Type            Create Date       Modified Date Description
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
1 REVENUE  FORMAT   16OCT1996:13:48:11  16OCT1996:13:48:11 FORMAT:MAXLEN=
                                                           16,16,12
2 DEPTCODE FORMATC  30OCT1996:13:40:42  30OCT1996:13:40:42 FORMAT:MAXLEN=1,1,14
3 LOAN     FRAME    30OCT1996:13:40:43  11FEB2002:13:20:50 Loan analysis
                                                           app. - ver1
4 LOAN     HELP     16OCT1996:13:48:10  16OCT1996:13:48:10 Information about
                                                           the application
5 LOAN     KEYS     16OCT1996:13:48:10  16OCT1996:13:48:10 Custom key
                                                           definitions for
                                                           application
6 LOAN     PMENU    16OCT1996:13:48:10  16OCT1996:13:48:10 Custom menu
                                                           definitions for
                                                           application
7 LOAN     SCL      16OCT1996:13:48:10  16OCT1996:13:48:10 SCL code for loan
                                                           analysis application
```

Example 3: Using the FORCE Option with the KILL Option

Procedure features:

PROC CATALOG statement:

- CATALOG= argument
- KILL option
- FORCE option

QUIT statement

RUN statement

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

Start a process (resource environment) 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 will not work and an error is sent to the log.

```
proc catalog c=work.sasmacr kill;
run;
quit;
```

Log

```
ERROR: You cannot open WORK.SASMACR.CATALOG for update access because
       WORK.SASMACR.CATALOG is in use by you in resource environment
       Line Mode Process.
WARNING: Command CATALOG not processed because of errors noted above.
NOTE: The SAS System stopped processing this step because of errors.
NOTE: PROCEDURE CATALOG used (Total process time):
      real time           0.04 seconds
      cpu time            0.03 seconds
```

Add the FORCE Option to the PROC CATALOG Statement

Add the FORCE option to the KILL option to delete the catalog entries.

```
proc catalog c=work.sasmacr kill force;
run;
quit;
```

Log

```
NOTE: Deleting entry MATT.MACRO in catalog WORK.SASMACR.
```

CHAPTER

7

The 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.

Output 7.1 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:

```
options nodate pageno=1 linesize=80
        pagesize=30;

proc chart data=survey;
   vbar response / sumvar=count
        midpoints='Always' 'Usually'
           'Sometimes' 'Rarely' 'Never';
run;
```

Output 7.1 Vertical Bar Chart

```
                             The SAS System                                1

Count Sum

200 +                   *****
    |                   *****
    |                   *****
    |                   *****
    |                   *****
150 +                   *****
    |                   *****
    |                   *****
    |                   *****       *****
    |       *****       *****       *****
100 +       *****       *****       *****       *****
    |       *****       *****       *****       *****
    |       *****       *****       *****       *****
    |       *****       *****       *****       *****
    |       *****       *****       *****       *****
 50 +       *****       *****       *****       *****
    |       *****       *****       *****       *****       *****
    |       *****       *****       *****       *****       *****
    |       *****       *****       *****       *****       *****
    |       *****       *****       *****       *****       *****
    --------------------------------------------------------------------
           Always      Usually    Sometimes    Rarely       Never

                                  Response
```

Output 7.2 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:

```
options nodate pageno=1 linesize=80
        pagesize=60;

proc chart data=survey;
   hbar response / sumvar=count
        midpoints='Always' 'Usually'
           'Sometimes' 'Rarely' 'Never';
run;
```

Output 7.2 Horizontal Bar Chart

```
                              The SAS System                              1

     Response                                                     Count
                                                                    Sum
                 |
     Always      |*********************                          106.0000
                 |
     Usually     |****************************************       202.0000
                 |
     Sometimes   |************************                       119.0000
                 |
     Rarely      |*******************                             97.0000
                 |
     Never       |*********                                       44.0000
                 |
                 ----+---+---+---+---+---+---+---+---+---+
                     20  40  60  80 100 120 140 160 180 200

                                  Count Sum
```

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. Output 7.3 shows the number of each survey response in the form of a block chart.

```
options nodate pageno=1 linesize=80
        pagesize=30;

proc chart data=survey;
   block response / sumvar=count
         midpoints='Always' 'Usually'
            'Sometimes' 'Rarely' 'Never';
run;
```

Output 7.3 Block Chart

```
                                The SAS System                               1

                           Sum of Count by Response

                        ___
                       /_ /|
                      |**| |
                      |**| |
                      |**| |          ___
          ___         |**| |         /_ /|          ___
         /_ /|        |**| |        |**| |         /_ /|
       -|**| |--------|**| |--------|**| |--------|**| |----------------------
      / |**| |      / |**| |      / |**| |      / |**| |      /   ___        /
     /  |**| |     /  |**| |     /  |**| |     /  |**| |     /   /_ /|     /
    /   |**| |    /   |**| |    /   |**| |    /   |**| |    /   |**| |    /
   /    |**|/    /    |**|/    /    |**|/    /    |**|/    /    |**|/    /
  /             /             /             /             /             /
 /      106    /      202    /      119    /       97    /       44    /
/-------------/-------------/-------------/-------------/-------------/

   Always        Usually      Sometimes      Rarely         Never

                              Response
```

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. Output 7.4 shows the survey results divided by response into five pie slices. The following statements produce the output:

```
options nodate pageno=1 linesize=80
        pagesize=35;

proc chart data=survey;
   pie response / sumvar=count;
run;
```

Output 7.4 Pie Chart

```
                      The SAS System                                1

                 Sum of Count by Response


                             Never
                       ***********
          Rarely   ****    .      ****
                 **        .      .   **
               **          . 44   .     **
              *            .7.75%.        *   Always
            **      97     .    ..         **
           **     17.08%   .   .            **
           * ..            .   .     106     *
          *     ..          . .    18.66%     *
          *        .. .     ..                *
          *            . .                    *
          *                 +  . . .. . .. . .*
          *     119                           *
          *   20.95%     ..                   *
Sometimes *             .                     *
           *           .                     *
           **         .        202          **
             *      ..       35.56%        *
              *    .                      *
               ** .                     **
                 **                   **
                   ****           ****
                       ***********    Usually
```

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, or for data like these in which the categories have an inherent order ("always" meaning more frequent than "usually" which means more frequent than "sometimes"). Output 7.5 shows the survey data displayed in a star chart. The following statements produce the output:

```
options nodate pageno=1 linesize=80
        pagesize=60;

proc chart data=survey;
   star response / sumvar=count;
run;
```

Output 7.5 Star Chart

```
                                 The SAS System                                1

Center = 0                  Sum of Count by Response               Outside = 202


                                                 Never
                                 *************    44
                            *****             *****
                         ***                       ***
                      ***                             ***
                    **                                   **
                   *                                       *
         Rarely  **                                         **
           97   *                                             *
              **                                               **
              *                                                 *
             *                                                   *
            **                                                   **
            *               *......                               *
           **               . ..   ......*.                       **
           *               ..   ..      .. ...                     *
           *               .      ..    .     ....                 *
           *               .        .. ..         ...              *
           *               .          .+..............*            *  Always
           *              .         .. ..             .            *   106
           *              .      ...    .            .             *
           *              .     .        .           .             *
           **            ..  ...         .          ..           **
            *            . ..             .         .             *
            **           *.               ..        .            **
             *             ..              .       ..            *
              *              ..             .      .            *
               *               ..           .     ..           *
                *                ...        ..    .           *
      Sometimes  **                 ..       .    .         **
         119       *                  ...     .  ..        *
                    **                   .    .  .       **
                      ***                 ...  . .    ***
                         ***                 ....  ***
                            *****             *.***
                                 *************   Usually
                                                   202
```

Syntax: CHART Procedure

Requirement: You must use at least one of the chart-producing statements.

Tip: Supports the Output Delivery System. See “Output Delivery System” on page 32 for details.

ODS Table Names: See: “ODS Table Names” on page 198

Reminder: You can use the ATTRIB, FORMAT, LABEL, and WHERE statements. See Chapter 3, “Statements with the Same Function in Multiple Procedures,” on page 57 for details. You can also use any global statements. See “Global Statements” on page 18 for a list.

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)*>;

PROC CHART Statement

PROC CHART <*option(s)*>;

Options

DATA=*SAS-data-set*
identifies the input SAS data set.

Main discussion: "Input Data Sets" on page 19

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.

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.

Range: PROC CHART uses 6 of the 20 formatting characters that SAS provides. Table 7.1 on page 186 shows the formatting characters that PROC CHART uses. Figure 7.1 on page 186 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 that they are listed. For instance, 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)='*#'
```

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 instance the following option assigns the hexadecimal character 2D 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 also: For information on which hexadecimal codes to use for which characters, consult the documentation for your hardware.

Table 7.1 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.
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 7.1 Formatting Characters Commonly Used in PROC CHART Output

```
                    Mean Yearly Pie Sales Grouped by Flavor                      1
                           within Bakery Location

 Pies_Sold Mean

 400 +
     |  ***                       ***
 300 +--***-------***-------------***--------***----------------------------------
     |  ***       ***             ***        ***         ***
 200 +--***--***--***-------------***--***--***---------***-------***--------------
     |  ***  ***  ***             ***  ***  ***         ***       ***
 100 +--***--***--***-------------***--***--***---------***--***--***--------------
     |  ***  ***  ***  ***        ***  ***  ***  ***    ***  ***  ***  ***
     ------------------------------------------------------------------------------
        a    b    c    r     |    a    b    c    r      a    b    c    r    Flavor
        p    l    h    h     |    p    l    h    h      p    l    h    h
        p    u    e    u     |    p    u    e    u      p    u    e    u
        l    e    r    b     |    l    e    r    b      l    e    r    b
        e    b    r    a     |    e    b    r    a      e    b    r    a
             e    y    r     |         e    y    r           e    y    r
             r         b     |         r         b           r         b
             r               |         r                     r

     |---- Clyde ----|   |   |------ Oak -----|    |---- Samford ---|   Bakery
```

LPI=*value*

specifies the proportions of PIE and STAR charts. The *value* is determined by

$$(\text{lines per inch} \,/\, \text{columns per inch}) \; * \; 10$$

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.

Featured in: Example 6 on page 210

BLOCK *variable(s)* </ *option(s)*>;

Required Arguments

variable(s)
specifies the variables for which PROC CHART produces a block chart, one chart for each variable.

Options

The options available on the BLOCK, HBAR, PIE, STAR, and VBAR statements are documented in "Customizing All Types of Charts" on page 191.

Statement Results

Because each block chart must fit on one output page, you may 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.

Table 7.2 on page 187 shows the maximum number of charted values of BLOCK variables for selected LINESIZE= (LS=) specifications that can fit on a 66-line page.

Table 7.2 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.

Main discussion: "BY" on page 58

Featured in: Example 6 on page 210

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 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*.

Options

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.

Featured in: Example 5 on page 209

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.

Options

The options available on the BLOCK, HBAR, PIE, STAR, and VBAR statements are documented in "Customizing All Types of Charts" on page 191.

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.

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.

Options

The options available on the BLOCK, HBAR, PIE, STAR, and VBAR statements are documented in "Customizing All Types of Charts" on page 191.

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 on the PROC CHART statement. See the description of LPI= on page 186 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.

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.

Options

The options available on the BLOCK, HBAR, PIE, STAR, and VBAR statements are documented in "Customizing All Types of Charts" on page 191.

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= option on page 193 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= option on page 186.

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.

Featured in: Example 1 on page 198, Example 2 on page 201, Example 3 on page 203, Example 4 on page 206

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.

Options

The options available on the BLOCK, HBAR, PIE, STAR, and VBAR statements are documented in "Customizing All Types of Charts" on page 191.

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.

Customizing All Types of Charts

Many options in PROC CHART are valid in more than one statement. This section describes the options that you can use on the chart-producing statements.

To do this	Use this option
Specify that numeric variables are discrete	DISCRETE
Specify a frequency variable	FREQ=
Specify that missing values are valid levels	MISSING

To do this	Use this option
Specify the variable for which values or means are displayed	SUMVAR=
Specify the statistic represented in the chart	TYPE=
Specify groupings	
Group the bars in side-by-side charts	GROUP=
Specify that group percentages sum to 100	G100
Group the bars in side-by-side charts	GROUP=
Specify the number of bars for continuous variables	LEVELS=
Define ranges for continuous variables	MIDPOINTS=
Divide the bars into categories	SUBGROUP=
Compute statistics	
Compute the cumulative frequency for each bar	CFREQ
Compute the cumulative percentage for each bar	CPERCENT
Compute the frequency for each bar	FREQ
Compute the mean of the observations for each bar	MEAN
Compute the percentage of total observations for each bar	PERCENT
Compute the total number of observations for each bar	SUM
Control output format	
Print the bars in ascending order of size	ASCENDING
Specify the values for the response axis	AXIS=
Print the bars in descending order of size	DESCENDING
Specify extra space between groups of bars	GSPACE=
Suppress the default header line	NOHEADER
Allow no space between vertical bars	NOSPACE
Suppress the statistics	NOSTATS
Suppress the subgroup legend or symbol table	NOSYMBOL
Suppress the bars with zero frequency	NOZEROS
Draw reference lines	REF=
Specify the spaces between bars	SPACE=
Specify the symbols within bars or blocks	SYMBOL=
Specify the width of bars	WIDTH=

Options

ASCENDING

prints the bars and any associated statistics in ascending order of size within groups.

Alias: ASC

Restriction: Available only on the HBAR and VBAR statements

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;
```

Restriction: Not available on the PIE statement

Restriction: Values must be uniformly spaced, even if you specify them individually.

Restriction: For frequency charts, values must be integers.

Interaction: 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.

Interaction: 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.

Interaction: 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.

Interaction: 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 on the HBAR statement

CPERCENT

prints the cumulative percentages.

Restriction: Available only on the HBAR statement

DESCENDING

prints the bars and any associated statistics in descending order of size within groups.

Alias: DESC

Restriction: Available only on the HBAR and VBAR statements

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 on 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.

Restriction: Available only on the BLOCK, HBAR, and VBAR statements

Featured in: Example 4 on page 206, Example 5 on page 209, Example 6 on page 210

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.

Restriction: Available only on the HBAR and VBAR statements

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 percent 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 percent; however, with G100, the three bars for females add to 100 percent, and the three bars for males add to 100 percent.

Restriction: Available only on the BLOCK, HBAR, and VBAR statements

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.

Restriction: Available only on the HBAR statement and only when you use SUMVAR= and TYPE=

Restriction: Not available when TYPE=CFREQ, CPERCENT, FREQ, or PERCENT

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.

NOHEADER
suppresses the default header line printed at the top of a chart.

Alias: NOHEADING

Restriction: Available only on the BLOCK, PIE, and STAR statements

Featured in: Example 6 on page 210

NOSTATS
suppresses the statistics on a horizontal bar chart.

Alias: NOSTAT

Restriction: Available only on the HBAR statement

NOSYMBOL
suppresses printing of the subgroup symbol or legend table.

Alias: NOLEGEND

Restriction: Available only on the BLOCK, HBAR, and VBAR statements

Interaction: PROC CHART ignores NOSYMBOL if you omit SUBGROUP=.

NOZEROS
suppresses any bar with zero frequency.

Restriction: Available only on the HBAR and VBAR statements

PERCENT
prints the percentages of observations having a given value for the chart variable.

Restriction: Available only on the HBAR statement

REF=*value(s)*
draws reference lines on the response axis at the specified positions.

Restriction: Available only on the HBAR and VBAR statements

Tip: The REF= values should correspond to values of the TYPE= statistic.

Featured in: Example 4 on page 206

SPACE=*n*
specifies the amount of space between individual bars.

Restriction: Available only on the HBAR and VBAR statements

Tip: Use SPACE=0 to leave no space between adjacent bars.

Tip: 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 may be higher or lower than the same bar without the SUBGROUP= option.

Restriction: Available only on the BLOCK, HBAR, and VBAR statements

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, then subdivides the bar into the percentages that each subgroup contributes.

Featured in: Example 3 on page 203

SUM

prints the total number of observations that each bar represents.

Restriction: Available only on the HBAR statement and only when you use both SUMVAR= and TYPE=

Restriction: 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: Both HBAR and VBAR charts can print labels for SUMVAR= variables if you use a LABEL statement.

Featured in: Example 3 on page 203, Example 4 on page 206, Example 5 on page 209, Example 6 on page 210

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 (*)

Restriction: Available only on the BLOCK, HBAR, and VBAR statements

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.

Featured in: Example 6 on page 210

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 only compute MEAN and FREQ statistics.

Featured in: Example 4 on page 206

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

Featured in: Example 2 on page 201

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 only compute SUM and FREQ statistics.

WIDTH=*n*
specifies the width of the bars on bar charts.

Restriction: Available only on the HBAR and VBAR statements

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 explicitly 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.

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 7.3 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

Procedure features:
VBAR statement

This example produces a vertical bar chart that shows a frequency count for the values of the chart variable.

Program

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 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

The frequency chart shows the store’s sales of the shirt for the week: 9 large shirts, 11 medium shirts, and 6 small shirts.

```
         Number of Each Shirt Size Sold                         1

Frequency

11 +                   *****
   |                   *****
   |                   *****
   |                   *****
10 +                   *****
   |                   *****
   |                   *****
   |                   *****
 9 +       *****       *****
   |       *****       *****
   |       *****       *****
   |       *****       *****
 8 +       *****       *****
   |       *****       *****
   |       *****       *****
   |       *****       *****
 7 +       *****       *****
   |       *****       *****
   |       *****       *****
   |       *****       *****
 6 +       *****       *****       *****
   |       *****       *****       *****
   |       *****       *****       *****
   |       *****       *****       *****
 5 +       *****       *****       *****
   |       *****       *****       *****
   |       *****       *****       *****
   |       *****       *****       *****
 4 +       *****       *****       *****
   |       *****       *****       *****
   |       *****       *****       *****
   |       *****       *****       *****
 3 +       *****       *****       *****
   |       *****       *****       *****
   |       *****       *****       *****
   |       *****       *****       *****
 2 +       *****       *****       *****
   |       *****       *****       *****
   |       *****       *****       *****
   |       *****       *****       *****
 1 +       *****       *****       *****
   |       *****       *****       *****
   |       *****       *****       *****
   |       *****       *****       *****
   ---------------------------------------------
           large      medium       small

                       Size
```

Example 2: Producing a Percentage Bar Chart

Procedure features:
VBAR statement option:
TYPE=

Data set: SHIRTS on page 199

This example produces a vertical bar chart. The chart statistic is the percentage for each category of the total number of shirts sold.

Program

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 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

The 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                   1

                  Percentage

                     |                   *****
                     |                   *****
                  40 +                   *****
                     |                   *****
                     |                   *****
                     |                   *****
                     |                   *****
                  35 +       *****       *****
                     |       *****       *****
                     |       *****       *****
                     |       *****       *****
                     |       *****       *****
                  30 +       *****       *****
                     |       *****       *****
                     |       *****       *****
                     |       *****       *****
                     |       *****       *****
                  25 +       *****       *****
                     |       *****       *****
                     |       *****       *****       *****
                     |       *****       *****       *****
                     |       *****       *****       *****
                  20 +       *****       *****       *****
                     |       *****       *****       *****
                     |       *****       *****       *****
                     |       *****       *****       *****
                     |       *****       *****       *****
                  15 +       *****       *****       *****
                     |       *****       *****       *****
                     |       *****       *****       *****
                     |       *****       *****       *****
                     |       *****       *****       *****
                  10 +       *****       *****       *****
                     |       *****       *****       *****
                     |       *****       *****       *****
                     |       *****       *****       *****
                     |       *****       *****       *****
                   5 +       *****       *****       *****
                     |       *****       *****       *****
                     |       *****       *****       *****
                     |       *****       *****       *****
                     |       *****       *****       *****
                     --------------------------------------------
                             large      medium       small

                                         Size
```

Example 3: Subdividing the Bars into Categories

Procedure features:

VBAR statement options:

SUBGROUP=
SUMVAR=

This example

- 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

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 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      1995  234
Samford  apple      1996  288
Samford  blueberry  1995  103
Samford  blueberry  1996  143
Samford  cherry     1995  173
Samford  cherry     1996  195
Samford  rhubarb    1995   26
Samford  rhubarb    1996   28
Oak      apple      1995  319
Oak      apple      1996  371
Oak      blueberry  1995  174
Oak      blueberry  1996  206
Oak      cherry     1995  246
Oak      cherry     1996  311
Oak      rhubarb    1995   51
Oak      rhubarb    1996   56
Clyde    apple      1995  313
Clyde    apple      1996  415
Clyde    blueberry  1995  177
Clyde    blueberry  1996  201
```

```
Clyde    cherry     1995  250
Clyde    cherry     1996  328
Clyde    rhubarb    1995   60
Clyde    rhubarb    1996   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

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, and 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              1

      Pies_Sold Sum

           |       SSSSS
           |       SSSSS
           |       SSSSS
      1800 +       SSSSS
           |       SSSSS
           |       SSSSS
           |       SSSSS
      1600 +       SSSSS
           |       SSSSS
           |       SSSSS                   SSSSS
           |       OOOOO                   SSSSS
      1400 +       OOOOO                   SSSSS
           |       OOOOO                   SSSSS
           |       OOOOO                   SSSSS
           |       OOOOO                   SSSSS
      1200 +       OOOOO                   SSSSS
           |       OOOOO                   OOOOO
           |       OOOOO                   OOOOO
           |       OOOOO       SSSSS       OOOOO
      1000 +       OOOOO       SSSSS       OOOOO
           |       OOOOO       SSSSS       OOOOO
           |       OOOOO       SSSSS       OOOOO
           |       OOOOO       SSSSS       OOOOO
       800 +       OOOOO       OOOOO       OOOOO
           |       CCCCC       OOOOO       OOOOO
           |       CCCCC       OOOOO       OOOOO
           |       CCCCC       OOOOO       OOOOO
       600 +       CCCCC       OOOOO       CCCCC
           |       CCCCC       OOOOO       CCCCC
           |       CCCCC       OOOOO       CCCCC
           |       CCCCC       OOOOO       CCCCC
       400 +       CCCCC       CCCCC       CCCCC
           |       CCCCC       CCCCC       CCCCC
           |       CCCCC       CCCCC       CCCCC
           |       CCCCC       CCCCC       CCCCC       SSSSS
       200 +       CCCCC       CCCCC       CCCCC       OOOOO
           |       CCCCC       CCCCC       CCCCC       OOOOO
           |       CCCCC       CCCCC       CCCCC       CCCCC
           |       CCCCC       CCCCC       CCCCC       CCCCC
           --------------------------------------------------------
                   apple     blueberr     cherry      rhubarb

                                  Flavor


           Symbol Bakery      Symbol Bakery      Symbol Bakery

              C   Clyde          O   Oak            S   Samford
```

Example 4: Producing Side-by-Side Bar Charts

Procedure features:

VBAR statement options:

GROUP=
REF=
SUMVAR=
TYPE=

Data set: PIESALES"Program" on page 203

This example

- □ 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

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 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 1995 and 1996 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

The 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                    1
                            within Bakery Location

Pies_Sold Mean

    |  ***
350 +  ***                   ***
    |  ***                   ***
    |  ***                   ***
    |  ***                   ***
    |  ***                   ***
300 +--***-------------------***----------------------------------------------
    |  ***       ***         ***
    |  ***       ***         ***       ***
    |  ***       ***         ***       ***
    |  ***       ***         ***       ***         ***
250 +  ***       ***         ***       ***         ***
    |  ***       ***         ***       ***         ***
    |  ***       ***         ***       ***         ***
    |  ***       ***         ***       ***         ***
    |  ***       ***         ***       ***         ***
200 +--***-------***---------***-------***---------***------------------------
    |  ***  ***  ***         ***  ***  ***         ***
    |  ***  ***  ***         ***  ***  ***         ***       ***
    |  ***  ***  ***         ***  ***  ***         ***       ***
    |  ***  ***  ***         ***  ***  ***         ***       ***
150 +  ***  ***  ***         ***  ***  ***         ***       ***
    |  ***  ***  ***         ***  ***  ***         ***       ***
    |  ***  ***  ***         ***  ***  ***         ***       ***
    |  ***  ***  ***         ***  ***  ***         ***  ***  ***
    |  ***  ***  ***         ***  ***  ***         ***  ***  ***
100 +--***--***--***---------***--***--***---------***--***--***--------------
    |  ***  ***  ***         ***  ***  ***         ***  ***  ***
    |  ***  ***  ***         ***  ***  ***         ***  ***  ***
    |  ***  ***  ***         ***  ***  ***         ***  ***  ***
    |  ***  ***  ***  ***    ***  ***  ***         ***  ***  ***
 50 +  ***  ***  ***  ***    ***  ***  ***  ***    ***  ***  ***
    |  ***  ***  ***  ***    ***  ***  ***  ***    ***  ***  ***
    |  ***  ***  ***  ***    ***  ***  ***  ***    ***  ***  ***  ***
    |  ***  ***  ***  ***    ***  ***  ***  ***    ***  ***  ***  ***
    |  ***  ***  ***  ***    ***  ***  ***  ***    ***  ***  ***  ***
    --------------------------------------------------------------------------
        a    b    c    r      a    b    c    r      a    b    c    r    Flavor
        p    l    h    h      p    l    h    h      p    l    h    h
        p    u    e    u      p    u    e    u      p    u    e    u
        l    e    r    b      l    e    r    b      l    e    r    b
        e    b    r    a      e    b    r    a      e    b    r    a
             e    y    r           e    y    r           e    y    r
             r         b           r         b           r         b
             r                     r                     r

       |----- Clyde ----|    |------ Oak -----|    |---- Samford ---|   Bakery
```

Example 5: Producing a Horizontal Bar Chart for a Subset of the Data

Procedure features:
HBAR statement options:
GROUP=
SUMVAR=

Other features:
WHERE= data set option

Data set: PIESALES"Program" on page 203

This example

- □ 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

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 variable value limitation for the horizontal bar chart. WHERE= limits the chart to only the 1995 sales totals.

```
proc chart data=piesales(where=(year=1995));
```

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 '1995 Pie Sales for Each Bakery According to Flavor';
run;
```

Output

```
              1995 Pie Sales for Each Bakery According to Flavor            1

Flavor     Bakery                                                     Pies_Sold
                                                                            Sum
                      |
apple      Clyde      |******************************************      313.0000
           Oak        |*******************************************     319.0000
           Samford    |*******************************             234.0000
                      |
blueberr   Clyde      |************************                    177.0000
           Oak        |***********************                     174.0000
           Samford    |**************                              103.0000
                      |
cherry     Clyde      |*********************************           250.0000
           Oak        |*********************************           246.0000
           Samford    |***********************                     173.0000
                      |
rhubarb    Clyde      |********                                     60.0000
           Oak        |*******                                      51.0000
           Samford    |***                                          26.0000
                      |
                     ----+---+---+---+---+---+---+---+---+---+---
                        30  60  90 120 150 180 210 240 270 300

                                    Pies_Sold Sum
```

Example 6: Producing Block Charts for BY Groups

Procedure 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“Program” on page 203

This example

□ 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

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 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. OVP allows overprinted characters in the charts.

```
options nobyline ovp;
```

Specify the BY group for multiple block charts. The BY statement produces one chart for 1995 sales and one for 1996 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';
```

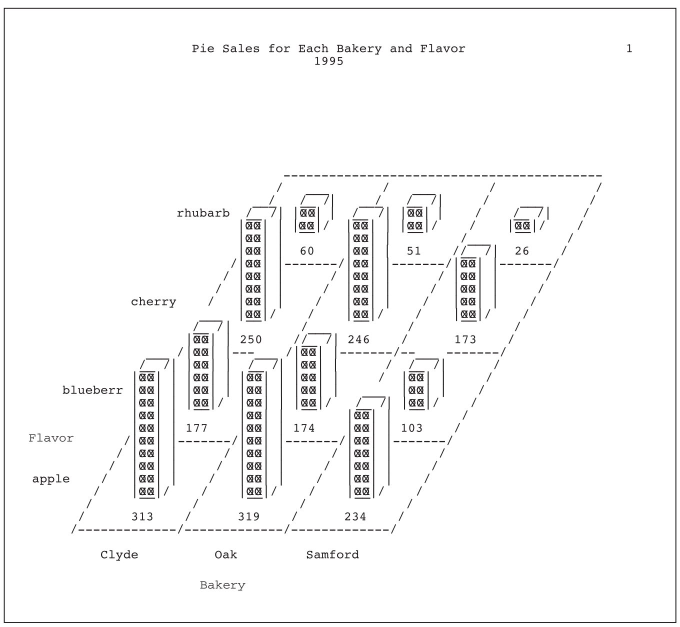
Specify the titles. The #BYVAL specification inserts the year into the second line of the title.

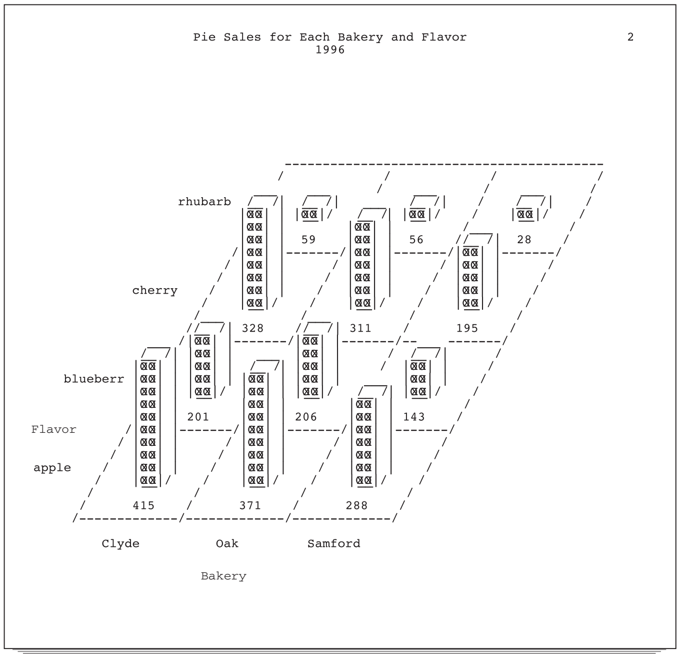
```
   title  'Pie Sales for Each Bakery and Flavor';
   title2 '#byval(year)';
run;
```

Reset the printing of the default BY line. The SAS system option BYLINE resets the printing of the default BY line.

```
options byline;
```

Output





References

Nelder, J.A. (1976), "A Simple Algorithm for Scaling Graphs," *Applied Statistics*, Volume 25, Number 1, London: The Royal Statistical Society.

Terrell, G.R. and Scott, D.W. (1985), "Oversmoothed Nonparametric Density Estimates," *Journal of the American Statistical Association*, 80, 389, 209–214.

CHAPTER 8

The 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 data library. *Transport files* are sequential files that each contain a SAS data 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 can read only transport files that PROC CPORT creates. For information on the transport files that the transport engine creates, see the section on SAS files in *SAS Language Reference: Concepts*.

PROC CIMPORT 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. In such cases, PROC CIMPORT automatically converts the contents of the transport file as it imports it.

PROC CIMPORT produces no output, but it does write notes to the SAS log.

General File Transport Process

To export and import files, follow these steps:

1 Use PROC CPORT to export the SAS files that you want to transport.

2 If you are changing operating environments, move the transport file to the new machine by using either communications software or a magnetic medium.

Note: If you use communications software to move the transport file, be sure that it treats the transport file as a *binary* file and that it modifies neither the attributes nor the contents of the file. △

3 Use PROC CIMPORT to translate the transport file into the format appropriate for the new operating environment or release.

Syntax: CIMPORT Procedure

See: CIMPORT Procedure in the documentation for your operating environment.

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*>;

PROC CIMPORT Statement

PROC CIMPORT *destination=libref* | <*libref.*> *member-name*<*option(s)*>;

To do this	Use this option
Identify the input transport file	
Specify a previously defined fileref or the filename of the transport file to read	INFILE=
Read the input transport file from a tape	TAPE
Select files to import	
Exclude specified entry types from the import process	EET=
Specify entry types to import	ET=
Control the contents of the transport file	
Import a SAS file without changing the created and modified date and time	DATECOPY
Specify whether to extend by 1 byte the length of short numerics (less than 8 bytes) when you import them	EXTENDSN=
Specify that only data sets, only catalogs, or both, be moved when a library is imported	MEMTYPE=

To do this	Use this option
Enable access to a locked catalog	FORCE
Create a new catalog for the imported transport file, and delete any existing catalog with the same name	NEW
Import SAS/AF PROGRAM and SCL entries without edit capability	NOEDIT
Suppress the importing of source code for SAS/AF entries that contain compiled SCL code	NOSRC

Required Arguments

destination=libref* | < *libref.* >*member-name

identifies the type of file to import and specifies the specific catalog, SAS data set, or SAS data library to import.

destination

identifies the file or files in the transport file as a single catalog, as a single SAS data set, or as the members of a SAS data 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 data 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*.

Options

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 destination 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 on the MODIFY statement“MODIFY Statement” on page 348 in a PROC DATASETS step.

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.

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 allows 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=

Featured in: Example 1 on page 222.

MEMTYPE=*mtype*
specifies that only data sets, only catalogs, or both, be moved when a SAS library is imported. Values for *mtype* can be

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 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 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.

EXCLUDE Statement

Excludes specified files or entries from the import process.

Tip: There is no limit to the number of EXCLUDE statements you can use in one invocation of PROC CIMPORT.

Interaction: You can use either EXCLUDE statements or SELECT statements in a PROC CIMPORT step, but not both.

EXCLUDE *SAS file(s)* | *catalog entry(s)*</ MEMTYPE=*mtype*></ ENTRYTYPE=*entry-type*>;

Required Arguments

SAS file(s) | catalog entry(s)

specifies either the name(s) of one or more SAS files or the name(s) of one or more catalog entries to be excluded from the import process. Specify SAS filenames if you import a data 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 24.

Options

ENTRYTYPE=*entry-type*
specifies a single entry type for the catalog entry(s) listed in the EXCLUDE statement. See *SAS Language Reference: Concepts* for a complete list of catalog entry types.

Restriction: ENTRYTYPE= is valid only when you import an individual SAS catalog.

Alias: ETYPE=, ET=

MEMTYPE=*mtype*
specifies a single member type for the SAS file(s) 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.

Restriction: MEMTYPE= is valid only when you import a SAS data library.

Alias: MTYPE=, MT=

Default: ALL

SELECT Statement

Specifies individual files or entries to import.

Tip: There is no limit to the number of SELECT statements you can use in one invocation of PROC CIMPORT.

Interaction: You can use either EXCLUDE statements or SELECT statements in a PROC CIMPORT step, but not both.

Featured in: Example 2 on page 223

SELECT *SAS file(s)* | *catalog entry(s)*</ MEMTYPE=*mtype*></ ENTRYTYPE=*entry-type*>;

Required Arguments

SAS file(s) | catalog entry(s)
specifies either the name(s) of one or more SAS files or the name(s) of one or more catalog entries to import. Specify SAS filenames if you import a data 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 24.

Options

ENTRYTYPE=*entry-type*
specifies a single entry type for the catalog entry(s) listed in the SELECT statement. See *SAS Language Reference: Concepts* for a complete list of catalog entry types.

Restriction: ENTRYTYPE= is valid only when you import an individual SAS catalog.

Alias: ETYPE=, ET=

MEMTYPE=*mtype*
specifies a single member type for the SAS file(s) 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.

Restriction: MEMTYPE= is valid only when you import a SAS data library.

Alias: MTYPE=, MT=

Default: ALL

Results: CIMPORT Procedure

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

Note: A BLKSIZE value of less than 8000 may be more efficient for your storage device in some cases. The BLKSIZE value must be an exact multiple of the LRECL value. △

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.

Examples: CIMPORT Procedure

Example 1: Importing an Entire Data Library

Procedure features:

PROC CIMPORT statement option:

INFILE=

This example shows how to use PROC CIMPORT to read from disk a transport file, named TRANFILE, that PROC CPORT created from a SAS data 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 data library, called NEWLIB, in the new operating environment.

Program

Specify the library name and filename. The LIBNAME statement specifies a libname for the new SAS data 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 data library in the NEWLIB library. PROC CIMPORT imports the SAS data 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

Procedure features:

PROC CIMPORT statement options:

INFILE=

SELECT statement

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

Specify the library name, filename, and operating environment options. The LIBNAME statement specifies a libname for the new SAS data 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

Procedure features:
PROC CIMPORT statement option:
INFILE=

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

Specify the library name, filename, and operating environment options. The LIBNAME statement specifies a libname for the new SAS data 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 9

The 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 explicitly 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 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:

```
proc compare base=proclib.one
             compare=proclib.two novalues;
run;
```

Output 9.1 Comparison of Two Data Sets

```
                                The SAS System                              1

                              COMPARE Procedure
                   Comparison of PROCLIB.ONE with PROCLIB.TWO
                                (Method=EXACT)

                              Data Set Summary

Dataset                Created          Modified  NVar    NObs  Label

PROCLIB.ONE  13MAY98:15:01:42  13MAY98:15:01:42     5       4  First Data Set
PROCLIB.TWO  13MAY98:15:01:44  13MAY98:15:01:44     6       5  Second Data Set


                              Variables Summary

              Number of Variables in Common: 5.
              Number of Variables in PROCLIB.TWO but not in PROCLIB.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   PROCLIB.ONE  Num        8
                               PROCLIB.TWO  Char       8


            Listing of Common Variables with Differing Attributes

          Variable  Dataset      Type  Length  Format  Label

          year      PROCLIB.ONE  Char       8          Year of Birth
                    PROCLIB.TWO  Char       8
          state     PROCLIB.ONE  Char       8
                    PROCLIB.TWO  Char       8          Home State
```

```
                           The SAS System                              2

                         COMPARE Procedure
              Comparison of PROCLIB.ONE with PROCLIB.TWO
                           (Method=EXACT)

        Listing of Common Variables with Differing Attributes

      Variable  Dataset      Type  Length  Format  Label

      gr1       PROCLIB.ONE  Num        8  4.1
                PROCLIB.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 PROCLIB.TWO but not in PROCLIB.ONE: 1.
   Total Number of Observations Read from PROCLIB.ONE: 4.
   Total Number of Observations Read from PROCLIB.TWO: 5.

   Number of Observations with Some Compared Variables Unequal: 4.
   Number of Observations with All Compared Variables Equal: 0.
```

```
                           The SAS System                              3

                         COMPARE Procedure
              Comparison of PROCLIB.ONE with PROCLIB.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 246 shows the default output for these two data sets. Example 1 on page 256 shows the complete output for these two data sets.

Syntax: COMPARE Procedure

Restriction: You must use the VAR statement when you use the WITH statement.

Tip: Supports the Output Delivery System. See "Output Delivery System" on page 32 for details.

ODS Table Names: See: "ODS Table Names" on page 253

Reminder: You can use the LABEL, ATTRIB, FORMAT, and WHERE statements. See Chapter 3, "Statements with the Same Function in Multiple Procedures," on page 57 for details. You can also use any global statements. See "Global Statements" on page 18 for a list.

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)*;

To do this	Use this statement
Produce a separate comparison for each BY group	BY
Identify variables to use to match observations	ID
Restrict the comparison to values of specific variables	VAR
Compare variables of different names	WITH and VAR
Compare two variables in the same data set	WITH and VAR

PROC COMPARE Statement

Restriction: If you omit COMPARE=, then you must use the WITH and VAR statements.

Restriction: 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.)

Reminder: You can use data set options with the BASE= and COMPARE= options.

PROC COMPARE <*option(s)*>;

To do this	Use this option
Specify the data sets to compare	
Specify the base data set	BASE=
Specify the comparison data set	COMPARE=
Control the output data set	
Create an output data set	OUT=
Write an observation for each observation in the BASE= and COMPARE= data sets	OUTALL
Write an observation for each observation in the BASE= data set	OUTBASE
Write an observation for each observation in the COMPARE= data set	OUTCOMP
Write an observation that contains the differences for each pair of matching observations	OUTDIF
Suppress the writing of observations when all values are equal	OUTNOEQUAL
Write an observation that contains the percent differences for each pair of matching observations	OUTPERCENT
Create an output data set that contains summary statistics	OUTSTATS=
Specify how the values are compared	
Specify the criterion for judging the equality of numeric values	CRITERION=
Specify the method for judging the equality of numeric values	METHOD=
Judge missing values equal to any value	NOMISSBASE and NOMISSCOMP
Control the details in the default report	
Include the values for all matching observations	ALLOBS
Print a table of summary statistics for all pairs of matching variables	ALLSTATS and STATS
Include in the report the values and differences for all matching variables	ALLVARS
Print only a short comparison summary	BRIEFSUMMARY
Change the report for numbers between 0 and 1	FUZZ=
Restrict the number of differences to print	MAXPRINT=
Suppress the print of creation and last-modified dates	NODATE
Suppress all printed output	NOPRINT
Suppress the summary reports	NOSUMMARY
Suppress the value comparison results.	NOVALUES
Produce a complete listing of values and differences	PRINTALL

To do this	Use this option
Print the value differences by observation, not by variable	TRANSPOSE
Control the listing of variables and observations	
List all variables and observations found in only one data set	LISTALL
List all variables and observations found only in the base data set	LISTBASE
List all observations found only in the base data set	LISTBASEOBS
List all variables found only in the base data set	LISTBASEVAR
List all variables and observations found only in the comparison data set	LISTCOMP
List all observations found only in the comparison data set	LISTCOMPOBS
List all variables found only in the comparison data set	LISTCOMPVAR
List variables whose values are judged equal	LISTEQUALVAR
List all observations found in only one data set	LISTOBS
List all variables found in only one data set	LISTVAR

Options

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 also: "Table of Summary Statistics" on page 250 for information on 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.

Featured in: Example 4 on page 264

COMPARE=*SAS-data-set*
specifies the data set to use as the comparison data set.

Aliases: COMP=, C=

Default: If you omit COMPARE=, then the comparison data set is the same as the base data set, and PROC COMPARE compares variables within the data set.

Restriction: If you omit COMPARE=, then you must use the WITH 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 which 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 also: "The Equality Criterion" on page 242 for more information

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 which 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 which 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 also: "The Equality Criterion" on page 242

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

judges a missing value in the base 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 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 on the UPDATE statement, see the chapter on SAS language statements in *SAS Language Reference: Dictionary*.

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 on the UPDATE statement, see the chapter on SAS language statements in *SAS Language Reference: Dictionary*.

NOMISSING

judges missing values in both the base and comparison data sets equal to any value. By default, a missing value is only equal 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.

Featured in: Example 6 on page 270

NOSUMMARY

suppresses the data set, variable, observation, and values comparison summary reports.

Tips: NOSUMMARY produces no output if there are no differences in the matching values.

Featured in: Example 2 on page 261

NOTE

displays notes in the SAS log that describe the results of the comparison, whether or not differences were found.

NOVALUES

suppresses the report of the value comparison results.

Featured in: “Overview: COMPARE Procedure” on page 226

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 also: “Output Data Set (OUT=)” on page 254

Featured in: Example 6 on page 270

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 also: “Output Data Set (OUT=)” on page 254

OUTBASE
writes an observation to the output data set for each observation in the base data set, creating observations in which _TYPE_=BASE.

See also: “Output Data Set (OUT=)” on page 254

Featured in: Example 6 on page 270

OUTCOMP
writes an observation to the output data set for each observation in the comparison data set, creating observations in which _TYPE_=COMP.

See also: “Output Data Set (OUT=)” on page 254

Featured in: Example 6 on page 270

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 explicitly specify the OUTDIF option to create _TYPE_=DIF observations in the output data set.

See also: “Output Data Set (OUT=)” on page 254

Featured in: Example 6 on page 270

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 also: “Output Data Set (OUT=)” on page 254

Featured in: Example 6 on page 270

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 also: “Output Data Set (OUT=)” on page 254

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 also: "Output Statistics Data Set (OUTSTATS=)" on page 255 and "Table of Summary Statistics" on page 250.

Featured in: Example 7 on page 273

PRINTALL
invokes the following options: ALLVARS, ALLOBS, ALLSTATS, LISTALL, and WARNING.

Featured in: Example 1 on page 256

STATS
prints a table of summary statistics for all pairs of matching numeric variables that are judged unequal.

See also: "Table of Summary Statistics" on page 250 for information on 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 also: "Comparison Results for Observations (Using the TRANSPOSE Option)" on page 252.

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.

Main discussion: "BY" on page 58

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 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*.

Options

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 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.

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 those 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
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 also: "A Comparison with an ID Variable" on page 241

Featured in: Example 5 on page 266

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

Required Arguments

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 you specify. These variables are *ID variables*. ID variables also identify observations on the printed reports and in the output data set.

Options

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 also: "Comparing Unsorted Data" on page 238

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

- 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 first observation with the duplicate value for the comparison.

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 those named in the VAR statement.

Featured in: Example 2 on page 261, Example 3 on page 263, and Example 4 on page 264

VAR *variable(s)*;

Required Arguments

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 those appearing 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.

Featured in: Example 2 on page 261, Example 3 on page 263, and Example 4 on page 264

WITH *variable(s)*;

Required Arguments

variable(s)
one or more variables to compare with variables in the VAR statement.

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.

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

Figure 9.1 on page 241 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 9.1 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 Figure 9.2 on page 242, 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 9.2 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 on page 266 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

$$\mathrm{ABS}\,(y - x) > \gamma$$

- If METHOD=RELATIVE, then the values are unequal if

$$\mathrm{ABS}\,(y - x)\,/\,((\mathrm{ABS}\,(x) + \mathrm{ABS}\,(y))\,/2 + \delta) > \gamma$$

  The values are equal if x=y=0.
- If METHOD=PERCENT, then the values are unequal if

$$100\,(\mathrm{ABS}\,(y - x)\,/\mathrm{ABS}\,(x)) > \gamma \quad \text{for} \quad x \neq 0$$

  or

$$y \neq 0 \quad \text{for} \quad \mathrm{x} = 0$$

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 is made equal to the absolute value of γ times a very small number ϵ (epsilon) that depends on the numerical precision of the computer. This number ϵ is defined as the smallest positive floating-point value such that, using machine arithmetic, $1-\epsilon<1<1+\epsilon$. Round-off or truncation error in floating-point computations is typically a few orders of magnitude larger than ϵ. This means that CRITERION=–1000 often provides a reasonable test of the equality of computed results at the machine level of precision.

The value δ added to the denominator in the RELATIVE method is specified in parentheses after the method name: METHOD=RELATIVE(δ). If not specified in METHOD=, then δ defaults to 0. The value of δ can be used to control the behavior of the error measure when both x and y are very close to 0. If δ is not given and x and y are very close to 0, then any error produces a large relative error (in the limit, 2).

Specifying a value for δ avoids this extreme sensitivity of the RELATIVE method for small values. If you specify METHOD=RELATIVE(δ) CRITERION=γ when both x and y are much smaller than δ in absolute value, then the comparison is as if you had specified METHOD=ABSOLUTE CRITERION=$\delta\gamma$. However, when either x or y is much larger than δ in absolute value, the comparison is like METHOD=RELATIVE CRITERION=γ. For moderate values of x and y, METHOD=RELATIVE(δ) CRITERION=γ is, in effect, a compromise between METHOD=ABSOLUTE CRITERION=$\delta\ \gamma$ 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 = $(y - x)/x * 100 \text{ for } x \neq 0$

Percent Difference = missing for $x = 0$.

How PROC COMPARE Handles Variable Formats

PROC COMPARE compares unformatted values. If you have two matching variables that are formatted differently, then PROC COMPARE lists the formats of the variables.

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.

Table 9.1 on page 245 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 Table 9.1 on page 245 for the applicable conditions:

Table 9.1 Macro Return Codes

Bit	Condition	Code	Hex	Description
1	DSLABEL	1	0001X	Data set labels differ
2	DSTYPE	2	0002X	Data set types differ
3	INFORMAT	4	0004X	Variable has different informat
4	FORMAT	8	0008X	Variable has different format
5	LENGTH	16	0010X	Variable has different length
6	LABEL	32	0020X	Variable has different label
7	BASEOBS	64	0040X	Base data set has observation not in comparison
8	COMPOBS	128	0080X	Comparison data set has observation not in base
9	BASEBY	256	0100X	Base data set has BY group not in comparison
10	COMPBY	512	0200X	Comparison data set has BY group not in base
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 forth, 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 226. Because PROC COMPARE produces lengthy output, the output is presented in seven pieces.

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.

Output 9.2 shows the Data Set Summary.

Output 9.2 Partial Output

```
                            COMPARE Procedure
                  Comparison of PROCLIB.ONE with PROCLIB.TWO
                               (Method=EXACT)

                             Data Set Summary

Dataset                 Created          Modified  NVar    NObs  Label


PROCLIB.ONE  11SEP97:15:11:07  11SEP97:15:11:09     5       4  First Data Set
PROCLIB.TWO  11SEP97:15:11:10  11SEP97:15:11:10     6       5  Second Data Set
```

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.)

Output 9.3 shows the Variables Summary.

Output 9.3 Partial Output

```
                          Variables Summary

              Number of Variables in Common: 5.
              Number of Variables in PROCLIB.TWO but not in PROCLIB.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   PROCLIB.ONE  Num        8
                               PROCLIB.TWO  Char       8


            Listing of Common Variables with Differing Attributes

          Variable  Dataset      Type  Length  Format  Label

          year      PROCLIB.ONE  Char       8          Year of Birth
                    PROCLIB.TWO  Char       8
          state     PROCLIB.ONE  Char       8
                    PROCLIB.TWO  Char       8          Home State
          gr1       PROCLIB.ONE  Num        8  4.1
                    PROCLIB.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 differing 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.

Output 9.4 shows the Observation Summary.

Output 9.4 Partial Output

```
                     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 PROCLIB.TWO but not in PROCLIB.ONE: 1.
 Total Number of Observations Read from PROCLIB.ONE: 4.
 Total Number of Observations Read from PROCLIB.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 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.

Output 9.5 shows the Values Comparison Summary.

Output 9.5 Partial Output

```
                    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 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).

Output 9.6 shows the Value Comparison Results for Variables.

Output 9.6 Partial Output

```
                 Value Comparison Results for Variables

          __________________________________________________________
                     ||  Home State
                     ||  Base Value           Compare Value
                Obs  ||  state                 state
           ________  ||  ________              ________
                     ||
                  2  ||  MD                    MA
                  4  ||  MA                    MD
          __________________________________________________________


          __________________________________________________________
                     ||       Base    Compare
                Obs  ||        gr1        gr1       Diff.     % Diff
           ________  ||  _________  _________  _________  _________
                     ||
                  1  ||       85.0      84.00    -1.0000    -1.1765
                  3  ||       78.0      79.00     1.0000     1.2821
          __________________________________________________________


          __________________________________________________________
                     ||       Base    Compare
                Obs  ||        gr2        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. △

Output 9.7 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

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.

Output 9.7 is from the ALLSTATS option using the two data sets shown in "Overview":

Output 9.7 Partial Output

```
                    Value Comparison Results for Variables

           __________________________________________________________
                     ||       Base    Compare
                 Obs ||        gr1        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
           __________________________________________________________


           __________________________________________________________
                     ||       Base    Compare
                 Obs ||        gr2        gr2      Diff.     % Diff
           ________  ||  _________  _________  _________  _________
                     ||
                   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
           __________________________________________________________
```

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.

Output 9.8 shows the differences in PROCLIB.ONE and PROCLIB.TWO by observation instead of by variable.

Output 9.8 Partial Output

```
                   Comparison Results for Observations

        _OBS_1=1 _OBS_2=1:
        Variable   Base Value      Compare         Diff.        % Diff
             gr1         85.0        84.00     -1.000000     -1.176471

        _OBS_1=2 _OBS_2=2:
        Variable   Base Value      Compare
           state           MD           MA

        _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 9.2 ODS Tables Produced by the COMPARE Procedure

Table Name	Description	Generated...
CompareDatasets	Information about the data set or data sets	by default, unless NOSUMMARY or NOVALUES option is specified
CompareDetails (Comparison Results for Observations)	A listing of observations that the base data set and the compare data set do not have in common	if PRINTALL option is specified

Table Name	Description	Generated...
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 on page 270.

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 250). 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 on page 273 for an example of an OUTSTATS= data set.

Examples: COMPARE Procedure

Example 1: Producing a Complete Report of the Differences

Procedure features:

PROC COMPARE statement options

- BASE=
- PRINTALL
- COMPARE=

Data sets:

PROCLIB.ONE, PROCLIB.TWO on page 226

This example shows the most complete report that PROC COMPARE produces as procedure output.

Program

Declare the PROCLIB SAS data library.

```
libname proclib 'SAS-data-library';
```

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=40;
```

Create a complete report of the differences between two data sets. BASE= and COMPARE= specify the data sets to compare. PRINTALL prints a full report of the differences.

```
proc compare base=proclib.one compare=proclib.two printall;
   title 'Comparing Two Data Sets: Full Report';
run;
```

Output

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 250.

```
                    Comparing Two Data Sets: Full Report                    1

                             COMPARE Procedure
                  Comparison of PROCLIB.ONE with PROCLIB.TWO
                               (Method=EXACT)

                             Data Set Summary

Dataset               Created          Modified  NVar    NObs  Label

PROCLIB.ONE  11SEP97:16:19:59  11SEP97:16:20:01     5       4  First Data Set
PROCLIB.TWO  11SEP97:16:20:01  11SEP97:16:20:01     6       5  Second Data Set


                             Variables Summary

              Number of Variables in Common: 5.
              Number of Variables in PROCLIB.TWO but not in PROCLIB.ONE: 1.
              Number of Variables with Conflicting Types: 1.
              Number of Variables with Differing Attributes: 3.


          Listing of Variables in PROCLIB.TWO but not in PROCLIB.ONE

                           Variable  Type  Length

>                           major     Char       8


              Listing of Common Variables with Conflicting Types

                     Variable  Dataset      Type  Length

                     student   PROCLIB.ONE  Num        8
                               PROCLIB.TWO  Char       8
```

```
                  Comparing Two Data Sets: Full Report                     2

                           COMPARE Procedure
               Comparison of PROCLIB.ONE with PROCLIB.TWO
                             (Method=EXACT)

         Listing of Common Variables with Differing Attributes

        Variable  Dataset      Type  Length  Format  Label

        year      PROCLIB.ONE  Char       8          Year of Birth
                  PROCLIB.TWO  Char       8
        state     PROCLIB.ONE  Char       8
                  PROCLIB.TWO  Char       8          Home State
        gr1       PROCLIB.ONE  Num        8  4.1
                  PROCLIB.TWO  Num        8  5.2


                  Comparison Results for Observations

>       Observation 5 in PROCLIB.TWO not found in PROCLIB.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 PROCLIB.TWO but not in PROCLIB.ONE: 1.
     Total Number of Observations Read from PROCLIB.ONE: 4.
     Total Number of Observations Read from PROCLIB.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                     3

                           COMPARE Procedure
               Comparison of PROCLIB.ONE with PROCLIB.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                  4

                            COMPARE Procedure
                 Comparison of PROCLIB.ONE with PROCLIB.TWO
                              (Method=EXACT)

                  Value Comparison Results for Variables

            __________________________________________________________
                      ||  Year of Birth
                      ||  Base Value            Compare Value
                 Obs  ||  year                   year
             ________ ||  ________               ________
                      ||
                   1  ||  1970                   1970
                   2  ||  1971                   1971
                   3  ||  1969                   1969
                   4  ||  1970                   1970
            __________________________________________________________


            __________________________________________________________
                      ||  Home State
                      ||  Base Value            Compare Value
                 Obs  ||  state                  state
             ________ ||  ________               ________
                      ||
                   1  ||  NC                     NC
                   2  ||  MD                     MA
                   3  ||  PA                     PA
                   4  ||  MA                     MD
            __________________________________________________________
```

```
                   Comparing Two Data Sets: Full Report                  5

                            COMPARE Procedure
                 Comparison of PROCLIB.ONE with PROCLIB.TWO
                              (Method=EXACT)

                  Value Comparison Results for Variables

            __________________________________________________________
                      ||       Base    Compare
                 Obs  ||        gr1        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                     6

                             COMPARE Procedure
                  Comparison of PROCLIB.ONE with PROCLIB.TWO
                              (Method=EXACT)

                    Value Comparison Results for Variables

          __________________________________________________________
                     ||       Base    Compare
                 Obs ||        gr2        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

Procedure features:

PROC COMPARE statement option

NOSUMMARY

VAR statement

WITH statement

Data sets:

PROCLIB.ONE, PROCLIB.TWO on page 226.

This example compares a variable from the base data set with a variable in the comparison data set. All summary reports are suppressed.

Program

Declare the PROCLIB SAS data library.

```
libname proclib 'SAS-data-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

```
                 Comparison of Variables in Different Data Sets                1

                              COMPARE Procedure
                  Comparison of PROCLIB.ONE with PROCLIB.TWO
                                (Method=EXACT)

NOTE: Data set PROCLIB.TWO contains 1 observations not in PROCLIB.ONE.
NOTE: Values of the following 1 variables compare unequal: gr1^=gr2


                     Value Comparison Results for Variables

           __________________________________________________________
                      ||       Base    Compare
                  Obs ||        gr1        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

Procedure features:
VAR statement
WITH statement

Data sets:
PROCLIB.ONE, PROCLIB.TWO on page 226.

This example compares one variable from the base data set with two variables in the comparison data set.

Program

Declare the PROCLIB SAS data library.

```
libname proclib 'SAS-data-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

The Value Comparison Results section shows the result of the comparison.

```
                 Comparison of One Variable with Two Variables                1

                              COMPARE Procedure
                   Comparison of PROCLIB.ONE with PROCLIB.TWO
                                (Method=EXACT)

NOTE: Data set PROCLIB.TWO contains 1 observations not in PROCLIB.ONE.
NOTE: Values of the following 2 variables compare unequal: gr1^=gr1 gr1^=gr2


                     Value Comparison Results for Variables

             __________________________________________________________
                       ||       Base    Compare
                   Obs ||        gr1        gr1       Diff.     % Diff
              ________ ||  _________  _________  _________  _________
                       ||
                     1 ||       85.0      84.00    -1.0000    -1.1765
                     3 ||       78.0      79.00     1.0000     1.2821
             __________________________________________________________


             __________________________________________________________
                       ||       Base    Compare
                   Obs ||        gr1        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

Procedure features:

PROC COMPARE statement options

- ALLSTATS
- BRIEFSUMMARY

VAR statement

WITH statement

Data set:

PROCLIB.ONE on page 226.

This example shows that PROC COMPARE can compare two variables that are in the same data set.

Program

Declare the PROCLIB SAS data library.

```
libname proclib 'SAS-data-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

```
                 Comparison of Variables in the Same Data Set                1

                             COMPARE Procedure
                   Comparisons of variables in PROCLIB.ONE
                               (Method=EXACT)

NOTE: Values of the following 1 variables compare unequal: gr1^=gr2


                  Value Comparison Results for Variables

           __________________________________________________________
                     ||       Base    Compare
                Obs  ||        gr1        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

Procedure features:
ID statement

In this example, PROC COMPARE compares only the observations that have matching values for the ID variable.

Program

Declare the PROCLIB SAS data library.

```
libname proclib 'SAS-data-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. A DATA step on page 1391 creates PROCLIB.EMP95. A DATA step on page 1392 creates PROCLIB.EMP96.

```
data proclib.emp95;
   input #1 idnum $4. @6 name $15.
         #2 address $42.
         #3 salary 6.;
   datalines;
2388 James Schmidt
100 Apt. C Blount St. SW Raleigh NC 27693
92100
2457 Fred Williams
99 West Lane  Garner NC 27509
33190
... more data lines...
3888 Kim Siu
5662 Magnolia Blvd Southeast Cary NC 27513
77558
;

data proclib.emp96;
   input #1 idnum $4. @6 name $15.
         #2 address $42.
         #3 salary 6.;
   datalines;
2388 James Schmidt
100 Apt. C Blount St. SW Raleigh NC 27693
92100
2457 Fred Williams
99 West Lane  Garner NC 27509
33190
...more data lines...
6544 Roger Monday
3004 Crepe Myrtle Court Raleigh NC 27604
47007
;
```

Sort the data sets by the ID variable. Both data sets must be sorted by the variable that will be used as the ID variable in the PROC COMPARE step. OUT= specifies the location of the sorted data.

```
proc sort data=proclib.emp95 out=emp95_byidnum;

 by idnum;
run;

proc sort data=proclib.emp96 out=emp96_byidnum;
   by idnum;
run;
```

Create a summary report that compares observations with matching values for the ID variable. The ID statement specifies IDNUM as the ID variable.

```
proc compare base=emp95_byidnum compare=emp96_byidnum;
   id idnum;
   title 'Comparing Observations that Have Matching IDNUMs';
run;
```

Output

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 on page 270.

```
                Comparing Observations that Have Matching IDNUMs              1

                               COMPARE Procedure
            Comparison of WORK.EMP95_BYIDNUM with WORK.EMP96_BYIDNUM
                                 (Method=EXACT)

                               Data Set Summary

       Dataset                      Created          Modified  NVar    NObs

       WORK.EMP95_BYIDNUM  13MAY98:16:03:36  13MAY98:16:03:36     4      10
       WORK.EMP96_BYIDNUM  13MAY98:16:03:36  13MAY98:16:03:36     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              2

                               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               3

                          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=)

Procedure features:

PROC COMPARE statement options:

- NOPRINT
- OUT=
- OUTBASE
- OUTBASE
- OUTCOMP
- OUTDIF
- OUTNOEQUAL

Other features: PRINT procedure

Data sets: PROCLIB.EMP95 and PROCLIB.EMP96 on page 267

This example creates and prints an output data set that shows the differences between matching observations.

In Example 5 on page 266, 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

Declare the PROCLIB SAS data library.

```
libname proclib 'SAS-data-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 Chapter 34, "The PRINT Procedure," on page 703 for more information on this technique.

```
proc print data=result noobs;
   by idnum;
   id idnum;
   title 'The Output Data Set RESULT';
run;
```

Output

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                                      1

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

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

3278    COMPARE      6    Mary Cravens      211 N. Cypress St. Cary NC 27512               35362

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

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

6544    COMPARE      9    Roger Monday      3004 Crepe Myrtle Court Raleigh NC 27604       47007

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    ...............   ...XXXXXXXXXXXXXX.XXXXX.XXXXXXXXXXX.......      1700
```

Example 7: Creating an Output Data Set of Statistics (OUTSTATS=)

Procedure features:

PROC COMPARE statement options:

NOPRINT
OUTSTATS=

Data sets: PROCLIB.EMP95, PROCLIB.EMP96 on page 267

This example creates an output data set that contains summary statistics for the numeric variables that are compared.

Program

Declare the PROCLIB SAS data library.

```
libname proclib 'SAS-data-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;
```

Output

The variables are described in "Output Statistics Data Set (OUTSTATS=)" on page 255.

```
                              The DIFFSTAT Data Set                              1

      _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
10

The CONTENTS Procedure

Overview: CONTENTS Procedure

The CONTENTS procedure shows the contents of a SAS data set and prints the directory of the SAS data 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 either WORK or USER. For the CONTENTS statement, the default is the libref of the procedure input library.
- □ PROC CONTENTS can read sequential files. The CONTENTS statement cannot.

Syntax: CONTENTS Procedure

Tip: Supports the Output Delivery System. See "Output Delivery System" on page 32 for details.

ODS Table Names: See: "ODS Table Names" on page 369

Reminder: You can use the ATTRIB, FORMAT, and LABEL statements. See Chapter 3, "Statements with the Same Function in Multiple Procedures," on page 57 for details. You can also use any global statements. See "Global Statements" on page 18 for a list.

Reminder: You can use data set options with the DATA= and OUT= options. See "Data Set Options" on page 18 for a list.

Reminder: Complete documentation for the CONTENTS statement and the CONTENTS procedure is in "CONTENTS Statement" on page 323.

See: CONTENTS Procedure in the documentation for your operating environment.

PROC CONTENTS <*option(s)*>;

To do this	Use this option
Print centiles information for indexed variables	CENTILES
Specify the input data set	DATA=
Include information in the output about the number of observations, number of variables, and data set labels	DETAILS|NODETAILS
Print a list of the SAS files in the SAS data library	DIRECTORY
Print the length of a variable's informat or format	FMTLEN
Restrict processing to one or more types of SAS file	MEMTYPE=
Suppress the printing of individual files	NODS
Suppress the printing of the output	NOPRINT
Print a list of variables in alphabetical order even if they include mixed case names	ORDER=IGNORECASE
Specify the output data set	OUT=
Specify an output data set that contains information about constraints	OUT2=
Print abbreviated output	SHORT
Print a list of the variables by their logical position in the data set	VARNUM

CHAPTER 11

The COPY Procedure

Overview: COPY Procedure

The COPY procedure copies one or more SAS files from a SAS data 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 data library from a previous release to the most recent release. For migration, PROC MIGRATE offers benefits that PROC COPY does not. For documentation on PROC MIGRATE, see the Migration Community at **`http://support.sas.com/rnd/migration`**. △

Syntax: COPY Procedure

Reminder: See Chapter 3, "Statements with the Same Function in Multiple Procedures," on page 57 for details. You can also use any global statements. See "Global Statements" on page 18 for a list.

Reminder: Complete documentation for the COPY statement and the COPY procedure is in "COPY Statement" on page 327.

Restriction: PROC COPY ignores explicit concatenations with catalogs. Use PROC CATALOG COPY to copy concatenated catalogs.

PROC COPY OUT=*libref-1* IN=*libref-2*
<CLONE|NOCLONE>
<CONSTRAINT=YES|NO>
<DATECOPY>
<INDEX=YES|NO>
<MEMTYPE=(*mtype(s)*)>
<MOVE <ALTER=*alter-password*>>;

EXCLUDE *SAS-file(s)* </ MEMTYPE=*mtype*>;

SELECT *SAS-file(s)* </ <MEMTYPE=*mtype*>
<ALTER=*alter-password*>>;

Concepts: COPY Procedure

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 data library.

For an example, see Example 1 on page 279.

For details on transporting files, see *Moving and Accessing SAS Files across Operating Environments*.

The CPORT and CIMPORT procedures also provide a way to transport SAS files. For information, see Chapter 8, “The CIMPORT Procedure,” on page 215 and Chapter 13, “The CPORT Procedure,” on page 285.

Example: COPY Procedure

Example 1: Copying SAS Data Sets between Hosts

Features:
PROC COPY statement options:

IN=
MEMTYPE=
OUT=

Other features: XPORT engine

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

Assign library references. Assign a libref, such as SOURCE, to the SAS data 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-data-library-on-sending-host';
libname xptout xport 'filename-on-sending-host';
```

Copy the SAS data sets to the transport file. Use PROC COPY to copy the SAS data sets from the IN= library to the transport file. MEMTYPE=DATA specifies that only SAS data sets are copied. SELECT selects the data sets that you want to copy.

```
proc copy in=source out=xptout memtype=data;
   select bonus budget salary;
run;
```

SAS Log

SAS Log on Sending Host

```
1    libname source 'SAS-data-library-on-sending-host ';
NOTE: Libref SOURCE was successfully assigned as follows:
      Engine:        V9
      Physical Name: SAS-data-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 and 3 variables.
```

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 NT 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;
```

SAS Log on Receiving Host

```
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.
```

CHAPTER 12

The CORR Procedure

Information about the CORR Procedure

See: The documentation for the CORR procedure has moved to Volume 3 of this book.

CHAPTER

13

The CPORT Procedure

Overview: CPORT Procedure

What Does the CPORT Procedure Do?

The CPORT procedure writes SAS data sets, SAS catalogs, or SAS data 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 data 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 data 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 data library. PROC CIMPORT restores (*imports*) the transport file to its original form as a SAS catalog, SAS data set, or SAS data library.

Only PROC CIMPORT can read the transport files that PROC CPORT creates. For information on the transport files that the transport engine creates, see the section on SAS files in *SAS Language Reference: Concepts*.

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. In such cases, PROC CIMPORT automatically converts the contents of the transport file as it imports it.

PROC CPORT produces no output (other than the transport files), but it does write notes to the SAS log.

General File Transport Process

To export and import files, follow these steps:

1 Use PROC CPORT to export the SAS files that you want to transport.

2 If you are changing operating environments, move the transport file to the new machine by using either communications software or a magnetic medium.

  Note: If you use communications software to move the transport file, be sure that it treats the transport file as a *binary* file and that it modifies neither the attributes nor the contents of the file. △

3 Use PROC CIMPORT to translate the transport file into the format appropriate for the new operating environment or release.

Syntax: CPORT Procedure

See: CPORT Procedure in the documentation for your operating environment.

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)>*;

PROC CPORT Statement

PROC CPORT *source-type=libref* | *<libref.>member-name<option(s)>*;

To do this	Use this option
Identify the transport file	
Specify the transport file to write to	FILE=

To do this	Use this option
Direct the output from PROC CPORT to a tape	TAPE
Select files to export	
Export copies of all data sets or catalog entries that have a modification date equal to or later than the date you specify	AFTER=
Exclude specified entry types from the transport file	EET=
Include specified entry types in the transport file	ET=
Specify whether to export all generations of a data set	GENERATION=
Specify that only data sets, only catalogs, or both, be moved when a library is exported	MEMTYPE=
Control the contents of the transport file	
Suppress the conversion of displayed character data to transport format	ASIS
Control the exportation of integrity constraints	CONSTRAINT
Copy the created and modified date and time to the transport file	DATECOPY
Control the exportation of indexes with indexed SAS data sets	INDEX
Suppress the compression of binary zeros and blanks in the transport file	NOCOMPRESS
Write all alphabetic characters to the transport file in uppercase	OUTTYPE= UPCASE
Translate specified characters from one ASCII or EBCDIC value to another	TRANSLATE
Export SAS/AF PROGRAM and SCL entries without edit capability when you import them	NOEDIT
Specify that exported catalog entries contain compiled SCL code, but not the source code	NOSRC
Specify a libref associated with a SAS data library	OUTLIB=

Required Arguments

source-type=libref* | <*libref.*>*member-name
identifies the type of file to export and specifies the catalog, SAS data set, or SAS data library to export.

source-type
identifies the file(s) to export as a single catalog, as a single SAS data set, or as the members of a SAS data library. The *source-type* 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 data 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.

Options

AFTER=*date*
exports copies of all data sets or catalog entries that have a modification date later than or equal to the date 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.

Featured in: Example 5 on page 298.

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.

Interaction: The ASIS option invokes the NOCOMPRESS option.

Interaction: 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 CONTRAINT=NO, indexes created without integrity constraints are ported, but neither integrity constraints nor any indexes created with integrity constraints are ported. For more information on integrity constraints, see the section on SAS files in *SAS Language Reference: Concepts*.

Alias: CON=

Default: YES

Interaction: You cannot specify both CONSTRAINT= and INDEX= in the same PROC CPORT step.

Interaction: 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 on the MODIFY statement"MODIFY Statement" on page 348 in a PROC DATASETS step.

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. △

Featured in: 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 on generation data sets, see *SAS Language Reference: Concepts*.

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. △

Alias: GEN=

Default: YES for libraries; NO for single data sets

INDEX=YES | NO
specifies whether to export indexes with indexed SAS data sets.

Default: YES

Interaction: You cannot specify both INDEX= and CONSTRAINT= in the same PROC CPORT step.

Interaction: 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

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.

Default: If the INTYPE= option is not used, the DBCS type defaults to the value of the SAS system option DBCSTYPE=.

Interaction: Use the INTYPE= option in conjunction with the OUTTYPE= option to change from one type of DBCS data to another.

Interaction: The INTYPE= option invokes the NOCOMRPESS option.

Interaction: 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

Featured in: Example 1 on page 295.

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.

Note: The NOEDIT option affects only SAS/AF PROGRAM and SCL entries. It does not affect FSEDIT SCREEN or FSVIEW FORMULA entries. △

Alias: NEDIT

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 data library. If you specify the OUTLIB= option, PROC CIMPORT is invoked automatically to re-create the input data 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 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.

Tip: There is no limit to the number of EXCLUDE statements you can use in one invocation of PROC CPORT.

Interaction: You can use either EXCLUDE statements or SELECT statements in a PROC CPORT step, but not both.

EXCLUDE *SAS file(s)* | *catalog entry(s)*</ MEMTYPE=*mtype*></ ENTRYTYPE=*entry-type*>;

Required Arguments

SAS file(s) | catalog entry(s)
specifies either the name(s) of one or more SAS files or the names of one or more catalog entries to be excluded from the transport file. Specify SAS filenames when you export a SAS data 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 24.

Options

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.

Restriction: ENTRYTYPE= is valid only when you export an individual SAS catalog.

Alias: ETYPE=, ET=

MEMTYPE=*mtype*
specifies a single member type for the SAS file(s) 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 file name 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:

Restriction: MEMTYPE= is valid only when you export a SAS data library.

Restriction: 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.

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.

SELECT Statement

Includes specified files or entries in the transport file.

Tip: There is no limit to the number of SELECT statements you can use in one invocation of PROC CPORT.

Interaction: You can use either EXCLUDE statements or SELECT statements in a PROC CPORT step, but not both.

Featured in: Example 2 on page 296

SELECT *SAS file(s)* | *catalog entry(s)*</ MEMTYPE=*mtype*> </ ENTRYTYPE=*entry-type*> ;

Required Arguments

SAS file(s) | catalog entry(s)
specifies either the name(s) of one or more SAS files or the names of one or more catalog entries to be included in the transport file. Specify SAS filenames when you export a SAS data 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 24.

Options

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.

Restriction: ENTRYTYPE= is valid only when you export an individual SAS catalog.

Alias: ETYPE=, ET=

MEMTYPE=*mtype*
specifies a single member type for the SAS file(s) 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.

Restriction: MEMTYPE= is valid only when you export a SAS data library.

Restriction: 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.

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.

TRANTAB Statement

Specifies translation tables for characters in catalog entries you export.

Tip: You can specify only one table for each TRANTAB statement, but there is no limit to the number of TRANTAB statements you can use in one invocation of PROC CPORT.

Featured in: Example 4 on page 297.

See: The TRANTAB Statement for the CPORT Procedure and the UPLOAD and DOWNLOAD Procedures in *SAS National Language Support (NLS): User's Guide*

TRANTAB NAME=*translation-table-name*
<*option(s)*>;

Concepts: CPORT Procedure

For password-protected data sets, the password(s) are applied to the destination data set when it is imported. If the data set is transported as part of a library, it is not necessary to supply the password. If the data set is transported singly, you must supply the read password. If you omit the password in the PROC CPORT step, SAS prompts you for the password. If the target SAS engine does not support passwords, then the import will fail. For example, the following SAS code transports a password-protected data set called WORK.ONE:

```
proc cport data=one(read=hithere) file='bin';
```

Results: CPORT Procedure

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:

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.

Examples: CPORT Procedure

Example 1: Exporting Multiple Catalogs

Procedure features:

PROC CPORT statement options:
- FILE=
- MEMTYPE=

This example shows how to use PROC CPORT to export entries from all of the SAS catalogs in the SAS data library you specify.

Program

Specify the library reference for the SAS data 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 data 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

Procedure features:

PROC CPORT statement options:

FILE=

SELECT statement

This example shows how to use PROC CPORT to export individual catalog entries, rather than all of the entries in a catalog.

Program

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

Procedure features:

PROC CPORT statement option:

FILE=

This example shows how to use PROC CPORT to export a single SAS data set.

Program

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

Procedure features:

PROC CPORT statement option:

FILE=

TRANTAB statement option:

TYPE=

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

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

Procedure features:

PROC CPORT statement options:

AFTER=
FILE=

This example shows how to use PROC CPORT to transport only the catalog entries with modification dates equal to or later than the date you specify in the AFTER= option.

Program

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

14

The CV2VIEW Procedure

Information about the CV2VIEW Procedure

See: For complete documentation of the CV2VIEW procedure, see *SAS/ACCESS for Relational Databases: Reference*.

CHAPTER 15

The 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

- 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, and so on
- 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

1 copies all data sets from the CONTROL library to the HEALTH library

2 lists the contents of the HEALTH library

3 deletes the SYNDROME data set from the HEALTH library

4 changes the name of the PRENAT data set to INFANT.

The SAS log is shown in Output 15.1.

```
libname control 'SAS-data-library-1';
libname health 'SAS-data-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;
```

Output 15.1 Log from PROC DATASETS

```
59   proc datasets library=health memtype=data details;
                                                     Directory

                                   Libref         HEALTH
                                   Engine         V9
                                   Physical Name  external-file
                                   File Name      external-file


                           Member  Obs, Entries                                  File
               #  Name      Type     or Indexes   Vars  Label                       Size  Last Modified

               1  ALL       DATA         23        17                            13312   29JAN2002:08:06:46
               2  BODYFAT   DATA          1         2                             5120   29JAN2002:08:06:46
               3  CONFOUND  DATA          8         4                             5120   29JAN2002:08:06:46
               4  CORONARY  DATA         39         4                             5120   29JAN2002:08:06:46
               5  DRUG1     DATA          6         2  JAN95 Data                 5120   29JAN2002:08:06:46
               6  DRUG2     DATA         13         2  MAY95 Data                 5120   29JAN2002:08:06:46
               7  DRUG3     DATA         11         2  JUL95 Data                 5120   29JAN2002:08:06:46
               8  DRUG4     DATA          7         2  JAN92 Data                 5120   29JAN2002:08:06:46
               9  DRUG5     DATA          1         2  JUL92 Data                 5120   29JAN2002:08:06:46
              10  GROUP     DATA        148        11                            25600   29JAN2002:08:06:46
              11  MLSCL     DATA         32         4  Multiple Sclerosis Data    5120   29JAN2002:08:06:46
              12  NAMES     DATA          7         4                             5120   29JAN2002:08:06:46
              13  OXYGEN    DATA         31         7                             9216   29JAN2002:08:06:46
              14  PERSONL   DATA        148        11                            25600   29JAN2002:08:06:46
              15  PHARM     DATA          6         3  Sugar Study                5120   29JAN2002:08:06:46
              16  POINTS    DATA          6         6                             5120   29JAN2002:08:06:46
              17  PRENAT    DATA        149         6                            17408   29JAN2002:08:06:46
              18  RESULTS   DATA         10         5                             5120   29JAN2002:08:06:46
              19  SLEEP     DATA        108         6                             9216   29JAN2002:08:06:46
              20  SYNDROME  DATA         46         8                             9216   29JAN2002:08:06:46
              21  TENSION   DATA          4         3                             5120   29JAN2002:08:06:46
              22  TEST2     DATA         15         5                             5120   29JAN2002:08:06:46
              23  TRAIN     DATA          7         2                             5120   29JAN2002:08:06:47
              24  VISION    DATA         16         3                             5120   29JAN2002:08:06:47
              25  WEIGHT    DATA         83        13  California Results        13312   29JAN2002:08:06:47
              26  WGHT      DATA         83        13  California Results        13312   29JAN2002:08:06:47
60      delete syndrome;
61      change prenat=infant;
62   run;
NOTE: Deleting HEALTH.SYNDROME (memtype=DATA).
NOTE: Changing the name HEALTH.PRENAT to HEALTH.INFANT (memtype=DATA).
63   quit;
```

Notes

- Although the DATASETS procedure can perform some operations on catalogs, generally the CATALOG procedure is the best utility to use for managing catalogs. For documentation of PROC CATALOG, see "Overview: CATALOG Procedure" on page 153.
- 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 *SAS Language Reference: Concepts*.
- PROC DATASETS cannot work with sequential data libraries.

Syntax: DATASETS Procedure

Tip: Supports RUN-group processing.

Tip: Supports the Output Delivery System. See "Output Delivery System" on page 32 for details.

ODS Table Names: See: "ODS Table Names" on page 369

Reminder: See Chapter 3, "Statements with the Same Function in Multiple Procedures," on page 57 for details. You can also use any global statements. See "Global Statements" on page 18 for a list.

See: DATASETS Procedure in the documentation for your operating environment.

PROC DATASETS <*option(s)*>;

AGE *current-name related-SAS-file(s)*
</ <ALTER=*alter-password*>
<MEMTYPE=*mtype*>>;

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

AUDIT *SAS-file* <(*SAS-password*)>;
INITIATE
<AUDIT_ALL=NO | YES>;
<LOG <ADMIN_IMAGE=YES | NO>
<BEFORE_IMAGE=YES | NO>
<DATA_IMAGE=YES | NO>
<ERROR_IMAGE=YES | NO>>;
<USER_VAR *variable-1* <... *variable-n*>>;

AUDIT *SAS-file* <(<*SAS-password*> <GENNUM= *integer*>)>;
SUSPEND | RESUME | TERMINATE;

CHANGE *old-name-1=new-name-1*
<*...old-name-n=new-name-n* >
</ <ALTER=*alter-password*>
<GENNUM=ALL | *integer*>
<MEMTYPE=*mtype*>>;

CONTENTS<*option(s)*>;

COPY OUT=*libref-1*
<CLONE | NOCLONE>
<CONSTRAINT=YES | NO>
<DATECOPY>
<FORCE>
<IN=*libref-2*>
<INDEX=YES | NO>
<MEMTYPE=(*mtype(s)*)>
<MOVE <ALTER=*alter-password*>>;
EXCLUDE *SAS-file(s)* < / MEMTYPE=*mtype*>;
SELECT *SAS-file(s)*
</ <ALTER=*alter-password*>
<MEMTYPE= *mtype*>>;

DELETE *SAS-file(s)*
</ <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(s)*)>
</ <CORRECTENCODING=*encoding-value*>
<DTC=*SAS-date-time*>
<GENNUM=*integer*>
<MEMTYPE=*mtype*>>;
FORMAT *variable-list-1 <format-1>*
<*...variable-list-n <format-n>*>;
IC CREATE <*constraint-name*=> *constraint*
<MESSAGE='*message-string*' <MSGTYPE=USER>>;
IC DELETE *constraint-name(s)* | _ALL_;
IC REACTIVATE *foreign-key-name* REFERENCES *libref*;
INDEX CENTILES *index(s)*
</ <REFRESH>
<UPDATECENTILES= ALWAYS | NEVER | *integer*>>;
INDEX CREATE *index-specification(s)*
</ <NOMISS>
<UNIQUE>
<UPDATECENTILES=ALWAYS | NEVER | *integer*>>;
INDEX DELETE *index(s)* | _ALL_;
INFORMAT *variable-list-1 <informat-1>*
<*...variable-list-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*>;

REPAIR *SAS-file(s)*
</ <ALTER=*alter-password*>
<GENNUM=*integer*>
<MEMTYPE=*mtype*>>;

SAVE *SAS-file(s)* </ MEMTYPE=*mtype*>;

PROC DATASETS Statement

PROC DATASETS <*option(s)*>;

To do this	Use this option
Specify the procedure input library	LIBRARY=
Provide alter access to any alter-protected SAS file in the SAS data library	ALTER=

To do this	Use this option
Include information in the log about the number of observations, number of variables, number of indexes, and data set labels	DETAILS|NODETAILS
Force a RUN group to execute even when there are errors	FORCE
Force an append operation	FORCE
Restrict processing for generation data sets	GENNUM=
Delete SAS files	KILL
Restrict processing to a certain type of SAS file	MEMTYPE=
Suppress the printing of the directory	NOLIST
Suppress error processing	NOWARN
Provide read, write, or alter access	PW=
Provide read access	READ=

Options

ALTER=*alter-password*
provides the alter password for any alter-protected SAS files in the SAS data library.

See also: "Using Passwords with the DATASETS Procedure" on page 359

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 will show 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 data 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 requestor window prompts you for each read password that you do not specify in the PROC DATASETS statement. Therefore, you may want to assign the same read password to all SAS files in the same SAS data library.

Featured in: Example 1 on page 376

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 357 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. Without the FORCE option, the procedure issues an error message and stops processing if you try to perform an append operation with two SAS data sets whose variables are not exactly the same. Refer to "APPEND Statement" on page 313 for more information on 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 also: "Restricting Processing for Generation Data Sets" on page 362

See also: "Understanding Generation Data Sets" in *SAS Language Reference: Concepts*

KILL

deletes *all* SAS files in the SAS data library that are available for processing. The MEMTYPE= option subsets the member types that the statement deletes.

CAUTION:

The KILL option deletes the SAS files immediately after you submit the statement. △

LIBRARY=*libref*
names the library that the procedure processes. This library is the *procedure input library*.

Aliases: DDNAME=, DD=, LIB=

Default: WORK or USER. See "Temporary and Permanent SAS Data Sets" on page 16 for more information on the WORK and USER libraries.

Restriction: 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.

Featured in: Example 1 on page 376

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;
```

Aliases: MTYPE=, MT=

Default: ALL

See also: "Restricting Member Types for Processing" on page 360

NODETAILS
See the description of DETAILS on page 309.

NOLIST
suppresses the printing of the directory of the SAS files in the SAS log.

Featured in: Example 3 on page 381

Note: If you specify the ODS RTF destination, PROC DATASETS output will go to both the SAS log and the ODS output area. The NOLIST option will suppress 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. △

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 data library. PW= can act as an alias for READ=, WRITE=, or ALTER=.

See also: "Using Passwords with the DATASETS Procedure" on page 359

READ=*read-password*
provides the read-password for any read-protected SAS files in the SAS data library.

See also: "Using Passwords with the DATASETS Procedure" on page 359

AGE Statement

Renames a group of related SAS files in a library.

Featured in: Example 6 on page 388

AGE *current-name related-SAS-file(s)*
</ <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(s)*.

related-SAS-file(s)
is one or more SAS files in the SAS data library.

Options

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 also: "Using Passwords with the DATASETS Procedure" on page 359

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.

Aliases: MTYPE=, MT=

Default: If you do not specify MEMTYPE= in the PROC DATASETS statement, the default is DATA.

See also: "Restricting Member Types for Processing" on page 360

Details

- The AGE statement renames *current-name* to the name of the first name in *related-SAS-file(s)*, renames the first name in *related-SAS-file(s)* to the second name in *related-SAS-file(s)*, and so on until it changes the name of the next-to-last SAS file in *related-SAS-file(s)* to the last name in *related-SAS-file(s)*. The AGE statement then deletes the last file in *related-SAS-file(s)*.
- If the first SAS file named in the AGE statement does not exist in the SAS data 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-file(s)*. To override this behavior, use the NOWARN option in the PROC DATASETS statement.

If one of the *related-SAS-file(s)* 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 3 historical versions and the generation group for data set B has 2 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.

Reminder: You can use data set options with the BASE= and DATA= options. See "Data Set Options" on page 18 for a list. You can also use any global statements as well. See "Global Statements" on page 18.

Requirement: The BASE= data set must be a member of a SAS library that supports update processing.

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 happens, no other user can update the file while the data set is processed.

Tip: If a failure occurs during processing, the data set is marked as damaged and is reset to its pre-append condition at the next REPAIR statement. If the data set has an index, the index is not updated with each observation but is updated once at the end. (This is Version 7 and later behavior, as long as APPENDVER=V6 is not set.)

Featured in: Example 5 on page 386

APPEND BASE=<*libref.*>*SAS-data-set*
<APPENDVER=V6>

<DATA=<*libref.*>*SAS-data-set*>
<FORCE>;

Required Arguments

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.

The BASE= data set is the current SAS data set after all append operations regardless of whether you are creating a new data set or appending to an existing data set.

Alias: OUT=

Featured in: Example 5 on page 386

Options

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 also: "Appending to an Indexed Data Set — Fast-Append Method" on page 316

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 data library, but 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 data library

See also: "Appending with Generation Groups" on page 318

Featured in: Example 5 on page 386

FORCE
forces the APPEND statement to concatenate data sets when the DATA= data set contains variables that either

- □ 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.

See also: "Appending to Data Sets with Different Variables" on page 317

See also: "Appending to Data Sets That Contain Variables with Different Attributes" on page 317

Featured in: Example 5 on page 386

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;
```

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 on the BASE= data set, it will take effect only if the WHEREUP= option is set to YES. △

CAUTION:

For the non-existent BASE= data set: If there is a WHERE statement on 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

- □ 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 may insert the observations into the middle of the BASE= data set to make use of available space.

For information on the COMPRESS= and REUSE= data set and system options, see *SAS Language Reference: Dictionary*.

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 in sequential order, which 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 code 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 displays if the fast-append method is in use or a message or messages display 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. This means that after observations are appended, some of them may 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 occurs

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 will be deleted, because the internal sort may 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 recreating it after the append could improve performance. The current method may 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.

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.

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 formats in the DATA= data set are different from those in the BASE= data set, then the 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 formats in the BASE= data set. The result could be data that appears to be incorrect. A warning message is displayed in the SAS log. The following example illustrates appending data by using different 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.

Output 15.2 Warning Message in 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 these 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.

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 may not be successful; it is possible that not all, perhaps none, of the observations will be added to the BASE= data set. In addition, the BASE= data set may 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 the PROC DATASETS “AUDIT Statement” on page 319.

AUDIT Statement

Initiates and controls event logging to an audit file as well as suspends, resumes, or terminates event logging in an audit file.

See also: “Understanding an Audit Trail” in *SAS Language Reference: Concepts*

Tip: 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.

AUDIT *SAS-file* <(*SAS-password*)>;

INITIATE
<AUDIT_ALL=NO | YES>;
<LOG <ADMIN_IMAGE=YES | NO>
<BEFORE_IMAGE=YES | NO>
<DATA_IMAGE=YES | NO>
<ERROR_IMAGE=YES | NO>>;
<USER_VAR *variable-1* <... *variable-n*>>;

AUDIT *SAS-file* <(<*SAS-password*> <GENNUM= *integer*>)>;
SUSPEND | RESUME | TERMINATE;

Required Arguments and Statements

SAS-file
specifies the SAS data file in the procedure input library that you want to audit.

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.

Options

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 *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. The parentheses are required.

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

LOG
specifies the audit settings:

ADMIN_IMAGE=YES | NO
controls the logging of administrative events to the audit file (that is, the SUSPEND and RESUME actions).

BEFORE_IMAGE=YES | NO
controls the storage of before-update record images.

DATA_IMAGE=YES | NO
controls the storage of added, deleted, and after-update record images.

ERROR_IMAGE=YES | NO
controls the storage of unsuccessful after-update record images.

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;
```

USER_VAR *variable-1 < ... variable-n>*
defines optional variables to be logged in the audit file with each update to an observation. The syntax for defining variables is

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.

$
indicates that the variable is a character variable.

length
specifies the length of the variable. If a length is not specified, the default is 8.

LABEL='*variable-label*'
specifies a label for the variable.
You can define attributes such as format and informat for the user variables in the data file by using the PROC DATASETS MODIFY statement.

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.

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=MyPassword);
   initiate;
run;
```

The following example creates the same audit file but stores only error record images:

```
proc datasets library=MyLib;
   audit MyFile (alter=MyPassword);
   initiate
      log data_image=NO before_image=NO;
run;
```

CHANGE Statement

Renames one or more SAS files in the same SAS data library.

Featured in: Example 1 on page 376

CHANGE *old-name-1=new-name-1*
<*…old-name-n=new-name-n* >
</ <ALTER=*alter-password*>
<GENNUM=ALL|*integer*>
<MEMTYPE=*mtype*>>;

Required Arguments

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.

Featured in: Example 1 on page 376

Options

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 also: "Using Passwords with the DATASETS Procedure" on page 359

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. Valid values are

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 also: "Restricting Processing for Generation Data Sets" on page 362

See also: "Understanding Generation Data Sets" in *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.

Aliases: MTYPE=, MT=

Default: If you do not specify MEMTYPE= in the PROC DATASETS statement, the default is MEMTYPE=ALL.

See also: "Restricting Member Types for Processing" on page 360

Details

- The CHANGE statement changes names by the order that the *old-names* occur in the directory listing, not in the order that you list the changes in the CHANGE statement.
- If the *old-name* SAS file does not exist in the SAS data 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 data library.

Reminder: You can use data set options with the DATA=, OUT=, and OUT2= options. See "Data Set Options" on page 18 for a list. You can use any global statements as well. See "Global Statements" on page 18.

Featured in: Example 4 on page 384

CONTENTS <*option(s)*>;

To do this	Use this option
Specify the input data set	DATA=
Specify the name for an output data set	OUT=
Specify the name of an output data set to contain information about indexes and integrity constraints	OUT2=
Include information in the output about the number of observations, number of variables, number of indexes, and data set labels	DETAILS | NODETAILS

To do this	Use this option
Print a list of the SAS files in the SAS data library	DIRECTORY
Print the length of a variable's informat or format	FMTLEN
Restrict processing to one or more types of SAS files	MEMTYPE=
Suppress the printing of individual files	NODS
Suppress the printing of the output	NOPRINT
Print a list of the variables by their position in the data set. By default, the CONTENTS statement lists the variables alphabetically.	VARNUM
Print a list of variables in alphabetical order even if they include mixed-case names	ORDER=IGNORECASE
Print abbreviated output	SHORT
Print centiles information for indexed variables	CENTILES

Options

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	percent of the data values that must be changed before the CENTILES for the indexed variables are automatically updated.
Current Update Percent	percent 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 data 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 data 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 data 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.

Featured in: Example 4 on page 384

DETAILS|NODETAILS

DETAILS includes these additional columns of information in the output, but only if DIRECTORY is also specified.

Default: If neither DETAILS or 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 on the PROC DATASETS statement, which also defaults to the system option setting.

See also: description of the additional columns in "Options" in "PROC DATASETS Statement" on page 308

DIRECTORY

prints a list of all SAS files in the specified SAS data library. If DETAILS is also specified, using DIRECTORY causes the additional columns described in DETAILS|NODETAILS on page 309 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(s)*)

restricts processing to one or more member types. The CONTENTS statement produces output only for member types DATA, VIEW, and ALL, which includes DATA and VIEW.

MEMTYPE= in the CONTENTS statement differs from MEMTYPE= in most of the other statements in the DATASETS procedure in the following ways:

- A slash does not precede the option.
- You cannot enclose the MEMTYPE= option in parentheses to limit its effect to only the SAS file immediately preceding it.

MEMTYPE= results in a directory of the library in which the DATA= member is located. However, MEMTYPE= does not limit the types of members whose contents are displayed unless the _ALL_ keyword is used in the DATA= option. For example, the following statements produce the contents of only the SAS data sets with the member type DATA:

```
proc datasets memtype=data;
   contents data=_all_;
run;
```

Aliases: 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 data library directory. You cannot use the NODS option when you specify only one SAS data set in the DATA= option.

NODETAILS
See the description of DETAILS|NODETAILS on page 325.

NOPRINT
suppresses printing the output of the CONTENTS statement.

ORDER= IGNORECASE | VARNUM

IGNORECASE	prints a list of variables in alphabetical order even if they include mixed-case names.
VARNUM	is the same as the VARNUM option. See VARNUM.

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 370 for a description of the variables in the OUT= data set.

See also: Example 7 on page 389 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.

Tip: 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.

Tip: OUT2= does not suppress the printed output from the statement. To suppress the printed output, use the NOPRINT option.

See also: "The OUT2= Data Set" on page 374 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.

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

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 366.

Requesting CONTENTS Output for a Password-Protected File with Integrity Constraints

For a SAS data file with defined referential integrity constraints that is also password protected, some SAS requests require that both files be open in order to process the request. If both files are password protected, then both passwords must be provided.

For example, suppose you want to execute the CONTENTS procedure for a data file with a primary key that is referenced by a foreign key. You must provide the password for the primary key data file as well as the password for the referential data file, because in order to obtain the information for the CONTENTS output for the primary key data file, SAS must open both files.

For an example, see “Understanding Integrity Constraints” in *SAS Language Reference: Concepts*.

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 either WORK or USER. For the CONTENTS statement, the default is the libref of the procedure input library.

COPY Statement

Copies all or some of the SAS files in a SAS library.

Featured in: Example 1 on page 376

COPY OUT=*libref-1*
<CLONE | NOCLONE>
<CONSTRAINT=YES | NO>
<DATECOPY>
<FORCE>
<IN=*libref-2*>
<INDEX=YES | NO>
<MEMTYPE=(*mtype(s)*)>
<MOVE <ALTER=*alter-password*>> ;

Required Arguments

OUT=*libref-1*
names the SAS library to copy SAS files to.

Aliases: OUTLIB= and OUTDD=

Featured in: Example 1 on page 376

Options

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 also: "Using Passwords with the DATASETS Procedure" on page 359

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.

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.

For the BUFSIZE= attribute, the following table summarizes how the COPY statement works:

Table 15.1 CLONE and the Buffer Page Size Attribute

If you use...	the COPY statement...
CLONE	uses the BUFSIZE= value from the input data set for the output data set.
NOCLONE	uses the current setting of the SAS system option BUFSIZE= for the output data set.
neither	determines the type of access method, sequential or random, used by the engine for the input data set and the engine for the output data set. If both engines use the same type of access, the COPY statement uses the BUFSIZE= value from the input data set for the output data set. If the engines do not use the same type of access, the COPY statement uses the setting of SAS system option BUFSIZE= for the output data set.

For the COMPRESS= and REUSE= attributes, the following table summarizes how the COPY statement works:

Table 15.2 CLONE and the Compression and Reuse Space Attributes

If you use…	the 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 compression and reuse space attributes, then the COPY statement uses the current setting of the corresponding SAS system option.
NOCLONE	uses the current setting of the SAS system options COMPRESS= and REUSE= for the output data set.
neither	defaults to CLONE.

For the OUTREP= attribute, the following table summarizes how the COPY statement works:

Table 15.3 CLONE and the Data Representation Attribute

If you use…	the COPY statement…
CLONE	results in a copy with the data representation of the input data set.
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 library.
neither	default is CLONE.

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 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 15.4 CLONE and the Encoding Attribute

If you use…	the 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.
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	default is 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 will be used in a encoding.

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 *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.

Restriction: DATECOPY cannot be used with encrypted files or catalogs.

Restriction: DATECOPY can be used only when the resulting SAS file uses the V8 or V9 engine.

Tip: You can alter the file creation date and time with the DTC= option on the MODIFY statement. See "MODIFY Statement" on page 348.

Tip: 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 this changes the last modified date 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. △

IN=*libref-2*
names the SAS library containing SAS files to copy.

Aliases: INLIB= and INDD=

Default: the libref of the procedure input library

To copy only selected members, use the SELECT or EXCLUDE statements.

INDEX=YES|NO
specifies whether to copy all indexes for a data set when copying the data set to another SAS data library.

Default: YES

MEMTYPE=(*mtype(s)*)
restricts processing to one or more member types.

Aliases: MT=, MTYPE=

Default: If you omit MEMTYPE= in the PROC DATASETS statement, the default is MEMTYPE=ALL.

See also: "Specifying Member Types When Copying or Moving SAS Files" on page 331

See also: "Member Types" on page 361

Featured in: Example 1 on page 376

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.

Featured in: Example 1 on page 376

NOCLONE
See the description of CLONE.

Copying an Entire Library

To copy an entire SAS data 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 331 and see Example 1 on page 376 for an example. Also, see “EXCLUDE Statement” on page 339 and “SELECT Statement” on page 356.

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 effected 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 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 or

MIXEDCASE, 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 is possible when the variable name that is already defined is exactly eight bytes long and ends in a digit. In that case, the truncated name is defined in the output data set and the name from the input data set is changed. For example,

```
options validvarname=mixedcase;
data test;
   lonvar10='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 LONVAR1 and placed in the output data set. Next, the original LONGVAR1 is copied. Its name is no longer unique and so it is renamed LONGVAR2. The other variables in the input data set are also renamed according to the renaming algorithm. △

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 differences are

- 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, to transport SAS data sets between hosts. See *Moving and Accessing SAS Files* for more information and an example.

DELETE Statement

Deletes SAS files from a SAS data library.

Featured in: Example 1 on page 376

DELETE *SAS-file(s)*
</ <ALTER=*alter-password*>
<GENNUM=ALL|HIST|REVERT|*integer*>
<MEMTYPE=*mtype*>>;

Required Arguments

SAS-file(s)
specifies one or more SAS files that you want to delete.

Options

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 also: "Using Passwords with the DATASETS Procedure" on page 359

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. Valid values are

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 also: "Restricting Processing for Generation Data Sets" on page 362

See also: "Understanding Generation Data Sets" in *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.

Aliases: MT=, MTYPE=

Default: DATA

See also: "Restricting Member Types for Processing" on page 360

Featured in: Example 1 on page 376

Details

- □ 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 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

When you are working with generation groups, you can use the DELETE statement to

- □ 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.

Featured in: Example 1 on page 376

EXCHANGE *name-1=other-name-1*
<*...name-n=other-name-n*>
</ <ALTER=*alter-password*>
<MEMTYPE=*mtype*>>;

Required Arguments

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.

Options

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 also: "Using Passwords with the DATASETS Procedure" on page 359

MEMTYPE=*mtype*
restricts processing to one member type. You can only exchange 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 also: "Restricting Member Types for Processing" on page 360

Details

- □ When you exchange more than one pair of names in one EXCHANGE statement, PROC DATASETS performs the exchanges in the order that the names of the SAS files occur in the directory listing, not in the order that you list the exchanges in the EXCHANGE statement.
- □ If the *name* SAS file does not exist in the SAS data 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.

Restriction: Must follow a COPY statement

Restriction: Cannot appear in the same COPY step with a SELECT statement

Featured in: Example 1 on page 376

EXCLUDE *SAS-file(s)* </ MEMTYPE=*mtype*>;

Required Arguments

SAS-file(s)
specifies one or more SAS files to exclude from the copy operation. All SAS files you name in the EXCLUDE statement must be in the library that is specified in the IN= option in the COPY statement. If the SAS files are generation groups, the EXCLUDE statement allows only selection of the base versions.

Options

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.

Aliases: MTYPE=, MT=

Default: If you do not specify MEMTYPE= in the PROC DATASETS statement, the COPY statement, or in the EXCLUDE statement, the default is MEMTYPE=ALL.

See also: “Restricting Member Types for Processing” on page 360

See also: “Specifying Member Types When Copying or Moving SAS Files” on page 331

Excluding Many Like-Named Files

You can use shortcuts for listing many SAS files in the EXCLUDE statement. For more information, see “Shortcuts for Specifying Lists of Variable Names” on page 24.

FORMAT Statement

Permanently assigns, changes, and removes variable formats in the SAS data set specified in the MODIFY statement.

Restriction: Must appear in a MODIFY RUN group

Featured in: Example 3 on page 381

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

Required Arguments

variable-list
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. For example:

```
format x1-x3 4.1 time hhmm2.2 age;
```

Options

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*.

Note: You can use shortcut methods for specifying variables, such as the keywords _NUMERIC, _CHARACTER_, and _ALL_. See "Shortcuts for Specifying Lists of Variable Names" on page 24 for more information. △

IC CREATE Statement

Creates an integrity constraint.

Restriction: Must be in a MODIFY RUN group

See also: "Understanding Integrity Constraints" in *SAS Language Reference: Concepts*

IC CREATE <*constraint-name*=> *constraint* <MESSAGE='*message-string*'
<MSGTYPE=USER>>;

Required Arguments

constraint
is the type of constraint. Valid values are as follows:

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 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.

Interaction: A primary key affects the values of an individual data file until it has a foreign key referencing it.

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.

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:

For a 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.

For a 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.

For a 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.

Interaction: Before it will enforce a SET NULL or CASCADE referential action, SAS checks to see if 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.

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.

- the foreign key's update and delete referential actions must both be RESTRICT.

Options

<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
CK*xxxx*	Check
PK*xxxx*	Primary key
FK*xxxx*	Foreign key

where *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. For example,

```
ic create not null(socsec)
   message='Invalid Social Security number';
```

Length: The maximum length of the message is 250 characters.

<MSGTYPE=USER> controls the format of the integrity constraint error message. By default when the MESSAGE= option is specified, the message 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.

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);
```

Note that 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.

IC DELETE Statement

Deletes an integrity constraint.

Restriction: Must be in a MODIFY RUN group

See also: "Understanding Integrity Constraints" in *SAS Language Reference: Concepts*

IC DELETE *constraint-name(s)* | _ALL_;

Arguments

constraint-name(s)
names one or more constraints to delete. For example, to delete the constraints Unique_D and Unique_E, use this statement:

```
ic delete Unique_D Unique_E;
```

ALL
deletes all constraints for the SAS data file specified in the preceding MODIFY statement.

IC REACTIVATE Statement

Reactivates a foreign key integrity constraint that is inactive.

Restriction: Must be in a MODIFY RUN group

See also: "Understanding Integrity Constraints" in *SAS Language Reference: Concepts*

IC REACTIVATE *foreign-key-name* REFERENCES *libref*;

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.

For 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

Updates centiles statistics for indexed variables.

Restriction: Must be in a MODIFY RUN group

See also: "Understanding SAS Indexes" in *SAS Language Reference: Concepts*

INDEX CENTILES *index(s)*
</ <REFRESH>
<UPDATECENTILES= ALWAYS|NEVER|*integer*>>;

Required Arguments

index(s)
names one or more indexes.

Options

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 percent of the data values that can be changed before centiles for the indexed variables are updated.

Valid values for UPDATECENTILES are

ALWAYS|0
updates centiles when the data set is closed if any changes have been made to the data set index.

NEVER|101
does not update centiles.

integer
is the percent 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 also: "Understanding SAS Indexes" in *SAS Language Reference: Concepts*

Featured in: Example 3 on page 381

INDEX CREATE *index-specification(s)*
</ <NOMISS>
<UNIQUE>
<UPDATECENTILES= ALWAYS|NEVER|*integer*>>;

Required Arguments

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 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*. △

Options

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*.

Note: BY-group processing ignores indexes that are created with the NOMISS option. △

Featured in: Example 3 on page 381

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.

Featured in: Example 3 on page 381

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 percent of the data values that can be changed before centiles for the indexed variables are updated. Valid values for UPDATECENTILES are as follows:

ALWAYS | 0
updates centiles when the data set is closed if any changes have been made to the data set index.

NEVER | 101
does not update centiles.

integer
specifies the percent 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

INDEX DELETE *index(s)* | _ALL_;

Required Arguments

index(s)
names one or more indexes to delete. The index(es) must be for variables in the SAS data set that is named in the preceding MODIFY statement. You can delete both simple and composite indexes.

ALL
deletes all indexes, except for indexes that are owned by an integrity constraint. When an index is created, it is marked as owned by the user, by an integrity constraint, or by both. If an index is owned by both a user and an integrity constraint, the index is not deleted until both an IC DELETE statement and an INDEX DELETE statement are processed.

Note: You can use the CONTENTS statement to produce a list of all indexes for a data set. △

INFORMAT Statement

Permanently assigns, changes, and removes variable informats in the data set specified in the MODIFY statement.

Restriction: Must appear in a MODIFY RUN group

Featured in: Example 3 on page 381

INFORMAT *variable-list-1 <informat-1>*
<*...variable-list-n <informat-n*>>;

Required Arguments

variable-list
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. For example:

```
informat a b 2. x1-x3 4.1 c;
```

Options

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*.

Note: You can use shortcut methods for specifying variables, such as the keywords _NUMERIC, _CHARACTER_, and _ALL_. See "Shortcuts for Specifying Lists of Variable Names" on page 24 for more information. △

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

Featured in: Example 3 on page 381

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

Required Arguments

variable=<'label'>
assigns a label to a variable. If a single quotation mark appears in the label, write it as two single quotation marks in the LABEL statement. Specifying *variable*= or *variable*=' 'removes the current label.

Range: 1-256 characters

MODIFY Statement

Changes the attributes of a SAS file and, through the use of subordinate statements, the attributes of variables in the SAS file.

Featured in: Example 3 on page 381

MODIFY *SAS-file* <(*option(s)*)>
</ <CORRECTENCODING=*encoding-value*>
<DTC=*SAS-date-time*>
<GENNUM=*integer*>
<MEMTYPE=*mtype*>>;

To do this	Use this option
Restrict processing to a certain type of SAS file	MEMTYPE=
Specify attributes	
Change the character-set encoding	CORRECTENCODING=
Specify a creation date and time	DTC=
Assign or change a data set label	LABEL=
Specify how the data are currently sorted	SORTEDBY=
Assign or change a special data set type	TYPE=
Modify passwords	
Modify an alter password	ALTER=
Modify a read, write, or alter password	PW=
Modify a read password	READ=
Modify a write password	WRITE=
Modify generation groups	
Modify the maximum number of versions for a generation group	GENMAX=
Modify a historical version	GENNUM=

Required Arguments

SAS-file
specifies a SAS file that exists in the procedure input library.

Options

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 also: "Manipulating Passwords" on page 351

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: The CORRECTENCODING= Option on the MODIFY Statement of the DATASETS Procedure in *SAS National Language Support (NLS): User's 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. For example:

```
modify mydata / dtc='03MAR00:12:01:00'dt;
```

Tip: Use DTC= to alter a SAS file's creation date and time prior to using the DATECOPY option in the CIMPORT procedure, COPY procedure, CPORT procedure, SORT procedure, and the COPY statement in the DATASETS procedure.

Restriction: A SAS file's creation date and time cannot be set later than the date and time the file was actually created.

Restriction: DTC= cannot be used with encrypted files or sequential files.

Restriction: DTC= can be used only when the resulting SAS file uses the V8 or V9 engine.

GENMAX=*number-of-generations*
specifies the maximum number of versions. You can use this option either in parentheses after the name of each SAS file or after a forward slash.

Range: 0 to 1,000

Default: 0

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 also: "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: 1-40 characters

Featured in: Example 3 on page 381

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.

Aliases: 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 also: "Manipulating Passwords" on page 351

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 also: "Manipulating Passwords" on page 351

Featured in: Example 3 on page 381

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 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 information.

Restriction: The data must be sorted in the order that you specify. If the data is not in the specified order, SAS will not sort it for you.

Featured in: Example 3 on page 381

TYPE=*special-type*
assigns or changes the special data set type of a SAS data set. SAS does *not* verify

- the SAS data set type 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 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 data 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 also: "Manipulating Passwords" on page 351

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;
```

RENAME Statement

Renames variables in the SAS data set specified in the MODIFY statement.

Restriction: Must appear in a MODIFY RUN group

Featured in: Example 3 on page 381

RENAME *old-name-1=new-name-1*
<*...old-name-n=new-name-n*>;

Required Arguments

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 SAS Variable Names" in *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.

REPAIR *SAS-file(s)*
</ <ALTER=*alter-password*>
<GENNUM=*integer*>
<MEMTYPE=*mtype*>>;

Required Arguments

SAS-file(s)
specifies one or more SAS data sets or catalogs in the procedure input library.

Options

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 also: "Using Passwords with the DATASETS Procedure" on page 359

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 also: "Restricting Processing for Generation Data Sets" on page 362

See also: "Understanding Generation Data Sets" in *SAS Language Reference: Concepts*

MEMTYPE=*mtype*
restricts processing to one member type.

Aliases: 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 also: "Restricting Member Types for Processing" on page 360

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 may 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 recreates all indexes for the data set. It also attempts to restore the data set to a usable condition, but the restored data set may not include the last several updates that occurred before the system failed. You cannot use the REPAIR statement to recreate 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 may 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 may 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 System 9, but not with Version 6.
- A damaged SAS System 9 catalog can be repaired with SAS System 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 may be lost during the repair process. For more information about CEDA, refer to "Processing Data Using Cross-Environment Data Access" in *SAS Language Reference: Concepts*.

SAVE Statement

Deletes all the SAS files in a library except the ones listed in the SAVE statement.

Featured in: Example 2 on page 380

SAVE *SAS-file(s)* </ MEMTYPE=*mtype*>;

Required Arguments

SAS-file(s)
specifies one or more SAS files that you do not want to delete from the SAS data library.

Options

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.

Aliases: 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 also: "Restricting Member Types for Processing" on page 360

Featured in: Example 2 on page 380

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.

  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, be 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.

Restriction: Must follow a COPY statement

Restriction: Cannot appear with an EXCLUDE statement in the same COPY step

Featured in: Example 1 on page 376

SELECT *SAS-file(s)*
</ <ALTER=*alter-password*>
<MEMTYPE= *mtype*>>;

Required Arguments

SAS-file(s)
specifies one or more SAS files that you want to copy. All of the SAS files that you name must be in the data library that is referenced by the libref named in the IN= option in the COPY statement. If the SAS files have generation groups, the SELECT statement allows only selection of the base versions.

Options

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 data 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 also: "Using Passwords with the DATASETS Procedure" on page 359

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.

Aliases: 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 also: "Specifying Member Types When Copying or Moving SAS Files" on page 331

See also: "Restricting Member Types for Processing" on page 360

Featured in: Example 1 on page 376

Selecting Many Like-Named Files

You can use shortcuts for listing many SAS files in the SELECT statement. For more information, see "Shortcuts for Specifying Lists of Variable Names" on page 24.

Concepts: DATASETS Procedure

Procedure Execution

Execution of Statements

When you start the DATASETS procedure, you specify the procedure input library in the PROC DATASETS statement. If you omit a procedure input library, the procedure processes the current default SAS data library (usually the WORK data library). To specify a new procedure input library, issue the DATASETS procedure again.

Statements execute in the order 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. It executes all of the preceding statements immediately, then 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-data-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.* If you do not know how passwords affect SAS files, refer to *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 327 and “MODIFY Statement” on page 348. △

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.

   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 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 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 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 name a member type or several member types in the PROC DATASETS statement, in most subsequent statements (except the CONTENTS and COPY statements), you can name only a subset of the list of member types included in the PROC DATASETS statement. The directory listing that the PROC DATASETS statement writes to the SAS log includes only those SAS files of the type specified in the MEMTYPE= option.

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 323, “COPY Statement” on page 327, and “MODIFY Statement” on page 348 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 15.5 on page 362 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 331. △

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

Table 15.5 on page 362 shows the member types that you can use in each statement:

Table 15.5 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[1]
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[2]
SAVE	ACCESS, ALL, CATALOG, DATA, FDB, MDDB, PROGRAM, VIEW	ALL
SELECT	ACCESS, ALL, CATALOG, DATA, FDB, MDDB, PROGRAM, VIEW	ALL

1 When DATA=_ALL_ in the CONTENTS statement, the default is ALL. ALL includes only DATA and VIEW.
2 ALL includes only DATA and CATALOG.

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.* If you do not know how to request and use generation data sets, refer to "Generation Data Sets" in *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 348. △

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. △

* 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.

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. △

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 data library. △

Note: If you specify the ODS RTF destination, the PROC DATASETS output will go to both the SAS log and the ODS output area. The NOLIST option will suppress 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 Output 15.3.

Only the items in the output that require explanation are discussed.

Data Set Attributes

Here are descriptions of selected fields shown in Output 15.3:

`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 number that can be stored in a double-precision integer, the count will show 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 will show as missing.

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), that 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 367 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.

Output 15.3 Data Set Attributes for the GROUP Data Set

```
                              The SAS System                                1

                          The DATASETS Procedure

Data Set Name        HEALTH.GROUP                      Observations          148
Member Type          DATA                              Variables             11
Engine               V9                                Indexes               1
Created              Wednesday, February               Observation Length    96
                     05, 2003 02:20:56
Last Modified        Wednesday, February               Deleted Observations  0
                     05, 2003 02:20:56
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.

Output 15.4 Engine and Operating Environment Dependent Information Section of CONTENTS Output

```
                       Engine/Host Dependent Information

Data Set Page Size          8192
Number of Data Set Pages    4
First Data Page             1
Max Obs per Page            84
Obs in First Data Page      62
Index File Page Size        4096
Number of Index File Pages  2
Number of Data Set Repairs  0
File Name                   c:\Myfiles\health\group.sas7bdat
Release Created             9.0101B0
Host Created                XP_PRO
```

Alphabetic List of Variables and Attributes

Here are descriptions of selected columns in Output 15.5:

#
: is the logical position of each variable in the observation. This is the number that is assigned to the variable when it 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 may 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 information on transcoding, see *SAS National Language Support (NLS): User's Guide*.

Note: If none of the variables in the SAS data set has a format, informat, or label associated with it, or if none of the variables are set to no transcoding, then the column for that attribute does not display. △

Output 15.5 Variable Attributes Section

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Informat	Label	Transcode
9	BIRTH	Num	8	DATE7.	DATE7.		YES
4	CITY	Char	15	$.	$.		NO
3	FNAME	Char	15	$.	$.		NO
10	HIRED	Num	8	DATE7.	DATE7.		YES
11	HPHONE	Char	12	$.	$.		YES
1	IDNUM	Char	4	$.	$.		YES
7	JOBCODE	Char	3	$.	$.		YES
2	LNAME	Char	15	$.	$.		YES
8	SALARY	Num	8	COMMA8.		current salary excluding bonus	YES
6	SEX	Char	1	$.	$.		YES
5	STATE	Char	2	$.	$.		YES

Alphabetic List of Indexes and Attributes

The section shown in Output 15.6 appears only if the data set has indexes associated with it.

#
: indicates the number of each index. The indexes are numbered sequentially as they are defined.

Index
: displays the name of each index. For simple indexes, the name of the index is the same as a variable in the data set.

Unique Option
: indicates whether the index must have unique values. If the column contains YES, the combination of values of the index variables is unique for each observation.

Nomiss Option
: indicates whether the index excludes missing values for all index variables. If the column contains YES, the index does not contain observations with missing values for all index variables.

# of Unique Values
: gives the number of unique values in the index.

Variables
: names the variables in a composite index.

Output 15.6 Index Attributes Section

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 Output 15.7 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 you use in the BY statement in PROC SORT, the column name in PROC SQL, or the values you specify in the SORTEDBY= option.

Validated
indicates whether PROC SORT or PROC SQL sorted the data. If PROC SORT or PROC SQL sorted the data set, the value is YES. If you assigned the sort information 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. This field does not appear if you do not specify a specific collating sequence that is different from the character set. (not shown)

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)

Output 15.7 Sort Information Section

```
                              The SAS System                              2

                          The DATASETS Procedure

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

PROC CONTENTS generates the same ODS tables as PROC DATASETS with the CONTENTS statement.

Table 15.6 ODS Tables Produced by the DATASETS Procedure without the CONTENTS Statement

ODS Table	Description	Table is generated:
Directory	General library information	unless you specify the NOLIST option.
Members	Library member information	unless you specify the NOLIST option.

Table 15.7 ODS Table Names Produced by PROC CONTENTS and PROC DATASETS with the CONTENTS Statement

ODS Table	Description	Table is generated:
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 integrtiy constraints and you do not specify the SHORT option.
IntegrityConstraintsShort	A concise listing of integrity constraints	if the data set has integrity constraints and you specify the SHORT option
Indexes	A detailed listing of indexes	if the data set is indexed and you do not specify the SHORT option.
IndexesShort	A concise listing of indexes	if the data set is indexed and you specify the SHORT option.
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.

ODS Table	Description	Table is generated:
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. These 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 sort information stored with it.

COLLATE
: the collating sequence used to sort the data set. A blank appears if the sort information 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 an SQL view is protected (`P`) or contributes (`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 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 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 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 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 if 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. Possible values are

. (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. Possible values are

. (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 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 4 on page 384 and in "Procedure Output" on page 364.

Output 15.8 The Data Set HEALTH.GRPOUT

```
                         An Example of an Output Data Set                       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       3        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       1        6
 11 HEALTH    GROUP   Test Subjects           STATE      2       2        5

OBS             LABEL               FORMAT  FORMATL  FORMATD  INFORMAT  INFORML

  1                                 DATE       7        0       DATE       7
  2                                 $          0        0       $          0
  3                                 $          0        0       $          0
  4                                 DATE       7        0       DATE       7
  5                                 $          0        0       $          0
  6                                 $          0        0       $          0
  7                                 $          0        0       $          0
  8                                 $          0        0       $          0
  9 current salary excluding bonus  COMMA      8        0                  0
 10                                 $          0        0       $          0
 11                                 $          0        0       $          0
```

```
                    An Example of an Output Data Set                       2

Obs INFORMD JUST NPOS NOBS ENGINE           CRDATE           MODATE DELOBS

  1    0      1    8   148   V9   29JAN02:08:06:46 29JAN02:09:13:36    0
  2    0      0   58   148   V9   29JAN02:08:06:46 29JAN02:09:13:36    0
  3    0      0   43   148   V9   29JAN02:08:06:46 29JAN02:09:13:36    0
  4    0      1   16   148   V9   29JAN02:08:06:46 29JAN02:09:13:36    0
  5    0      0   79   148   V9   29JAN02:08:06:46 29JAN02:09:13:36    0
  6    0      0   24   148   V9   29JAN02:08:06:46 29JAN02:09:13:36    0
  7    0      0   76   148   V9   29JAN02:08:06:46 29JAN02:09:13:36    0
  8    0      0   28   148   V9   29JAN02:08:06:46 29JAN02:09:13:36    0
  9    0      1    0   148   V9   29JAN02:08:06:46 29JAN02:09:13:36    0
 10    0      0   75   148   V9   29JAN02:08:06:46 29JAN02:09:13:36    0
 11    0      0   73   148   V9   29JAN02:08:06:46 29JAN02:09:13:36    0

OBS IDXUSAGE  MEMTYPE IDXCOUNT PROTECT FLAGS COMPRESS REUSE SORTED SORTEDBY

  1 COMPOSITE  DATA       1       R--    ---     NO     NO      0       .
  2 NONE       DATA       1       R--    ---     NO     NO      0       .
  3 NONE       DATA       1       R--    ---     NO     NO      0       .
  4 NONE       DATA       1       R--    ---     NO     NO      0       .
  5 NONE       DATA       1       R--    ---     NO     NO      0       .
  6 NONE       DATA       1       R--    ---     NO     NO      0       .
  7 NONE       DATA       1       R--    ---     NO     NO      0       .
  8 NONE       DATA       1       R--    ---     NO     NO      0       1
  9 COMPOSITE  DATA       1       R--    ---     NO     NO      0       .
 10 NONE       DATA       1       R--    ---     NO     NO      0       .
 11 NONE       DATA       1       R--    ---     NO     NO      0       .
```

```
                         An Example of an Output Data Set                       3

OBS CHARSET COLLATE NODUPKEY NODUPREC ENCRYPT POINTOBS GENMAX GENNUM GENNEXT

  1  ANSI              NO       NO       NO      YES       0      .      .
  2  ANSI              NO       NO       NO      YES       0      .      .
  3  ANSI              NO       NO       NO      YES       0      .      .
  4  ANSI              NO       NO       NO      YES       0      .      .
  5  ANSI              NO       NO       NO      YES       0      .      .
  6  ANSI              NO       NO       NO      YES       0      .      .
  7  ANSI              NO       NO       NO      YES       0      .      .
  8  ANSI              NO       NO       NO      YES       0      .      .
  9  ANSI              NO       NO       NO      YES       0      .      .
 10  ANSI              NO       NO       NO      YES       0      .      .
 11  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 7 on page 389. △

The OUT2= Data Set

The OUT2= option in the CONTENTS statement creates an output data set that contains information about indexes and integrity constraints. These 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 will be 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 recreate 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, etc.).

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: Manipulating SAS Files

Procedure 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=

This example

- □ changes the names of SAS files
- □ copies SAS files between SAS data libraries
- □ deletes SAS files
- □ selects SAS files to copy
- □ exchanges the names of SAS files
- □ excludes SAS files from a copy operation.

Program

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-data-library-1';
libname dest2 'SAS-data-library-2';
libname health 'SAS-data-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 data views. MOVE specifies that all SAS data 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 data 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 a 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

```
1    options pagesize=60 linesize=80 nodate pageno=1 source;
2    libname dest1 'c:\Myfiles\dest1';
NOTE: Libref DEST1 was successfully assigned as follows:
      Engine:        V9
      Physical Name: c:\Myfiles\dest1
3    libname dest2 'c:\Myfiles\dest2';
NOTE: Libref DEST2 was successfully assigned as follows:
      Engine:        V9
      Physical Name: c:\Myfiles\dest2
4    libname health 'c:\Myfiles\health';
NOTE: Libref HEALTH was successfully assigned as follows:
      Engine:        V9
      Physical Name: c:\Myfiles\health
5    proc datasets library=health details;
                                   Directory

              Libref         HEALTH
              Engine         V9
              Physical Name  c:\Myfiles\health
              File Name      c:\Myfiles\health


              Member   Obs, Entries                    File
 #  Name      Type      or Indexes   Vars  Label       Size  Last Modified

 1  A1        CATALOG       23                        62464   19FEB2002:14:41:15
 2  A2        CATALOG        1                        17408   19FEB2002:14:41:15
 3  ALL       DATA          23        17              13312   19FEB2002:14:41:19
 4  BODYFAT   DATA           1         2               5120   19FEB2002:14:41:19
 5  CONFOUND  DATA           8         4               5120   19FEB2002:14:41:19
 6  CORONARY  DATA          39         4               5120   19FEB2002:14:41:20
 7  DRUG1     DATA           6         2  JAN95        5120   19FEB2002:14:41:20
                                          Data
 8  DRUG2     DATA          13         2  MAY95        5120   19FEB2002:14:41:20
                                          Data
 9  DRUG3     DATA          11         2  JUL95        5120   19FEB2002:14:41:20
                                          Data
10  DRUG4     DATA           7         2  JAN92        5120   19FEB2002:14:41:20
                                          Data
11  DRUG5     DATA           1         2  JUL92        5120   19FEB2002:14:41:20
                                          Data
12  ETEST1    CATALOG        1                        17408   19FEB2002:14:41:20
13  ETEST2    CATALOG        1                        17408   19FEB2002:14:41:20
14  ETEST3    CATALOG        1                        17408   19FEB2002:14:41:20
15  ETEST4    CATALOG        1                        17408   19FEB2002:14:41:20
16  ETEST5    CATALOG        1                        17408   19FEB2002:14:41:20
17  ETESTS    CATALOG        1                        17408   19FEB2002:14:41:21
18  FORMATS   CATALOG        6                        17408   19FEB2002:14:41:21
19  GROUP     DATA         148        11              25600   19FEB2002:14:41:21
20  INFANT    DATA         149         6              17408   05FEB2002:12:52:30
21  MLSCL     DATA          32         4  Multiple     5120   19FEB2002:14:41:21
                                          Sclerosi
                                          s Data
22  NAMES     DATA           7         4               5120   19FEB2002:14:41:21
23  OXYGEN    DATA          31         7               9216   19FEB2002:14:41:21
24  PERSONL   DATA         148        11              25600   19FEB2002:14:41:21
25  PHARM     DATA           6         3  Sugar        5120   19FEB2002:14:41:21
                                          Study
26  POINTS    DATA           6         6               5120   19FEB2002:14:41:21
27  PRENAT    DATA         149         6              17408   19FEB2002:14:41:22
28  RESULTS   DATA          10         5               5120   19FEB2002:14:41:22
29  SLEEP     DATA         108         6               9216   19FEB2002:14:41:22
30  SPDATA    VIEW           .         2               5120   19FEB2002:14:41:29
31  SYNDROME  DATA          46         8               9216   19FEB2002:14:41:22
32  TENSION   DATA           4         3               5120   19FEB2002:14:41:22
33  TEST2     DATA          15         5               5120   19FEB2002:14:41:22
34  TRAIN     DATA           7         2               5120   19FEB2002:14:41:22
35  VISION    DATA          16         3               5120   19FEB2002:14:41:22
36  WEIGHT    DATA          83        13  Californ    13312   19FEB2002:14:41:22
                                          ia
                                          Results
```

```
37  WGHT      DATA          83        13   Californ   13312   19FEB2002:14:41:23
                                           ia
                                           Results
6       delete tension a2(mt=catalog);
7       change a1=postdrug;
8       exchange weight=bodyfat;
NOTE: Deleting HEALTH.TENSION (memtype=DATA).
NOTE: Deleting HEALTH.A2 (memtype=CATALOG).
NOTE: Changing the name HEALTH.A1 to HEALTH.POSTDRUG (memtype=CATALOG).
NOTE: Exchanging the names HEALTH.WEIGHT and HEALTH.BODYFAT (memtype=DATA).
9       copy out=dest1 move memtype=view;
10         select spdata;
11         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).
12      copy out=dest2;
13         exclude d: mlscl oxygen test2 vision weight;
14   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.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.PRENAT to DEST2.PRENAT (memtype=DATA).
NOTE: There were 149 observations read from the data set HEALTH.PRENAT.
NOTE: The data set DEST2.PRENAT has 149 observations and 6 variables.
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.SYNDROME to DEST2.SYNDROME (memtype=DATA).
NOTE: There were 46 observations read from the data set HEALTH.SYNDROME.
NOTE: The data set DEST2.SYNDROME has 46 observations and 8 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.
```

Example 2: Saving SAS Files from Deletion

Procedure 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

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-data-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

```
41   options pagesize=40 linesize=80 nodate pageno=1 source;
42   libname elder 'c:\Myfiles\elder';
NOTE: Libref ELDER was successfully assigned as follows:
      Engine:        V9
      Physical Name: c:\Myfiles\elder
43   proc datasets lib=elder;
                                   Directory

Libref         ELDER
Engine         V9
Physical Name  c:\Myfiles\elder
File Name      c:\Myfiles\elder


                            Member     File
               #  Name      Type       Size  Last Modified

               1  AGING     DATA       5120   06FEB2003:08:51:21
               2  ALCOHOL   DATA       5120   06FEB2003:08:51:21
               3  BACKPAIN  DATA       5120   06FEB2003:08:51:21
               4  CHRONIC   DATA       5120   06FEB2003:08:51:21
               5  CLINICS   CATALOG   17408   06FEB2003:08:51:21
               6  CLINICS   DATA       5120   06FEB2003:08:51:21
               7  DISEASE   DATA       5120   06FEB2003:08:51:21
               8  GROWTH    DATA       5120   06FEB2003:08:51:21
               9  HOSPITAL  CATALOG   17408   06FEB2003:08:51:21
44      save chronic aging clinics / memtype=data;
45   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 3: Modifying SAS Data Sets

Procedure 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

This example modifies two SAS data sets using the MODIFY statement and statements subordinate to it. Example 4 on page 384 shows the modifications to the GROUP data set.

Tasks include

- 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

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-data-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

```
6    options pagesize=40 linesize=80 nodate pageno=1 source;
7    libname health 'c:\Myfiles\health';
NOTE: Libref HEALTH was successfully assigned as follows:
      Engine:        V9
      Physical Name: c:\Myfiles\health

8    proc datasets library=health nolist;
9       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.
10      index create vital=(birth salary) / nomiss unique;
NOTE: Composite index vital has been defined.
11      informat birth date7.;
12      format birth date7.;
13      label salary='current salary excluding bonus';
14      modify oxygen;
15      rename oxygen=intake;
NOTE: Renaming variable oxygen to intake.
16      label intake='Intake Measurement';
17   quit;

NOTE: MODIFY was successful for HEALTH.OXYGEN.DATA.
NOTE: PROCEDURE DATASETS used (Total process time):
      real time           16.96 seconds
      cpu time            0.73 seconds
```

Example 4: Describing a SAS Data Set

Procedure 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 3 on page 381.

Program

```
options pagesize=40 linesize=132 nodate pageno=1;
```

```
libname health 'SAS-data-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 370.

```
   contents data=group (read=green) out=grpout;
   title  'The Contents of the GROUP Data Set';
run;
```

Output

Output 15.9 The Contents of the GROUP Data Set

```
                          The Contents of the GROUP Data Set                                  1

                                The DATASETS Procedure

      Data Set Name        HEALTH.GROUP                                Observations          148
      Member Type          DATA                                        Variables             11
      Engine               V9                                          Indexes               1
      Created              Wednesday, February 05, 2003 02:20:56       Observation Length    96
      Last Modified        Thursday, February 06, 2003 09:07:54        Deleted Observations  0
      Protection           READ                                        Compressed            NO
      Data Set Type                                                    Sorted                YES
      Label                Test Subjects
      Data Representation  WINDOWS_32
      Encoding             wlatin1  Western (Windows)


                          Engine/Host Dependent Information

       Data Set Page Size          8192
       Number of Data Set Pages    4
       First Data Page             1
       Max Obs per Page            84
       Obs in First Data Page      62
       Index File Page Size        4096
       Number of Index File Pages  2
       Number of Data Set Repairs  0
       File Name                   c:\Myfiles\health\group.sas7bdat
       Release Created             9.0101B0
       Host Created                XP_PRO


                      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    $.         $.
```

```
                              The Contents of the GROUP Data Set                              2

                                    The DATASETS Procedure

                          Alphabetic List of Variables and Attributes

          #    Variable    Type    Len    Format     Informat    Label

          1    IDNUM       Char      4    $.         $.
          7    JOBCODE     Char      3    $.         $.
          2    LNAME       Char     15    $.         $.
          8    SALARY      Num       8    COMMA8.                current salary excluding bonus
          6    SEX         Char      1    $.         $.
          5    STATE       Char      2    $.         $.


                           Alphabetic List of Indexes and Attributes

                                                      # of
                                  Unique    NoMiss    Unique
                        #    Index    Option    Option    Values    Variables

                        1    vital    YES       YES          148    BIRTH SALARY


                                       Sort Information

                                     Sortedby       LNAME
                                     Validated      NO
                                     Character Set  ANSI
```

Example 5: Concatenating Two SAS Data Sets

Procedure features:

APPEND statement options:

BASE=
DATA=
FORCE=

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

Input Data Sets

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-data-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 15.10

```
                  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 6: Aging SAS Data Sets

Procedure features:

AGE statement

This example shows how the AGE statement ages SAS files.

Program

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-data-library';
```

Specify DAILY as the procedure input library and suppress the directory listing.

```
proc datasets library=daily nolist;
```

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 7: PROC CONTENTS ODS Output

Procedures features:
CONTENTS Statement

The example shows how to get PROC CONTENTS output into an ODS output data set for processing.

Program

```
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 will be 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;
```

Output 15.11 PROC CONTENTS ODS Output

```
PROC CONTENTS ODS Output
all Attributes data


Member    Label1                 cValue1

WORK.A    Data Set Name          WORK.A
WORK.A    Member Type            DATA
WORK.A    Engine                 V9
WORK.A    Created                Thursday, October 10, 2002 00:56:03
WORK.A    Last Modified          Thursday, October 10, 2002 00:56:03
WORK.A    Protection
WORK.A    Data Set Type
WORK.A    Label
WORK.A    Data Representation    WINDOWS_32
WORK.A    Encoding               wlatin1  Western (Windows)

                                        c
     nValue1    Label2                 Value2         nValue2

           .    Observations             1           1.000000
           .    Variables                1           1.000000
           .    Indexes                  0                  0
  1349873763    Observation Length       8           8.000000
  1349873763    Deleted Observations     0                  0
           .    Compressed               NO                 .
           .    Sorted                   NO                 .
           .                                                0
           .                                                0
           .                                                0
```

```
PROC CONTENTS ODS Output
all Variables data

Member    Num    Variable    Type    Len    Pos

WORK.A     1         x       Num       8      0
```

```
PROC CONTENTS ODS Output
all EngineHost data

Member    Label1

WORK.A    Data Set Page Size
WORK.A    Number of Data Set Pages
WORK.A    First Data Page
WORK.A    Max Obs per Page
WORK.A    Obs in First Data Page
WORK.A    Number of Data Set Repairs
WORK.A    File Name
WORK.A    Release Created
WORK.A    Host Created

cValue1                                                                     nValue1

4096                                                                    4096.000000
1                                                                          1.000000
1                                                                          1.000000
501                                                                      501.000000
1                                                                          1.000000
0                                                                                 0
C:\DOCUME~1\userid\LOCALS~1\Temp\SAS Temporary Files\_TD3084\a.sas7bdat           .
9.0101B0                                                                          .
XP_PRO                                                                            .
```

```
PROC CONTENTS ODS Output
example of post-processing of ODS output data

Member    attribute              value

WORK.A    Data Representation    WINDOWS_32
WORK.A    Encoding               wlatin1  Western (Windows)
WORK.A    Observations           1
WORK.A    Variables              1
```

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

CHAPTER
16

The DBCSTAB Procedure

Information about the DBCSTAB Procedure

See: For documentation of the DBCSTAB procedure, see *SAS National Language Support (NLS): User's Guide*.

CHAPTER

17

The DISPLAY Procedure

Overview: DISPLAY Procedure

The DISPLAY procedure executes SAS/AF applications. These applications are composed 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 *SAS Guide to 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

Featured in: Example 1 on page 396

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 data 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

Options

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: DISPLAY Procedure

Example 1: Executing a SAS/AF Application

Procedure features:
PROC DISPLAY statement:
CATALOG = argument

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 data library, as a FRAME entry in a catalog named INVOICES.WIDGETS. You can execute this application using the following SAS code:

Program

```
proc display catalog=sasuser.invoices.widgets.frame;
run;
```

CHAPTER

18

The DOCUMENT Procedure

Information about the DOCUMENT Procedure

See: For complete documentation of the DOCUMENT procedure, see *SAS Output Delivery System: User's Guide*.

CHAPTER
19

The EXPLODE Procedure

Information about the EXPLODE Procedure

See: For documentation of the EXPLODE procedure, go to **`http://support.sas.com/documentation/onlinedoc`**. Select Base SAS from the Product-Specific Documentation list.

CHAPTER
20

The EXPORT Procedure

Overview: EXPORT Procedure

The EXPORT procedure reads data from a SAS data set and writes it to an external data source. External data sources can include Microsoft Access Database, Excel files, Lotus spreadsheets, and delimited external files (in which columns of data values are separated by a delimiter such as a blank, comma, or tab).

When you execute PROC EXPORT, the procedure reads the input data set and writes the data to the external data source. PROC EXPORT exports the data by one of the following methods:

- 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. PROC EXPORT produces the specified output file and writes information about the export to the SAS log. In the log, you see the DATA step or the SAS/ACCESS code that is generated by PROC EXPORT. If a translation engine is used, then no code is submitted.

Note: To export data, you can also use the Export Wizard, which is a windowing tool that guides you through the steps to export a SAS data set. You can request the Export Wizard to generate EXPORT procedure statements, which you can save to a file for subsequent use. To invoke the Export Wizard, from the SAS windowing environment select

File ▶ Export Data

△

Syntax: EXPORT Procedure

Restriction: PROC EXPORT is available for the following operating environments:

- OpenVMS Alpha
- UNIX
- Microsoft Windows.

PROC EXPORT DATA=<*libref.*>*SAS-data-set* <(*SAS-data-set-options*)>
OUTFILE="*filename*" | OUTTABLE="*tablename*"
<DBMS=*identifier*> <REPLACE>;
<*data-source-statement(s)*;>

PROC EXPORT Statement

Featured in: All examples

PROC EXPORT DATA=<*libref.*>*SAS-data-set* <(*SAS-data-set-options*)>
OUTFILE="*filename*" | OUTTABLE="*tablename*"
<DBMS=*identifier*> <REPLACE>;

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, PROC EXPORT uses either the USER library (if assigned) or the WORK library (if USER not assigned).

Default: If you do not specify a SAS data set, PROC EXPORT uses the most recently created SAS data set, which SAS keeps track of with the system variable _LAST_. However, in order to be certain that PROC EXPORT uses the correct data set, you should identify the SAS data set.

Restriction: PROC EXPORT can export data only if the format of the data is supported by the data source or the amount of data is within the limitations of the data source. For example, some data sources have a maximum number of rows or columns, and some data sources cannot support SAS user-defined formats and informats. If the data that you want to export exceeds the limits of the data source, PROC EXPORT may not be able to export it correctly. When incompatible formats are encountered, the procedure formats the data to the best of its ability.

Restriction: PROC EXPORT does not support writing labels as column names. However, SAS does support column names up to 32 characters.

Featured in: All examples

(*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 option, or to export only data that meets a specified condition, you can use the WHERE= data set option. For information about SAS data set options, see “Data Set Options” in *SAS Language Reference: Dictionary*.

Restriction: You cannot specify data set options when exporting delimited, comma-separated, or tab-delimited external files.

Featured in: Example 2 on page 414

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 does 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 the output file. To assign a fileref, use the FILENAME statement. For more information about PC file formats, see *SAS/ACCESS for PC Files: Reference*.

Featured in Example 1 on page 411, Example 2 on page 414, and Example 3 on page 415

Restriction: PROC EXPORT does not support device types or access methods for the FILENAME statement except for DISK. For example, PROC EXPORT does not support the TEMP device type, which creates a temporary external file.

Restriction: For client/server applications: When running SAS/ACCESS software on UNIX to access data that is stored on a PC server, you must specify the full path and filename of the file that you want to import. The use of a fileref is not supported.

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 may be case sensitive.

Requirement: When you export a DBMS table, you must specify the DBMS= option.

Featured in: Example 4 on page 415

Options

DBMS=*identifier*

specifies the type of data to export. To export a DBMS table, you must specify DBMS= by using a valid database identifier. For example, DBMS=ACCESS specifies to export a table into a Microsoft Access 2000 or 2002 database. To export PC files, spreadsheets, and delimited external files, you do not have to specify DBMS= if the filename that is specified in OUTFILE= contains a valid extension so that PROC EXPORT can recognize the type of data. For example, PROC EXPORT recognizes the filename ACCOUNTS.WK1 as a Lotus 1-2-3 Release 2 spreadsheet and the filename MYDATA.CSV as an external file that contains comma-separated data values; therefore, a DBMS= specification is not necessary.

The following values are valid for the DBMS= option:

Identifier	Output Data Source	Extension	Host Availability	Version of File Created
ACCESS	Microsoft Access 2000 or 2002 table	.mdb	Microsoft Windows *	2000
ACCESS97	Microsoft Access 97 table	.mdb	Microsoft Windows *	97

Identifier	Output Data Source	Extension	Host Availability	Version of File Created
ACCESS2000	Microsoft Access 2000 table	.mdb	Microsoft Windows *	2000
ACCESS2002	Microsoft Access 2002 table	.mdb	Microsoft Windows *	2000
ACCESSCS	Microsoft Access table	.mdb	UNIX	2000**
CSV	delimited file (comma-separated values)	.csv	OpenVMS Alpha, UNIX, Microsoft Windows	
DBF	dBASE 5.0, IV, III+, and III files	.dbf	UNIX, Microsoft Windows	5.0
DLM	delimited file (default delimiter is a blank)	.*	OpenVMS Alpha, UNIX, Microsoft Windows	
EXCEL	Excel 97 or 2000 or 2002 spreadsheet	.xls	Microsoft Windows *	97
EXCEL4	Excel 4.0 spreadsheet	.xls	Microsoft Windows	4.0
EXCEL5	Excel 5.0 or 7.0 (95) spreadsheet	.xls	Microsoft Windows	5.0
EXCEL97	Excel 97 spreadsheet	.xls	Microsoft Windows *	97
EXCEL2000	Excel 2000 spreadsheet	.xls	Microsoft Windows *	97
EXCEL2002	Excel 2002 spreadsheet	.xls	Microsoft Windows *	97
EXCELCS	Excel spreadsheet	.xls	UNIX	97**
JMP	JMP table	.jmp	UNIX, Microsoft Windows	
PCFS	Files on PC server	.*	UNIX	
TAB	delimited file (tab-delimited values)	.txt	OpenVMS Alpha, UNIX, Microsoft Windows	
WK1	Lotus 1-2-3 Release 2 spreadsheet	.wk1	Microsoft Windows	

Identifier	Output Data Source	Extension	Host Availability	Version of File Created
WK3	Lotus 1-2-3 Release 3 spreadsheet	.wk3	Microsoft Windows	
WK4	Lotus 1-2-3 Release 4 and 5 spreadsheet	.wk4	Microsoft Windows	

* Not available for Microsoft Windows 64-Bit Edition.** Value listed here is the default value. The real version of file loaded depends on the version of the existing file or the value specified for VERSION= statement.

Restriction: The availability of an output data source depends on

- the operating environment, and in some cases the platform, as specified in the previous table.
- whether your site has a license to the SAS/ACCESS software for PC file formats. If you do not have a license, only delimited files are available.

Featured in: Example 1 on page 411 and Example 4 on page 415

When you specify a value for DBMS=, consider the following for specific data sources:

- To export to an existing Microsoft Access database, PROC EXPORT can write to Access 97, Access 2000, or Access 2002 regardless of your specification. For example, if you specify DBMS=ACCESS2000 and the database is in Access 97 format, PROC EXPORT exports the table, and the database remains in Access 97 format. However, if you specify OUTFILE= for an Access database that does not exist, a new database is created using the format specified in DBMS=. For example to create a new Access database, specifying DBMS=ACCESS (which defaults to Access 2000 or 2002 format) creates an MDB file that can be read by Access 2000 or Access 2002, not by Access 97.

  The following table lists the DBMS= specifications and indicates which version of Microsoft Access can open the resulting database:

Specification	Access 2002	Access 2000	Access 97
ACCESS	yes	yes	no
ACCESS2002	yes	yes	no
ACCESS2000	yes	yes	no
ACCESS97	yes	yes	yes

- To export a Microsoft Excel spreadsheet, PROC EXPORT creates an XLS file for the version specified. The following table lists the DBMS= specifications and indicates which version of Microsoft Excel can open the resulting spreadsheet:

Specification	Excel 2002	Excel 2000	Excel 97	Excel 5.0	Excel 4.0
EXCEL	yes	yes	yes	no	no
EXCEL2002	yes	yes	yes	no	no
EXCEL2000	yes	yes	yes	no	no

Specification	Excel 2002	Excel 2000	Excel 97	Excel 5.0	Excel 4.0
EXCEL97	yes	yes	yes	no	no
EXCEL5	yes	yes	yes	yes	no
EXCEL4	yes	yes	yes	yes	yes

Note: Later versions of Excel can open and update files in earlier formats. △

- When exporting a SAS data set to a dBASE file (DBF), if the data set contains missing values (for either character or numeric values), the missing values are translated to blanks.
- When exporting a SAS data set to a dBASE file (DBF), values for a character variable that are longer than 255 characters are truncated in the resulting dBASE file because of dBASE limitations.

REPLACE
overwrites an existing file. Note that for a Microsoft Access database or an Excel workbook, REPLACE overwrites the target table or spreadsheet. If you do not specify REPLACE, PROC EXPORT does not overwrite an existing file.

Featured in: Example 2 on page 414 and Example 4 on page 415

Data Source Statements

PROC EXPORT provides a variety of statements that are specific to the output data source.

Statements for PC Files, Spreadsheets, or Delimited Files

The following statement is available when you export delimited external files:

DELIMITER='*char*' | '*nn*'*x*;
specifies the delimiter to separate columns of data 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='&'. If you do not specify DELIMITER=, PROC EXPORT assumes that the delimiter is a blank. You can replace the equal sign with a blank.

Interaction: You do not have to specify DELIMITER= if you specify DBMS=CSV, DBMS=TAB, or if the output filename has an extension of .CSV or .TXT.

Featured in: Example 1 on page 411

SHEET=*spreadsheet-name*;
identifies a particular spreadsheet name to load into a workbook. You use this statement for Microsoft Excel 97, 2000, or 2002 only. If the SHEET= statement is not specified, PROC EXPORT uses the SAS data set name as the spreadsheet name to load the data.

For Excel data access, a spreadsheet name is treated as a special case of a range name with a dollar sign ($) appended. For example, if you export a table and specify **`sheet=Invoice`**, you will see a range (table) name INVOICE and another range (table) name 'INVOICES$' created. Excel appends a dollar sign ($) to a spreadsheet name in order to distinguish it from the corresponding range name.

Note: You should not append the dollar sign ($) when you specify the spreadsheet name. For example, SHEET= 'Invoice$' is not allowed. △

You should avoid using special characters for spreadsheet names when exporting a table to an Excel file. Special characters such as a space or a hyphen are replaced with an underscore. For example, if you export a table and specify **`sheet='Sheet Number 1'`**, PROC EXPORT creates the range names **`Sheet_Number_1`** and **`Sheet_Number_1$`**.

Featured in: Example 3 on page 415

Statements for DBMS Tables

The following statements are available to establish a connection to the DBMS when you are exporting to a DBMS table:

DATABASE="*database*";
: specifies the complete path and filename of the database to contain the specified DBMS table. If the database name does not contain lowercase characters, special characters, or national characters ($, #, or @), you can omit the quotation marks. You can replace the equal sign with a blank.

  Note: A default may be configured in the DBMS client software; SAS does not generate a default value. △

  Featured in: Example 4 on page 415

DBPWD="*database-password*";
: specifies a password that allows access to a database. You can replace the equal sign with a blank.

PWD="*password*";
: specifies the user password used by the DBMS to validate a specific userid. If the password does not contain lowercase characters, special characters, or national characters, you can omit the quotation marks. You can replace the equal sign with a blank.

  Note: The DBMS client software may default to the userid and password that was used to log in to the operating environment; SAS does not generate a default value. △

UID="*userid*";
: identifies the user to the DBMS. If the userid does not contain lowercase characters, special characters, or national characters, you can omit the quotation marks. You can replace the equal sign with a blank.

  Note: The DBMS client software may default to the userid and password that were used to log in to the operating environment; SAS does not generate a default value. △

WGDB="*workgroup-database-name*";
: specifies the workgroup (security) database name that contains the USERID and PWD data for the DBMS. If the workgroup database name does not contain lowercase characters, special characters, or national characters, you can omit the quotation marks. You can replace the equal sign with a blank.

Note: A default workgroup database may be used by the DBMS; SAS does not generate a default value. △

Security Levels for Microsoft Access Tables

Microsoft Access tables have the following levels of security, for which specific combinations of security statements must be used:

None
: Do not specify DBPWD=, PWD=, UID=, or WGDB=.

Password
: Specify only DBPWD=.

User-level
: Specify only PWD=, UID=, and WGDB=.

Full
: Specify DBPWD=, PWD=, UID=, and WGDB=.

Each statement has a default value; however, you may find it necessary to provide a value for each statement explicitly.

Statement for Client/Server Model

The following statements are available to establish a connection from SAS running on UNIX to a PC server when you are exporting a table to Microsoft Access database or Excel workbook:

SERVER="*PC-server-name*";
: specifies the name of the PC server. You must bring up the listener on the PC server before you can establish a connection to it. You can configure the service name, port number, maximum number of connections allowed, and use of data encryption on your PC server. This is a required statement. Refer to your PC server administrator for the information that is needed.

  Alias: SERVER_NAME=

SERVICE="*service-name*";
: specifies the service name that is defined on your service file for your client and server machines. This statement and the PORT= statement should not be used in the same procedure. Note that this service name needs to be defined on both your UNIX machine and your PC server.

  Alias: SERVER_NAME=, SERVICE_NAME=

PORT=*port-number*;
: specifies the number of the port that is listening on the PC server. The valid value is between 1 and 32767. This statement and the SERVICE= statement should not be used in the same procedure.

  Alias: PORT_NUMBER=

VERSION="*file-version*";
: specifies the version of the file that you want to create with if the file does not exist on your PC server yet. The default version is data-source specific. For Microsoft Access database, the valid values are '2002', '2000' and '97', and its default value is '2000'. For Microsoft Excel workbook, the valid values are '2002', '2000', '97', '95' and '5', and its default value is '97'.

  Note: Always quote the version value. △

  Note: If the file already exists in the PC Server, then this value can be ignored. △

Examples: EXPORT Procedure

Example 1: Exporting a Delimited External File

Procedure features:

PROC EXPORT statement arguments:

DATA=
DBMS=
OUTFILE=

Data source statement:

DELIMITER=

This example exports the following SAS data set named SASHELP.CLASS and creates a delimited external file:

Output 20.1 PROC PRINT of SASHELP.CLASS

```
                          The SAS System                              1

     Obs    Name       Sex          Age        Height       Weight

       1    Alfred      M            14          69          112.5
       2    Alice       F            13          56.5         84
       3    Barbara     F            13          65.3         98
       4    Carol       F            14          62.8        102.5
       5    Henry       M            14          63.5        102.5
       6    James       M            12          57.3         83
       7    Jane        F            12          59.8         84.5
       8    Janet       F            15          62.5        112.5
       9    Jeffrey     M            13          62.5         84
      10    John        M            12          59           99.5
      11    Joyce       F            11          51.3         50.5
      12    Judy        F            14          64.3         90
      13    Louise      F            12          56.3         77
      14    Mary        F            15          66.5        112
      15    Philip      M            16          72          150
      16    Robert      M            12          64.8        128
      17    Ronald      M            15          67          133
      18    Thomas      M            11          57.5         85
      19    William     M            15          66.5        112
```

Program

Identify the input SAS data set, specify the output filename, and specify the type of file. Note that the filename does not contain an extension. DBMS=DLM specifies that the output file is a delimited external file.

```
proc export data=sashelp.class
   outfile='c:\myfiles\class'
   dbms=dlm;
```

Specify the delimiter. The DELIMITER= option specifies that an & (ampersand) will delimit data fields in the output file. The delimiter separates the columns of data in the output file.

```
   delimiter='&';
run;
```

SAS Log

The SAS log displays the following information about the successful export. Notice the generated SAS DATA step.

```
47     /**********************************************************************
48     *   PRODUCT:   SAS
49     *   VERSION:   9.00
50     *   CREATOR:   External File Interface
51     *   DATE:      07FEB02
52     *   DESC:      Generated SAS Datastep Code
53     *   TEMPLATE SOURCE:  (None Specified.)
54     ***********************************************************************/
55        data _null_;
56        set  SASHELP.CLASS                                end=EFIEOD;
57        %let _EFIERR_ = 0; /* set the ERROR detection macro variable */
58        %let _EFIREC_ = 0;     /* clear export record count macro variable */
59        file 'c:\myfiles\class' delimiter='&' DSD DROPOVER
59 ! lrecl=32767;
60           format Name $8. ;
61           format Sex $1. ;
62           format Age best12. ;
63           format Height best12. ;
64           format Weight best12. ;
65        if _n_ = 1 then        /* write column names */
66         do;
67           put
68           'Name'
69           '&'
70           'Sex'
71           '&'
72           'Age'
73           '&'
74           'Height'
75           '&'
76           'Weight'
77           ;
78         end;
79         do;
80           EFIOUT + 1;
81           put Name $ @;
82           put Sex $ @;
83           put Age @;
84           put Height @;
85           put Weight ;
86           ;
87         end;
88        if _ERROR_ then call symput('_EFIERR_',1);  /* set ERROR detection
88 ! macro variable */
89        If EFIEOD then
90           call symput('_EFIREC_',EFIOUT);
91        run;

NOTE: Numeric values have been converted to character
      values at the places given by: (Line):(Column).
      88:44   90:31
NOTE: The file 'c:\myfiles\class' is:
      File Name=c:\myfiles\class,
      RECFM=V,LRECL=32767

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           0.13 seconds
      cpu time            0.05 seconds


19 records created in c:\myfiles\class from SASHELP.CLASS
                  .


NOTE: c:\myfiles\class was successfully created.
```

Output

The external file produced by PROC EXPORT follows.

```
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 an Excel Spreadsheet

Procedure features:

PROC EXPORT statement arguments:

DATA=
DBMS=
OUTFILE=
REPLACE

This example exports the SAS data set SASHELP.CLASS, shown in Output 20.1. PROC EXPORT creates an Excel file named Femalelist.xsl, and by default, creates a spreadsheet named Class. Since the SHEET= data source statement is not specified, PROC EXPORT uses the name of the SAS data set as the spreadsheet name. The WHERE= SAS data set option is specified in order to export a subset of the observations, which results in the spreadsheet containing only the female students.

Program

Identify the input SAS data set, request a subset of the observations, specify the output data source, specify the output file, and overwrite the target spreadsheet if it exists. The output file is an Excel 2000 spreadsheet.

```
proc export data=sashelp.class (where=(sex='F'))
   outfile='c:\myfiles\Femalelist.xls'
```

```
      dbms=excel
      replace;
   run;
```

Example 3: Exporting to a Specific Spreadsheet in an Excel Workbook

Procedure features:

PROC EXPORT statement arguments:

- DATA=
- DBMS=
- OUTFILE=

Data Source Statement:

- SHEET=

This example exports a SAS data set named MYFILES.GRADES1 and creates an Excel 2000 workbook named Grades.xsl. MYFILES.GRADES1 becomes one spreadsheet in the workbook named Grades1.

Program

Identify the input SAS data set, specify the output data source, and specify the output file.

```
proc export data=myfiles.grades1
   dbms=excel2000
   outfile='c:\Myfiles\Grades.xls';
```

Identify a particular spreadsheet to write to in a workbook.

```
   sheet=Grades1;
run;
```

Example 4: Exporting a Microsoft Access Table

Procedure features:

PROC EXPORT statement arguments:

- DATA=
- DBMS=
- OUTTABLE=
- REPLACE

Data Source Statement:

- DATABASE=

This example exports a SAS data set named SASUSER.CUST, the first five observations of which follow, and creates a Microsoft Access 97 table. The security level for this Access table is none, so it is not necessary to specify any of the database security statements.

```
Obs     Name                    Street                    Zipcode

 1      David Taylor            124 Oxbow Street          72511
 2      Theo Barnes             2412 McAllen Avenue       72513
 3      Lydia Stirog            12550 Overton Place       72516
 4      Anton Niroles           486 Gypsum Street         72511
 5      Cheryl Gaspar           36 E. Broadway            72515
```

Program

Identify the input SAS data set, specify the output DBMS table name and the output data source, and overwrite the output file if it exists. The output file is a Microsoft Access 97 table. The option REPLACE overwrites an existing file. If you do not specify REPLACE, PROC EXPORT does not overwrite an existing file.

```
proc export data=sasuser.cust
   outtable="customers"
   dbms=access97
   replace;
```

Specify the path and filename of the database to contain the table.

```
   database="c:\myfiles\mydatabase.mdb";
run;
```

Example 5: Exporting a Specific Spreadsheet in an Excel Workbook on a PC Server

Procedure features:

PROC EXPORT statement arguments:

DATA=
DBMS=
OUTFILE=

Data Source Statement:

SHEET=
SERVER=
PORT=
VERSION=

This example exports a SAS data set named SASHELP.CLASS and creates an Excel 2000 workbook named demo.xls. SASHELP.CLASS becomes one spreadsheet named ’Class’ in the workbook named demo.xls.

Program

```
proc export data=sashelp.class
    dbms=excelcs
    outfile='c:\Myfiles\demo.xls';
    sheet='Class';
    server='sales';
    port= 4632;
    version='2000';
run;
```

CHAPTER 21

The 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 technology 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 *FreeType fonts*.

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. △

Syntax: FONTREG Procedure

Interaction: If no statements are specified, then PROC FONTREG searches for TrueType font files in the directory that is indicated in the FONTSLOC= SAS system option.

Tip: If more than one statement is specified, then the statements are executed in the order in which they appear. 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*'>;
FONTPATH '*directory*' <...'*directory*'>;
TRUETYPE '*directory*' <...'*directory*'>;
TYPE1 '*directory*' <...'*directory*'>;

Operating Environment Information: For z/OS sites that do not use the hierarchical file system (HFS), only the FONTFILE statement is supported. See "FONTREG Procedure" in *SAS Companion for z/OS* for details. △

PROC FONTREG Statement

PROC FONTREG <*option(s)*>;

Options

MODE=ADD | REPLACE | ALL
specifies how to handle new and existing fonts in the SAS registry:

ADD
add fonts that do not already exist in the SAS registry. Do not modify existing fonts.

REPLACE
replace fonts that already exist in the SAS registry. Do not add new fonts.

ALL
add new fonts that do not already exist in the SAS registry and replace fonts that already exist in the SAS registry.

Default: ADD

Featured in: Example 3 on page 427

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, as well as 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

Featured in: Example 2 on page 426

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.

Featured in: Example 1 on page 425

FONTFILE '*file*' <…'*file*'>;

Argument

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.

FONTPATH Statement

Specifies one or more directories to be searched for valid font files to process.

Featured in: Example 2 on page 426

FONTPATH '*directory*' <…'*directory*'>;

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.

TRUETYPE Statement

Specifies one or more directories to be searched for TrueType font files.

Featured in: Example 3 on page 427

TRUETYPE *'directory'* <…*'directory'*>;

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 (Experimental)

Specifies one or more directories to be searched for valid Type 1 font files.

TYPE1 *'directory'* <…*'directory'*>;

CAUTION:
TYPE1 is an experimental statement that is available in SAS 9.1. Do not use this statement in production jobs. △

Argument

directory
specifies a directory to search. Only files that are recognized as valid Type 1 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.

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 (< >), that 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, then SAS searches the registry for fonts with that name. If more than one font with that name is encountered, then SAS uses the one that has the highest rank in the following table.

Table 21.1 Supported Font Types

Rank	Type	Tag	File extension(s)
1	TrueType	**<ttf>**	**.ttf**
2	Type1	**<at1>**	**.pfa** **.pfb**
3	PFR	**<pfr>**	**.pfr**

CAUTION:
Support for the Type1 and PFR font types is experimental in SAS 9.1. Do not use these fonts in production jobs. △

Note: SAS does not support nonscalable FreeType fonts of any type. Even if they are recognized as valid FreeType fonts, they will not be added to the SAS registry. △

Font files that are not produced by major vendors can be unreliable, and in some cases SAS might not be able to use them.

The following SAS output methods and device drivers can use FreeType fonts:

- SAS/GRAPH GIF, GIF733, GIFANIM
- SAS/GRAPH JPEG
- SAS/GRAPH PNG
- SAS/GRAPH SASEMF
- SAS/GRAPH SASWMF
- SAS/GRAPH TIFFP, TIFFB
- Universal Printing GIF
- Universal Printing PCL
- Universal Printing PDF.

Removing Fonts from the SAS Registry

There are two ways to remove a font from the SAS registry:

□ by using the SAS Registry Editor

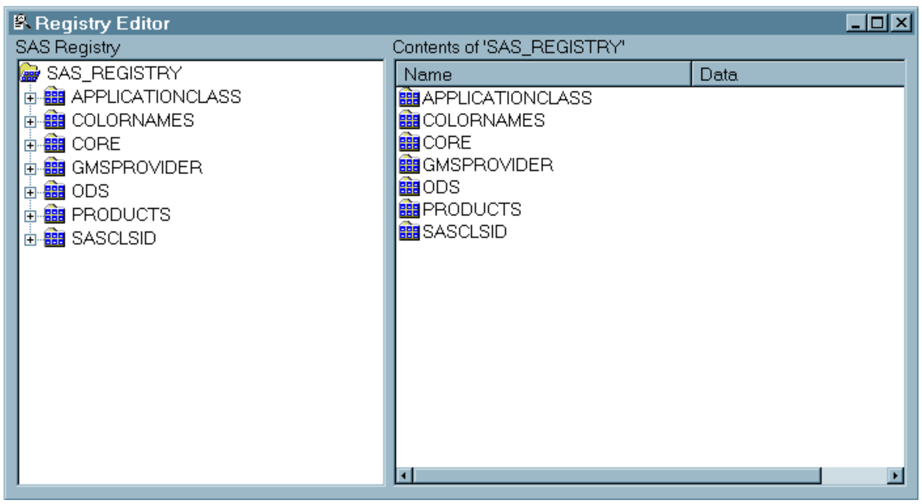
□ by using the REGISTRY procedure.

To remove a font by using the SAS Registry Editor, select

Solutions ▶ Accessories ▶ Registry Editor

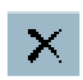
(Alternatively, you can type **regedit** in the command window or **Command ===>** prompt.)

Display 21.1 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**.

□ 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;
```

For more information about PROC REGISTRY, see Chapter 41, "The REGISTRY Procedure," on page 831.

Modifying SAS/GRAPH Device Drivers to Use System Fonts

To access FreeType fonts with the SAS/GRAPH device drivers, the CHARREC field of the device driver entry must be modified from its default value of **`DMS Font`** to any FreeType font. It is recommended that you use the **`<ttf> SAS Monospace`** font for this purpose, because it is shipped with SAS and is always available in the SAS registry. Changing the CHARREC value in this way enables you to use any FreeType font in your SAS/GRAPH programs.

Here is an example that shows how to modify the CHARREC field:

```
/* Assign a location for the personal devices catalog */
libname gdevice0 '.';

/* Create a new GIF device driver, FTGIF, */
/* that will recognize FreeType fonts     */
proc gdevice nofs c=gdevice0.devices;
   copy GIF from=sashelp.devices newname=FTGIF;
   mod FTGIF charrec=(0, 1, 1, '<ttf> SAS Monospace', 'Y');
end;
```

The following device drivers can be modified to recognize FreeType fonts:

- GIF, GIF733, GIFANIM
- JPEG
- PNG
- TIFFP, TIFFB.

The SASWMF and SASEMF device drivers do not require this change.

For more information about SAS/GRAPH device drivers and the GDEVICE procedure, see *SAS/GRAPH Reference, Volumes 1 and 2*.

Examples: FONTREG Procedure

Example 1: Adding a Single Font File

Procedure features: FONTFILE statement

This example shows how to add a single font file to the SAS registry.

Program

Specify a font file to add. The FONTFILE statement specifies the complete path to a single font file.

```
proc fontreg;
   fontfile 'your-font-file';
run;
```

Log

```
NOTE: PROCEDURE PRINTTO used (Total process time):
      real time           0.03 seconds
      cpu time            0.00 seconds

20   proc fontreg;
21      fontfile 'your-font-file';
22   run;
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

NOTE: PROCEDURE FONTREG used (Total process time):
      real time           0.17 seconds
      cpu time            0.03 seconds
```

Example 2: Adding All Font Files from Multiple Directories

Procedure features:

MSGLEVEL= option

FONTPATH statement

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

Write complete details to the SAS log. The MSGLEVEL=VERBOSE option writes complete details about what fonts were added, what fonts were not added, and what font files were not understood.

```
proc fontreg msglevel=verbose;
```

Specify the directories to search for valid fonts. You can specify more than one directory in the FONTPATH statement. Each directory must be enclosed in quotation marks. If you specify more than one directory, then you must separate the directories with a space.

```
   fontpath 'your-font-directory-1' 'your-font-directory-2';
run;
```

Log (Partial)

```
NOTE: PROCEDURE PRINTTO used (Total process time):
      real time           0.03 seconds
      cpu time            0.00 seconds

34   proc fontreg msglevel=verbose;
35      fontpath 'your-font-directory-1'
36               'your-font-directory-2';
37   run;

ERROR: FreeType base module FT_New_Face -- unknown file format.
WARNING: A problem was encountered with file
         your-font-directory-2\SCRIPT.FON.

NOTE: The font Albertus Extra Bold (Style: Regular, Weight: Bold) has been
      added to the SAS Registry at [CORE\PRINTING\FREETYPE\FONTS\<ttf>
      Albertus Extra Bold]. Since it is a TRUETYPE font, it must be
      referenced as <ttf> Albertus Extra Bold in SAS. The font resides in
      file
      your-font-directory-1\albr85w.ttf.

. . . more log entries . . .

SUMMARY:
  Files processed: 138
  Unusable files: 4
  Files identified as fonts: 134
  Fonts that were processed: 134
  Fonts replaced in the SAS registry: 0
  Fonts added to the SAS registry: 127
  Fonts that could not be used: 7

NOTE: PROCEDURE FONTREG used (Total process time):
      real time           7.11 seconds
      cpu time            3.80 seconds
```

Example 3: Replacing Existing TrueType Font Files from a Directory

Procedure features:

MODE= option

TRUETYPE statement

This example reads all the TrueType Fonts in the specified directory and replaces those that already exist in the SAS registry.

Program

Replace existing fonts only. The MODE=REPLACE option limits the action of the procedure to replacing fonts that are already defined in the SAS registry. New fonts will not be added.

```
proc fontreg mode=replace;
```

Specify a directory that contains TrueType font files. Files in the directory that are not recognized as being TrueType font files are ignored.

```
   truetype 'your-font-directory';
run;
```

Log

```
53   proc fontreg mode=replace;
54      truetype 'your-font-directory';
55   run;
SUMMARY:
  Files processed: 49
  Unusable files: 4
  Files identified as fonts: 45
  Fonts that were processed: 39
  Fonts replaced in the SAS registry: 39
  Fonts added to the SAS registry: 0
  Fonts that could not be used: 0

NOTE: PROCEDURE FONTREG used (Total process time):
      real time           1.39 seconds
      cpu time            0.63 seconds
```

See Also

- The GDEVICE procedure in *SAS/GRAPH Reference, Volumes 1 and 2*
- The FONTSLOC and SYSPRINTFONT SAS system options in *SAS Language Reference: Dictionary*
- **http://www.freetype.org** for more information about the FreeType project.

CHAPTER 22

The 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 the SAS System 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 forth). The SAS System provides informats and formats for reading and writing variables. For a thorough description of informats and formats that SAS provides, see the sections on formats and informats in *SAS Language Reference: Dictionary*.

With informats, you can

- 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 through 9 to 1, 10 through 100 to 2, and so forth.

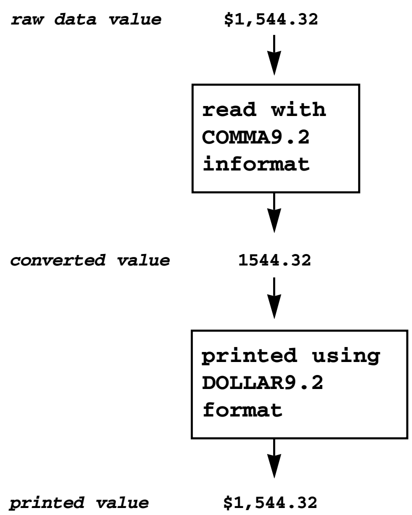
With formats, you can

- 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 22.1 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 455.

Syntax: FORMAT Procedure

Restriction: You cannot use a SELECT statement and an EXCLUDE statement within the same PROC FORMAT step.

Reminder: You can also use appropriate global statements with this procedure. See “Global Statements” on page 18 for a list.

See: FORMAT Procedure in the documentation for your operating environment.

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);

To do this	Use this statement
Exclude catalog entries from processing by the FMTLIB and CNTLOUT= options	EXCLUDE
Create an informat for reading and converting raw data values	INVALUE
Create a template for printing numbers	PICTURE
Select catalog entries from processing by the FMTLIB and CNTLOUT= options	SELECT
Create a format that specifies character strings to use to print variable values	VALUE

PROC FORMAT Statement

Reminder: You can use data set options with the CNTLIN= and CNTLOUT= data set options. See Section 2, "Fundamental Concepts for Using Base SAS Procedures," for a list.

PROC FORMAT <*option(s)*>;

To do this	Use this option
Specify a SAS data set from which PROC FORMAT builds informats or formats	CNTLIN=
Create a SAS data set that stores information about informats or formats	CNTLOUT=
Print information about informats or formats	FMTLIB
Specify a SAS library or catalog that will contain the informats or formats that you are creating in the PROC FORMAT step	LIBRARY=
Specify the number of characters of the informatted or formatted value that appear in PROC FORMAT output	MAXLABLEN=
Specify the number of characters of the start and end values that appear in the PROC FORMAT output	MAXSELEN=
Prevent a new informat or format from replacing an existing one of the same name	NOREPLACE
Print information about each format and informat on a separate page[1]	PAGE

1 Used in conjunction with FMTLIB. If PAGE is specified, FMTLIB is invoked (or assumed).

Options

CNTLIN=*input-control-SAS-data-set*
specifies a SAS data set from which PROC FORMAT builds informats and 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 data library (either the WORK data library or USER data library) for the data set, regardless of whether you specify the LIBRARY= option.

Note: LIBRARY= can point to either a data 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 also: "Input Control Data Set" on page 460

Featured in: Example 5 on page 472

CNTLOUT=*output-control-SAS-data-set*
creates a SAS data set that stores information about informats and formats that are contained in the catalog specified in the LIBRARY= option.

Note: LIBRARY= can point to either a data library or a catalog. If only a libref is specified, then a catalog name of FORMATS is assumed. △

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 data library (either the WORK data library or the USER data library), regardless of whether you specify the LIBRARY= option.

Tip: You can use an output control data set as an input control data set in subsequent PROC FORMAT steps.

See also: "Output Control Data Set" on page 458

FMTLIB
prints information about all the informats and formats in the catalog that is specified in the LIBRARY= option. To get information only about specific informats or formats, subset the catalog using the SELECT or EXCLUDE statement.

Interaction: The PAGE option invokes FMTLIB.

Tip: If your output from FMTLIB is not formatted correctly, then try increasing the value of the LINESIZE= system option.

Tip: 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.

Featured in: Example 6 on page 477

LIBRARY=*libref<.catalog>*
specifies a catalog to contain informats or formats that you are creating in the current 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.

Note: LIBRARY= can point to either a data library or a catalog. If only a libref is specified, then a catalog name of FORMATS is assumed. △

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.

Tip: SAS automatically searches LIBRARY.FORMATS. You might want to 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 further information about FMTSEARCH=, see the section on SAS system options in *SAS Language Reference: Dictionary*.

See also: "Storing Informats and Formats" on page 456

Featured in: Example 1 on page 464

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 that you are creating from replacing an existing informat or format 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 (that is, each entry) in the catalog on a separate page.

Tip: The PAGE option activates the FMTLIB option.

EXCLUDE Statement

Excludes entries from processing by the FMTLIB and CNTLOUT= options.

Restriction: Only one EXCLUDE statement can appear in a PROC FORMAT step.

Restriction: You cannot use a SELECT statement and an EXCLUDE statement within the same PROC FORMAT step.

EXCLUDE *entry(s)*;

Required Arguments

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.

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 FMTLIB.

INVALUE Statement

Creates an informat for reading and converting raw data values.

Featured in: Example 4 on page 470.

See also: The section on informats in *SAS Language Reference: Dictionary* for documentation on informats supplied by SAS.

INVALUE <$>*name* <(*informat-option(s)*)>
<*value-range-set(s)*>;

To do this	Use this option
Specify the default length of the informat	DEFAULT=
Specify a fuzz factor for matching values to a range	FUZZ=
Specify a maximum length for the informat	MAX=
Specify a minimum length for the informat	MIN=
Store values or ranges in the order that you define them	NOTSORTED
Left-justify all input strings before they are compared to ranges	JUST
Uppercase all input strings before they are compared to ranges	UPCASE

Required Arguments

name
names the informat that you are creating.

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; this is why a character informat is limited to 30 characters.

Restriction: A user-defined informat name cannot be the same as an informat name that is supplied by SAS.

Interaction: The maximum length of an informat name is controlled by the VALIDFMTNAME= SAS system option. See *SAS Language Reference: Dictionary* for details on VALIDFMTNAME=.

Tip: 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.

Tip: 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; this 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.

Options

The following options are common to the INVALUE, PICTURE, and VALUE statements and are described in "Informat and Format Options" on page 451:

DEFAULT=*length*

FUZZ= *fuzz-factor*

MAX=*length*

MIN=*length*

NOTSORTED

In addition, you can use the following options:

JUST
left-justifies all input strings before they are compared to the ranges.

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 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.

informatted-value
is the value 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 2 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 a user-defined informat. The informat you are creating uses the existing informat to convert the raw data that match *value-or-range* on the left side of the equals 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.*

value-or-range
: See "Specifying Values or Ranges" on page 453.

Consider the following examples:

- 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.

- When you are creating 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;
```

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.

- 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_;
```

PICTURE Statement

Creates a template for printing numbers.

Featured in: Example 1 on page 464 and Example 9 on page 483

See also: The section on formats in *SAS Language Reference: Dictionary* for documentation on formats supplied by SAS.

PICTURE *name* <(*format-option(s)*)>
<*value-range-set-1* <(*picture-1-option(s)*)>
<*...value-range-set-n* <(*picture-n-option(s)*)>>>;

To do this	Use this option
Control the attributes of the format	
Specify that you can use directives in the picture as a template to format date, time, or datetime values	DATATYPE=
Specify the default length of the format	DEFAULT=
Specify the separator character for the fractional part of a number	DECSEP=
Specify the three-digit separator character for a number	DIG3SEP=
Specify a fuzz factor for matching values to a range	FUZZ=
Specify a maximum length for the format	MAX=
Specify a minimum length for the format	MIN=
Specify multiple pictures for a given value or range and for overlapping ranges	MULTILABEL
Store values or ranges in the order that you define them	NOTSORTED
Round the value to the nearest integer before formatting	ROUND
Control the attributes of each picture in the format	
Specify a character that completes the formatted value	FILL=
Specify a number to multiply the variable's value by before it is formatted	MULTIPLIER=

To do this	Use this option
Specify that numbers are message characters rather than digit selectors	NOEDIT
Specify a character prefix for the formatted value	PREFIX=

Required Arguments

name
names the format you are creating.

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, then use a dollar sign ($) as the first character, which is why a character informat is limited to 30 characters.

Restriction: A user-defined format cannot be the name of a format supplied by SAS.

Interaction: The maximum length of a format name is controlled by the VALIDFMTNAME= SAS system option. See *SAS Language Reference: Dictionary* for details on VALIDFMTNAME=.

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.

Options

The following options are common to the INVALUE, PICTURE, and VALUE statements and are described in "Informat and Format Options" on page 451:

DEFAULT= *length*

FUZZ=*fuzz-factor*

MAX=*length*

MIN=*length*

NOTSORTED

In addition, you can use the following arguments:

DATATYPE=DATE | TIME | DATETIME
specifies that you can use *directives* in the picture as a template to format date, time, or datetime values. See the definition and list of directives on page 442.

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.

DECSEP='*character*'
specifies the separator character for the fractional part of a number.

Default: . (a decimal point)

DIG3SEP='*character*'
specifies the three-digit separator character for a number.

Default: , (a comma)

FILL='*character*'

specifies a character that completes the formatted value. If the number of significant digits is less than the length of the format, then the format must complete, or fill, the formatted value:

- The format uses *character* to fill the formatted value if you specify zeros as digit selectors.
- The format uses zeros to fill the formatted value if you specify nonzero digit selectors. The FILL= option has no effect.

If the picture includes other characters, such as a comma, which appear to the left of the digit selector that maps to the last significant digit placed, then the characters are replaced by the fill character or leading zeros.

Default: ' ' (a blank)

Interaction: If you use the FILL= and PREFIX= options in the same picture, then the format places the prefix and then the fill characters.

Featured in: Example 9 on page 483

MULTILABEL

allows the assignment of multiple labels or external values to internal values. 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 internal value. Multiple labels may also be assigned to a single range of internal values. The second PICTURE statement assigns labels to overlapping ranges of internal values. The MULTILABEL option allows the assignment of multiple labels to the overlapped internal 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 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. For example, in the first PICTURE statement, the primary label for 1000 is 1,000 because the format 9,999 is the first external value that is assigned to 1000. The secondary label for 1000 is 1000, based on the 9999 format.

In the second PICTURE statement, the primary label for 5 is 5.000 based on the 9.999 format that is assigned to the range 0–9 because 0–9 is sequentially the first range of internal values containing 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 an internal value. Unless you use the NOTSORTED option when you assign variables, the SAS System stores the variables in sorted order. This may produce unexpected results when variables with the MULTILABEL format are processed. For 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.

MULTIPLIER=*n*
specifies a number that the variable's value is to be multiplied by before it is formatted. For example, the following PICTURE statement creates the MILLION. format, which formats the variable value 1600000 as **`$1.6M`**:

```
picture million low-high='00.0M'
        (prefix='$' mult=.00001);
```

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 1 on page 464

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);
```

PREFIX='*prefix*'
specifies a character prefix to place in front of the value's first significant digit. You must use zero digit selectors or the prefix will not be used.

The picture must be wide enough to contain both the value and the prefix. If the picture is not wide enough to contain both the value and the prefix, then the format truncates or omits the prefix. 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.

Featured in: Example 1 on page 464 and Example 9 on page 483

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: Note that the ROUND option rounds a value of .5 to the next highest integer.

value-range-set

specifies one or more variable values and a template for printing those values. The *value-range-set* is the following:

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 chapter 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';
```

For 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.

Restriction: You can only use directives when you specify the DATATYPE= option in the PICTURE statement.

The permitted directives are

%a	Locale's abbreviated weekday name
%A	Locale's full weekday name
%b	Locale's abbreviated month name
%B	Locale's full month name
%d	Day of the month as a decimal number (1–31), with no leading zero
%H	Hour (24-hour clock) as a decimal number (0–23), with no leading zero
%I	Hour (12-hour clock) as a decimal number (1–12), with no leading zero
%j	Day of the year as a decimal number (1–366), with no leading zero
%m	Month as a decimal number (1–12), with no leading zero
%M	Minute as a decimal number (0–59), with no leading zero

%p	Locale's equivalent of either AM or PM
%S	Second as a decimal number (0–59), with no leading zero
%U	Week number of the year (Sunday as the first day of the week) as a decimal number (0,53), with no leading zero
%w	Weekday as a decimal number (1= Sunday, 7=Saturday)
%y	Year without century as a decimal number (0–99), with no leading zero
%Y	Year with century as a decimal number
%%	%

Any directive that generates numbers can produce a leading zero, if desired, by adding a 0 before the directive. This applies to %d, %H, %I, %j, %m, %M, %S, %U, and %y. For example, if you specify %y in the picture, then 2001 would be formatted as '1', but if you specify %0y, then 2001 would be formatted as '01'.

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 453.

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:

```
options nodate pageno=1 linesize=64 pagesize=60;

data sample;
   input Amount;
   datalines;
-2.051
-.05
-.017
  0
 .093
 .54
 .556
6.6
14.63
;


proc print data=sample;
   title 'Default Printing of the Variable Amount';
run;
```

```
            Default Printing of the Variable Amount           1

                         Obs     Amount

                          1      -2.051
                          2      -0.050
                          3      -0.017
                          4       0.000
                          5       0.093
                          6       0.540
                          7       0.556
                          8       6.600
                          9      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-data-library';


proc format library=library;
   picture nozeros (round)
           low -  -1  =  '00.00'
              (prefix='-')
           -1 <-<  0  =     '99'
              (prefix='-.' mult=100)
            0  -< 1   =     '99'
              (prefix='.'  mult=100)
            1  - high =  '00.00';
run;
```

The following table explains how one value from each range is formatted. Figure 22.1 on page 446 provides an illustration of each step. The circled numbers in the figure correspond to the step numbers in the table.

Table 22.1 Building a Picture Format

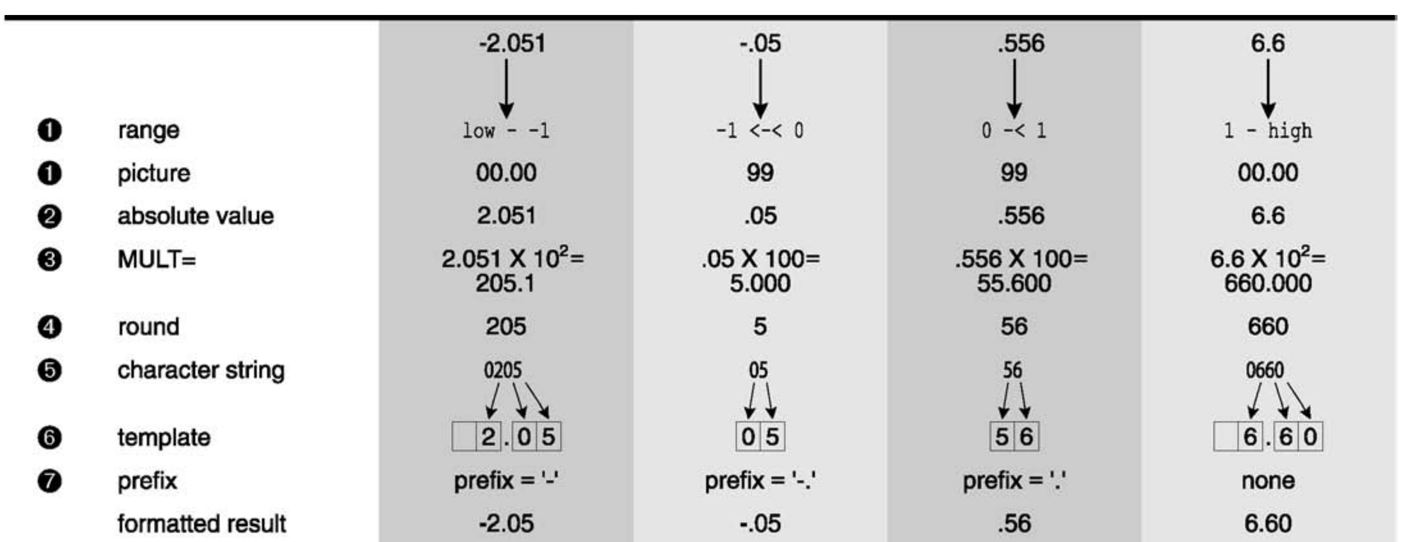
Step	Rule	In this example
1	Determine into which range the value falls and use that picture.	In the second range, the exclusion operator < appears on both sides of the hyphen and excludes –1 and 0 from the range.
2	Take the absolute value of the numeric value.	Because the absolute value is used, you need a separate range and picture for the negative numbers in order to prefix the minus sign.

Step	Rule	In this example
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[1] 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 two 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	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.	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 Language Reference: Dictionary*.	The numbers 205, 5, and 660 become the character strings **0205**, **05**, 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 55 because the corresponding picture only has 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 rightmost *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.[2]	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**.

1 A decimal in a PREFIX= option is not part of the picture.
2 You can use the FILL= option to specify a character other than a blank to become part of the formatted value.

Figure 22.1 Formatting One Value in Each Range



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;
```

```
                 Formatting the Variable Amount               1
                    with the NOZEROS. Format

                             Amount

                             -2.05
                              -.05
                              -.02
                               .00
                               .09
                               .54
                               .56
                              6.60
                             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. △

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 from processing by the FMTLIB and CNTLOUT= options.

Restriction: Only one SELECT statement can appear in a PROC FORMAT step.

Restriction: You cannot use a SELECT statement and an EXCLUDE statement within the same PROC FORMAT step.

Featured in: Example 6 on page 477.

SELECT *entry(s)*;

Required Arguments

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.

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;
```

FMTLIB Output

If you use the SELECT statement without either FMTLIB or CNTLOUT= in the PROC FORMAT statement, then the procedure invokes FMTLIB.

VALUE Statement

Creates a format that specifies character strings to use to print variable values.

Featured in: Example 2 on page 466.

See also: The chapter about formats in *SAS Language Reference: Dictionary* for documentation about SAS formats.

VALUE <$>*name* <(*format-option(s)*)>
<*value-range-set(s)*>;

To do this	Use this option
Specify the default length of the format	DEFAULT=
Specify a fuzz factor for matching values to a range	FUZZ=
Specify a maximum length for the format	MAX=

To do this	Use this option
Specify a minimum length for the format	MIN=
Specify multiple values for a given range, or for overlapping ranges	MULTILABEL
Store values or ranges in the order that you define them	NOTSORTED

Required Arguments

name

names the format that you are creating.

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 and cannot end in a number. If you are creating a character format, then use a dollar sign ($) as the first character.

Restriction: The name of a user-defined format cannot be the same as the name of a format that is supplied by SAS.

Interaction: The maximum length of a format name is controlled by the VALIDFMTNAME= SAS system option. See *SAS Language Reference: Dictionary* for details about VALIDFMTNAME=.

Tip: Refer to the format later by using the name followed by a period. However, do not use a period after the format name in the VALUE statement.

Options

The following options are common to the INVALUE, PICTURE, and VALUE statements and are described in “Informat and Format Options” on page 451:

DEFAULT=*length*

FUZZ= *fuzz-factor*

MAX=*length*

MIN=*length*

NOTSORTED

In addition, you can use the following options:

MULTILABEL

allows 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 may 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. For 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.

value-range-set(s)

specifies one or more variable values and a character string or an existing format. The *value-range-set(s)* can be one or more of the following:

value-or-range-1 <..., value-or-range-n>='formatted-value' | [*existing-format*]

The variable values on the left side of the equals sign print as the character string on the right side of the equals sign.

formatted-value

specifies a character string that becomes the printed value of the variable value that appears on the left side of the equals 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.

Requirement: 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 "(pass)"
   F=Female "(pass)";
```

Requirement: 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 supplied by SAS or an existing user-defined format. The format you are creating uses the existing format to convert the raw data that match *value-or-range* on the left side of the equals sign.

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.*

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.

Tip: Avoid nesting formats more than one level. The resource requirements can increase dramatically with each additional level.

value-or-range

For details on how to specify *value-or-range*, see "Specifying Values or Ranges" on page 453.

Consider the following examples:

- 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. △

- The numeric format ANSWER.writes the values 1 and 2 as **yes** and **no**:

```
value answer 1='yes'
             2='no';
```

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.

Informat and Format Options

This section discusses options that are valid in the INVALUE, PICTURE, and VALUE statements. These options appear in parentheses after the informat or format name. They affect the entire informat or format that you are creating.

DEFAULT=*length*

specifies the default length of the informat or format. The value for DEFAULT= becomes the length of the informat or format if you do not give a specific length when you associate the informat or format with a variable.

The default length of a format is the length of the longest formatted value.

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 equals sign. If you have a quoted string to the left of the equals sign, then the default length is the length of the longest string.

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.

Tip: A value that is excluded from a range using the < operator does not receive the formatted value, even if it falls into the range when you use the fuzz factor.

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

- 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.

Note: 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:

1. Use the CNTLOUT= option in the PROC FORMAT statement to create an output control data set.
2. Use the CPORT procedure to create a transport file for the control data set.
3. Use the CIMPORT procedure in the target operating environment to import the transport file.
4. In the target operating environment, use PROC FORMAT with the CNTLIN= option to build the formats and informats from the imported control data set.

△

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 equals sign you specify the values that you want to convert to other values. On the right side of the equals 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 equals sign. For details about how to specify values for the right side of the equals 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 equals sign. INVALUE and VALUE also accept character strings on the left side of the equals 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.

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'**.

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, these 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 equals 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**. △

Concepts: FORMAT Procedure

Associating Informats and Formats with Variables

Methods of Associating Informats and Formats with Variables

Table 22.2 on page 455 summarizes the different methods for associating informats and formats with variables.

Table 22.2 Associating Informats and Formats with Variables

Step	Informats	Formats
In a DATA step	Use the ATTRIB or INFORMAT statement to permanently associate an informat with a variable. Use the INPUT function or INPUT statement to associate the informat with the variable only for the duration of the DATA step.	Use the ATTRIB or FORMAT statement to permanently associate a format with a variable. Use the PUT function or PUT statement to associate the format with the variable only for the duration of the DATA step.
In a PROC step	The ATTRIB and INFORMAT statements are valid in base SAS procedures. However, in base SAS software, typically you do not assign informats in PROC steps because the data has already been read into SAS variables.	Use the ATTRIB statement or the FORMAT statement to associate formats with variables. If you use either statement in a procedure that produces an output data set, then the format is permanently associated with the variable in the output data set. If you use either statement in a procedure that does not produce an output data set or modify an existing data set, the statement associates the format with the variable only for the duration of the PROC step.

Tips

- □ 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 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.

See Also

- For complete documentation on the ATTRIB, INFORMAT, and FORMAT statements, see the section on statements in *SAS Language Reference: Dictionary*.
- For complete documentation on the INPUT and PUT functions, see the section on functions in *SAS Language Reference: Dictionary*.
- See “Formatted Values” on page 25 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.* 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

- FORMAT for numeric formats
- FORMATC for character formats
- INFMT for numeric informats
- INFMTC for character informats.

Temporary Informats and Formats

Informats and formats are temporary when they are stored in a catalog in the WORK library. If you omit the LIBRARY= option, then PROC FORMAT stores the informats and formats in the temporary catalog WORK.FORMATS. You can retrieve temporary informats and formats only in the same SAS session or job in which they are created. To retrieve a temporary format or informat, simply include the name of the format or informat in the appropriate SAS statement. SAS automatically looks for the format or informat in the WORK.FORMATS catalog.

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 “PROC FORMAT Statement” on page 432.

---

* Catalogs are a type of SAS file and reside in a SAS data library. If you are unfamiliar with the types of SAS files or the SAS data library structure, then see the section on SAS files in *SAS Language Reference: Concepts*.

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 you should do the following:
  1. Assign the LIBRARY libref to a SAS data 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 will store 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 you should do the following:
  1. Assign a libref to a SAS data 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*>);

where 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 that they are listed until the format or informat is found.

For further information on FMTSEARCH=, see the section on SAS system options in *SAS Language Reference: Dictionary*. For an example that uses the LIBRARY= and FMTSEARCH= options together, see Example 8 on page 480.

Missing Formats and Informats

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;
```

Refer to the section on SAS system options in *SAS Language Reference: Dictionary* for more information on NOFMTERR.

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. Additionally, 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

DEFAULT
: a numeric variable that indicates the default length for format or informat

END
: a character variable that gives the range's ending value

EEXCL
: a character variable that indicates whether the range's ending value is excluded. Values are

Y	the range's ending value is excluded
N	the range's ending value is not excluded

FILL
: for picture formats, a numeric variable whose value is the value of the FILL= option

FMTNAME
: a character variable whose value is the format or informat name

FUZZ
: a numeric variable whose value is the value of the FUZZ= option

HLO
: a character variable that contains range information about the format or informat in the form of eight different letters that can appear in any combination. Values are

F	standard SAS format or informat used for formatted value or informatted value
H	range's ending value is HIGH
I	numeric informat range (informat defined with unquoted numeric range)

L	range's starting value is LOW
N	format or informat has no ranges, including no OTHER= range
O	range is OTHER
M	MULTILABEL option is in effect
R	ROUND option is in effect
S	NOTSORTED option is in effect

LABEL
a character variable whose value is the informatted or formatted value or the name of an existing informat or format

LENGTH
a numeric variable whose value is the value of the LENGTH= option

MAX
a numeric variable whose value is the value of the MAX= option

MIN
a numeric variable whose value is the value of the MIN= option

MULT
a numeric variable whose value is the value of the MULT= option

NOEDIT
for picture formats, a numeric variable whose value indicates whether the NOEDIT option is in effect. Values are

1	NOEDIT option is in effect
0	NOEDIT option is not in effect

PREFIX
for picture formats, a character variable whose value is the value of the PREFIX= option

SEXCL
a character variable that indicates whether the range's starting value is excluded. Values are

Y	the range's starting value is excluded
N	the range's starting value is not excluded

START
a character variable that gives the range's starting value

TYPE
a character variable that indicates the type of format. Possible values are

C	character format
I	numeric informat
J	character informat
N	numeric format (excluding pictures)
P	picture format

Output 22.1 shows an output control data set that contains information on all the informats and formats created in "Examples: FORMAT Procedure" on page 463.

Output 22.1 Output Control Data Set for PROC FORMAT Examples

```
                                            An Output Control Data Set                                              1

                                                                                                               D L
   F                                                                    D                                    D A A
   M                                                                    E  L        P           N          D I T N
   T              S                                   L                 F  E        R           O   S E    E G A G
   N              T                                   A                 A  N    F   E       M F E T E E    C 3 T U
 O A              A                E                  B           M  M  U  G    U   F       U I D Y X X H  S S Y A
 b M              R                N                  E           I  A  L  T    Z   I       L L I P C C L  E E P G
 s E              T                D                  L           N  X  T  H    Z   X       T L T E L L O  P P E E

 1 BENEFIT LOW                          7304 WORDDATE20.          1 40 20 20  1E-12      0.00   0 N N N LF
 2 BENEFIT             7305 HIGH               ** Not Eligible ** 1 40 20 20  1E-12      0.00   0 N N N H
 3 NOZEROS LOW                            -1 00.00                1 40  5  5  1E-12 -  100.00   0 P N N L  . ,
 4 NOZEROS               -1                0 99                   1 40  5  5  1E-12 -. 100.00   0 P Y Y    . ,
 5 NOZEROS                0                1 99                   1 40  5  5  1E-12 .  100.00   0 P N Y    . ,
 6 NOZEROS                1 HIGH             00.00                1 40  5  5  1E-12    100.00   0 P N N H  . ,
 7 PTSFRMT                0                3 0%                   1 40  3  3  1E-12      0.00   0 N N N
 8 PTSFRMT                4                6 3%                   1 40  3  3  1E-12      0.00   0 N N N
 9 PTSFRMT                7                8 6%                   1 40  3  3  1E-12      0.00   0 N N N
10 PTSFRMT                9               10 8%                   1 40  3  3  1E-12      0.00   0 N N N
11 PTSFRMT               11 HIGH             10%                  1 40  3  3  1E-12      0.00   0 N N N H
12 USCURR  LOW              HIGH             000,000              1 40  7  7  1E-12 $    1.61   0 P N N LH . ,
13 CITY    BR1              BR1              Birmingham UK        1 40 14 14      0      0.00   0 C N N
14 CITY    BR2              BR2              Plymouth UK          1 40 14 14      0      0.00   0 C N N
15 CITY    BR3              BR3              York UK              1 40 14 14      0      0.00   0 C N N
16 CITY    US1              US1              Denver USA           1 40 14 14      0      0.00   0 C N N
17 CITY    US2              US2              Miami USA            1 40 14 14      0      0.00   0 C N N
18 CITY    **OTHER**        **OTHER**        INCORRECT CODE       1 40 14 14      0      0.00   0 C N N O
19 EVAL    C                C                                   1 1 40  1  1      0      0.00   0 I N N
20 EVAL    E                E                                   2 1 40  1  1      0      0.00   0 I N N
21 EVAL    N                N                                   0 1 40  1  1      0      0.00   0 I N N
22 EVAL    O                O                                   4 1 40  1  1      0      0.00   0 I N N
23 EVAL    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 447 and "EXCLUDE Statement" on page 434.

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 458. 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 5 on page 472.

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 447 and "EXCLUDE Statement" on page 434. For an example, see Example 6 on page 477.

The FMTLIB output shown in Output 22.2 contains a description of the NOZEROS. format, which is created in "Building a Picture Format: Step by Step" on page 443, and the EVAL. informat, which is created in Example 4 on page 470.

Output 22.2 Output from PROC FORMAT with the FMTLIB Option

```
                FMTLIB Output for the NOZEROS. Format and the             1
                                EVAL. Informat

  ----------------------------------------------------------------------------
  |       FORMAT NAME: NOZEROS  LENGTH:    5   NUMBER OF VALUES:    4         |
  |   MIN LENGTH:   1  MAX LENGTH:  40  DEFAULT LENGTH   5  FUZZ: STD         |
  |--------------------------------------------------------------------------|
  |START           |END             |LABEL  (VER. 7.00    29MAY98:10:00:24)  |
  |----------------+----------------+----------------------------------------|
  |LOW             |              -1|00.00               P-  F  M100         |
  |              -1<               0<99                  P-. F  M100         |
  |               0|               1<99                  P.  F  M100         |
  |               1|HIGH            |00.00               P   F  M100         |
  ----------------------------------------------------------------------------


  ----------------------------------------------------------------------------
  |     INFORMAT NAME: @EVAL    LENGTH:    1   NUMBER OF VALUES:    5         |
  |   MIN LENGTH:   1  MAX LENGTH:  40  DEFAULT LENGTH   1  FUZZ:        0    |
  |--------------------------------------------------------------------------|
  |START           |END             |INVALUE(VER. 7.00    29MAY98:10:00:25)  |
  |----------------+----------------+----------------------------------------|
  |C               |C               |                                       1|
  |E               |E               |                                       2|
  |N               |N               |                                       0|
  |O               |O               |                                       4|
  |S               |S               |                                       3|
  ----------------------------------------------------------------------------
```

The fields are described below in the order they appear in the output, from left to right:

INFORMAT NAME
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 equals sign.
- For numeric informats
  - LENGTH is 12 if all values on the left side of the equals 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 @EVAL., the length is 1 because 1 is the length of the longest raw data value on the left side of the equals sign.

In the output for NOZEROS., the LENGTH is 5 because the longest picture is 5 characters.

NUMBER OF VALUES
the number of values or ranges associated with the informat or format. NOZEROS. has 4 ranges, 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
LABEL
INVALUE appears only for informats and contains the informatted values. LABEL appears only for formats and contains either the formatted value or picture. The SAS release number and the date on which the format or informat was created are in parentheses after INVALUE or LABEL.

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 dash followed by a period.

FMTLIB prints only 40 characters in the LABEL column.

Examples: FORMAT Procedure

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 458 contains a description of these informats and the formats.

```
libname proclib 'SAS-data-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;
   input Name & $16. IdNumber $ Salary
```

```
          Site $ HireDate date7.;
    format hiredate date7.;
    datalines;
Capalleti, Jimmy  2355 21163 BR1 30JAN79
Chen, Len         5889 20976 BR1 18JUN76
Davis, Brad       3878 19571 BR2 20MAR84
Leung, Brenda     4409 34321 BR2 18SEP74
Martinez, Maria   3985 49056 US2 10JAN93
Orfali, Philip    0740 50092 US2 16FEB83
Patel, Mary       2398 35182 BR3 02FEB90
Smith, Robert     5162 40100 BR5 15APR86
Sorrell, Joseph   4421 38760 US1 19JUN93
Zook, Carla       7385 22988 BR3 18DEC91
;
```

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.

Example 1: Creating a Picture Format

Procedure features:

PROC FORMAT statement options:

- LIBRARY=

PICTURE statement options:

- MULT=
- PREFIX=

LIBRARY libref

LOW and HIGH keywords

Data set:

PROCLIB.STAFF on page 463.

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

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-data-library-1 ';
libname library 'SAS-data-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

```
              PROCLIB.STAFF with a Format for the Variable Salary            1

                                      Salary in
                         Id             U.S.                      Hire
      Name             Number         Dollars      Site           Date

      Capalleti, Jimmy   2355         $34,072      BR1          30JAN79
      Chen, Len          5889         $33,771      BR1          18JUN76
      Davis, Brad        3878         $31,509      BR2          20MAR84
      Leung, Brenda      4409         $55,256      BR2          18SEP74
      Martinez, Maria    3985         $78,980      US2          10JAN93
      Orfali, Philip     0740         $80,648      US2          16FEB83
      Patel, Mary        2398         $56,643      BR3          02FEB90
      Smith, Robert      5162         $64,561      BR5          15APR86
      Sorrell, Joseph    4421         $62,403      US1          19JUN93
      Zook, Carla        7385         $37,010      BR3          18DEC91
```

Example 2: Creating a Format for Character Values

Procedure features:
VALUE statement
OTHER keyword

Data set:
PROCLIB.STAFF on page 463.

Format: USCurrency. on page 465

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

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-data-library-1';
libname library 'SAS-data-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;
```

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 (created in Example 1 on page 464) 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

```
              PROCLIB.STAFF with a Format for the Variables              1
                             Salary and Site

                                 Salary in
                       Id          U.S.                               Hire
Name                 Number      Dollars     Site                     Date

Capalleti, Jimmy      2355       $34,072     Birmingham UK     30JAN79
Chen, Len             5889       $33,771     Birmingham UK     18JUN76
Davis, Brad           3878       $31,509     Plymouth UK       20MAR84
Leung, Brenda         4409       $55,256     Plymouth UK       18SEP74
Martinez, Maria       3985       $78,980     Miami USA         10JAN93
Orfali, Philip        0740       $80,648     Miami USA         16FEB83
Patel, Mary           2398       $56,643     York UK           02FEB90
Smith, Robert         5162       $64,561     INCORRECT CODE    15APR86
Sorrell, Joseph       4421       $62,403     Denver USA        19JUN93
Zook, Carla           7385       $37,010     York UK           18DEC91
```

Example 3: Writing a Format for Dates Using a Standard SAS Format

Procedure features:

VALUE statement:

HIGH keyword

Data set:

PROCLIB.STAFF on page 463.

Formats:

USCurrency. on page 465 and $CITY. on page 467.

This example uses an existing format that is supplied by SAS as a formatted value. Tasks include

- creating a numeric format
- nesting formats
- writing a format using a standard SAS format
- formatting dates.

Program

This program defines a format called BENEFIT, which differentiates between employees hired on or before 31DEC1979. The purpose of this program is to indicate any employees who are eligible to receive a benefit, based on a hire date on or prior to December 31, 1979. 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-data-library-1';
libname library 'SAS-data-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;
```

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 31DEC1979 and those who were hired after that date. The keyword LOW and the SAS date constant '31DEC1979'D create the first range, which includes all date values that occur on or before December 31, 1979. For values that fall into this range, SAS applies the WORDDATE*w*. format.*

```
      value benefit  low-'31DEC1979'd=[worddate20.]
```

Define the second range in the BENEFIT. format. The second range consists of all dates on or after January 1, 1980. The SAS date constant '01JAN1980'D and the keyword HIGH specify the range. Values that fall into this range receive `** Not Eligible **` as a formatted value.

```
                     '01JAN1980'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;
```

* For more information about SAS date constants, see the section on dates, times, and intervals in *SAS Language Reference: Concepts*. For complete documentation on WORDDATE*w*., see the section on formats in *SAS Language Reference: Dictionary*.

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 1 on page 464) with SALARY, the $CITY. format (created in Example 2 on page 466) 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

```
                   PROCLIB.STAFF with a Format for the Variables                1
                             Salary, Site, and HireDate

                                  Salary in
                         Id         U.S.
Name                   Number     Dollars     Site                    HireDate

Capalleti, Jimmy        2355      $34,072     Birmingham UK       January 30, 1979
Chen, Len               5889      $33,771     Birmingham UK          June 18, 1976
Davis, Brad             3878      $31,509     Plymouth UK       ** Not Eligible **
Leung, Brenda           4409      $55,256     Plymouth UK       September 18, 1974
Martinez, Maria         3985      $78,980     Miami USA         ** Not Eligible **
Orfali, Philip          0740      $80,648     Miami USA         ** Not Eligible **
Patel, Mary             2398      $56,643     York UK           ** Not Eligible **
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 4: Converting Raw Character Data to Numeric Values

Procedure feature:
INVALUE statement

This example uses an INVALUE statement to create a numeric informat that converts numeric and character raw data to numeric data.

Program

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-data-library-1';
libname library 'SAS-data-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=64 pagesize=40;
```

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

```
                The PROCLIB.POINTS Data Set                1

          Employee                                 Total
             Id       Q1     Q2     Q3     Q4     Points

           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 5: Creating a Format from a Data Set

Procedure features:

PROC FORMAT statement option:

CNTLIN=

Input control data set

Data set:

WORK.POINTS, created from data lines in the sample code.

This example shows how to create a format from a SAS data set.
Tasks include

- □ creating a format from an input control data set
- □ creating an input control data set from an existing SAS data set.

Program

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 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;
```

Note that although the last observation contains values for START and END, these values are ignored because of the **O** value in the HLO variable.

```
                         The CTRL Data Set                               1

     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;
   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. See Chapter 42, "The REPORT Procedure," on page 845 for more information about PROC REPORT.

```
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 22.3

```
              The Percentage of Salary for Calculating Bonus              1

                   Employee      Total
                         Id     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 6: Printing the Description of Informats and Formats

Procedure features:
PROC FORMAT statement option:
FMTLIB
SELECT statement

Format:
NOZEROS on page 444.

Informat:
Evaluation. on page 471

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

Set up a SAS library reference named LIBRARY.

```
libname library 'SAS-data-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=60;
```

Print a description of Evaluation. and NOZEROS. 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 EVAL and NOZEROS, which were created in previous examples. The at sign (@) in front of EVAL indicates that EVAL. is an informat.

```
   select @evaluation nozeros;
```

Specify the titles.

```
   title 'FMTLIB Output for the NOZEROS. Format and the';
   title2 'Evaluation. Informat';
run;
```

Output

The output is described in "Procedure Output" on page 461.

```
                 FMTLIB Output for the NOZEROS. Format and the                1
                              Evaluation. Informat

   ----------------------------------------------------------------------------
   |       FORMAT NAME: NOZEROS  LENGTH:    5   NUMBER OF VALUES:    4        |
   |   MIN LENGTH:   1  MAX LENGTH:  40  DEFAULT LENGTH   5  FUZZ: STD        |
   |--------------------------------------------------------------------------|
   |START           |END             |LABEL  (VER. V7|V8   10APR2002:18:55:08)|
   |----------------+----------------+----------------------------------------|
   |LOW             |              -1|00.00                P-  F  M100        |
   |              -1<               0<99                   P-. F  M100        |
   |               0|               1<99                   P.  F  M100        |
   |               1|HIGH            |00.00                P   F  M100        |
   ----------------------------------------------------------------------------


   ----------------------------------------------------------------------------
   |                   INFORMAT NAME: @EVALUATION LENGTH: 1                    |
   |   MIN LENGTH:   1  MAX LENGTH:  40  DEFAULT LENGTH   1  FUZZ:        0   |
   |--------------------------------------------------------------------------|
   |START           |END             |INVALUE(VER. 9.00    10APR2002:18:55:11)|
   |----------------+----------------+----------------------------------------|
   |C               |C               |                                       1|
   |E               |E               |                                       2|
   |N               |N               |                                       0|
   |O               |O               |                                       4|
   |S               |S               |                                       3|
   ----------------------------------------------------------------------------
```

Example 7: Retrieving a Permanent Format

Procedure features:
PROC FORMAT statement options:
LIBRARY=

Other features:
FMTSEARCH= system option

Data sets:
SAMPLE on page 443.

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

Set up a SAS library reference named PROCLIB.

```
libname proclib 'SAS-data-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=64 pagesize=60;
```

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 "Building a Picture Format: Step by Step" on page 443.

```
   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.*

```
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.;
```

* For complete documentation on the FMTSEARCH= system option, see the section on SAS system options in *SAS Language Reference: Dictionary*.

Specify the titles.

```
   title1 'Retrieving the NOZEROS. Format from PROCLIB.FORMATS';
   title2 'The SAMPLE Data Set';
run;
```

Output

```
       Retrieving the NOZEROS. Format from PROCLIB.FORMATS      1
                      The SAMPLE Data Set

                          Obs    Amount

                           1     -2.05
                           2      -.05
                           3      -.01
                           4       .00
                           5       .09
                           6       .54
                           7       .55
                           8      6.60
                           9     14.63
```

Example 8: Writing Ranges for Character Strings

Data sets:
PROCLIB.STAFF on page 463.

This example creates a format and shows how to use ranges with character strings.

Program

```
libname proclib 'SAS-data-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 TRAIN data set from the PROCLIB.STAFF data set. PROCLIB.STAFF was created in "Examples: FORMAT Procedure" on page 463.

```
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;
```

```
                    The TRAIN Data Set without a Format                    1

                                           Id
                         Name            Number

                         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 Chapter 42, "The REPORT Procedure," on page 845for 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

```
                        Test Assignment for Each Employee                     1

                        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 9: Filling a Picture Format

Procedure features:

PICTURE statement options:

FILL=
PREFIX=

This example

- 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

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 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

```
                 Printing Salaries for a Check                 1

                     Name     MonthlySalary

                     Liu      ****$1,259.45
                     Lars     ****$1,289.33
                     Kim      ****$1,439.02
                     Wendy    ****$1,675.21
                     Alex     ****$1,623.73
```

See Also

FMTSEARCH= System option
VALIDFMTNAME= System option
FORMAT Statement

CHAPTER
23

The FORMS Procedure

Information about the FORMS Procedure

See: For documentation of the FORMS procedure, go to **`http://support.sas.com/documentation/onlinedoc`**. Select Base SAS from the Product-Specific Documentation list.

CHAPTER

24

The FREQ Procedure

Information about the FREQ Procedure

See: The documentation for the FREQ procedure has moved to Volume 3 of this book.

CHAPTER
25

The FSLIST Procedure

Overview: FSLIST Procedure

The FSLIST procedure enables you to browse external files that are not SAS data sets within a SAS session. 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* <*option(s)*>;

- You must specify either the FILEREF= or the UNIT= argument with the PROC FSLIST statement.
- *Option(s)* can be one or more of the following:
  CAPS | NOCAPS
  CC | FORTCC | NOCC
  HSCROLL=HALF | PAGE | *n*
  NOBORDER

NUM | NONUM
OVP | NOOVP

Statement Descriptions

The only statement that the FSLIST procedure supports is the PROC FSLIST statement, which starts the procedure.

Requirements

You must specify an external file for PROC FSLIST to browse.

FSLIST Command

The FSLIST procedure can also be initiated by entering the following command on the command line of any SAS window:

FSLIST <* | ? | *file-specification* <*carriage-control-option* <*overprinting-option*>>>

where *carriage-control-option* can be CC, FORTCC, or NOCC and *overprinting-option* can be OVP or NOOVP.

Note: OVP is ignored if NOCC is in effect. △

PROC FSLIST Statement

The PROC FSLIST statement initiates the FSLIST procedure and specifies the external file to browse. Statement options enable you to modify the default behavior of the procedure.

PROC FSLIST Statement Requirements

The PROC FSLIST statement must include one of the following arguments that specifies the external file to browse.

FILEREF=*file-specification*
DDNAME=*file-specification*
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 Language Reference: Dictionary*.

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

```
proc fslist unit=20;
```

instead of

```
proc fslist fileref=ft20f001;
```

PROC FSLIST Statement Options

The following options can be used with the PROC FSLIST statement:

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. △

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 honored:

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.

The default is NOOVP. The OVP option is ignored if the NOCC option is in effect.

FSLIST Command

The FSLIST command provides a handy way to initiate 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.

FSLIST Command Syntax

The general form of the FSLIST command is

FSLIST <*|?| *file-specification* <*carriage-control-option* <*overprinting-option*>>>

where *carriage-control-option* can be CC, FORTCC, or NOCC and *overprinting-option* can be OVP or NOOVP.

Note: OVP is ignored if NOCC is in effect. △

FSLIST Command Arguments

You can specify one of the following arguments with the FSLIST command:

*****

opens a dialog window in which you can specify the name of the file to browse, along with various FSLIST procedure options. In the dialog window, 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).

Note: 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. △

To select a file, position the cursor on the corresponding fileref and press ENTER.

Note: 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 opens 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 opens that shows all available filerefs. An error message in the selection window indicates that the specified fileref is not defined.

If you do not specify any of these three arguments, then a selection window opens that enables you to select an external filename.

FSLIST Command Options

If you use a *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. 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 on page 491 for details.

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 is the default).

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

General Information about 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, this text 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 *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 on 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 "FSLIST Command Options" on page 493 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
26

The IMPORT Procedure

Overview: IMPORT Procedure

The IMPORT procedure reads data from an external data source and writes it to a SAS data set. External data sources can include Microsoft Access Database, Excel files, Lotus spreadsheets, and delimited external files (in which columns of data values are separated by a delimiter such as a blank, comma, or tab).

When you execute PROC IMPORT, the procedure reads the input file and writes the data to a SAS data set. The SAS variable definitions are based on the input records. PROC IMPORT imports the data by one of the following methods:

- generated DATA step code
- generated SAS/ACCESS code
- translation engines.

You control the results with statements and options that are specific to the input data source. PROC IMPORT generates the specified output SAS data set and writes information regarding the import to the SAS log. In the log, you see the DATA step or the SAS/ACCESS code that is generated by PROC IMPORT. If a translation engine is used, then no code is submitted.

Note: To import data, you can also use the Import Wizard, which is a windowing tool that guides you through the steps to import an external data source. You can request the Import Wizard to generate IMPORT procedure statements, which you can save to a file for subsequent use. To invoke the Import Wizard, from the SAS windowing environment select

File ▶ Import Data

△

Syntax: IMPORT Procedure

Restriction: PROC IMPORT is available for the following operating environments:

- OpenVMS Alpha
- UNIX
- Microsoft Windows.

PROC IMPORT
DATAFILE="*filename*" | TABLE="*tablename*"
OUT=<*libref.*>*SAS-data-set* <(*SAS-data-set-options*)>
<DBMS=*identifier*><REPLACE> ;

<*data-source-statement(s)*;>

PROC IMPORT Statement

Featured in: All examples

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 a fileref for the input PC file, spreadsheet, or delimited external file. 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, you can omit the quotation marks. A fileref is a SAS name that is associated with the physical location of the output file. To assign a fileref, use the FILENAME statement. For more information about PC file formats, see *SAS/ACCESS for PC Files: Reference*.

Featured in: Example 1 on page 514, Example 2 on page 517, and Example 3 on page 518

Restriction: PROC IMPORT does not support device types or access methods for the FILENAME statement except for DISK. For example, PROC IMPORT does not support the TEMP device type, which creates a temporary external file.

Restriction: For client/server applications: When running SAS/ACCESS software on UNIX to access data that is stored on a PC server, you must specify the full path and filename of the file that you want to import. The use of a fileref is not supported.

Interaction: For some input data sources like a Microsoft Excel spreadsheet, in order to determine the data type (numeric or character) for a column, the first eight rows of data are scanned and the most prevalent type of data is used. If most of the data in the first eight rows is missing, SAS defaults to the character data type; any subsequent numeric data for that column becomes missing as well. Mixed data can also create missing values. For example, if the first eight rows contain mostly character data, SAS assigns the column as a character data type; any subsequent numeric data for that column becomes missing.

Restriction: PROC IMPORT can import data only if the data type is supported by SAS. 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 not supported by SAS, PROC IMPORT may not be able to import it correctly. In many cases, the procedure attempts to convert the data to the best of its ability; however, for some types, this is not possible.

Tip: For information about how SAS converts data types, see the specific information for the data source that you are importing in *SAS/ACCESS for PC Files: Reference*. For example, see the chapter "Understanding XLS Essentials" for a table that lists XLS data types and the resulting SAS variable data type and formats.

Tip: For a DBF file, if the file was created by Microsoft Visual FoxPro, the file must be exported by Visual FoxPro into an appropriate dBASE format in order to import the file to SAS.

TABLE="*tablename*"

specifies the table 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 may be case sensitive.

Requirement: When you import a DBMS table, you must specify the DBMS= option.

Featured in: Example 4 on page 519

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, PROC IMPORT creates it. If you specify a one-level name, by default PROC IMPORT uses either the USER library (if assigned) or the WORK library (if USER not assigned).

Featured in: All examples

(*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, or to import only data that meets a specified condition, you can use the WHERE= data set option. For information about all SAS data set options, see "Data Set Options" in *SAS Language Reference: Dictionary*.

Restriction: You cannot specify data set options when importing delimited, comma-separated, or tab-delimited external files.

Featured in: Example 3 on page 518

Options

DBMS=*identifier*

specifies the type of data to import. To import a DBMS table, you must specify DBMS= using a valid database identifier. For example, DBMS=ACCESS specifies to import a Microsoft Access 2000 or 2002 table. To import PC files, spreadsheets, and delimited external files, you do not have to specify DBMS= if the filename that is specified by DATAFILE= contains a valid extension so that PROC IMPORT can recognize the type of data. For example, PROC IMPORT recognizes the filename ACCOUNTS.WK1 as a Lotus 1-2-3 Release 2 spreadsheet and the filename MYDATA.CSV as a delimited external file that contains comma-separated data values; therefore, a DBMS= specification is not necessary.

The following values are valid for the DBMS= option:

Identifier	**Input Data Source**	**Extension**	**Host Availability**
ACCESS	Microsoft Access 2000 or 2002 table	.mdb	Microsoft Windows *
ACCESS97	Microsoft Access 97 table	.mdb	Microsoft Windows *
ACCESS2000	Microsoft Access 2000 table	.mdb	Microsoft Windows *
ACCESS2002	Microsoft Access 2002 table	.mdb	Microsoft Windows *
ACCESSCS	Microsoft Access table	.mdb	UNIX
CSV	delimited file (comma-separated values)	.csv	OpenVMS Alpha, UNIX, Microsoft Windows
DBF	dBASE 5.0, IV, III+, and III files	.dbf	UNIX, Microsoft Windows
DLM	delimited file (default delimiter is a blank)	.*	OpenVMS Alpha, UNIX, Microsoft Windows
EXCEL	Excel 2000 or 2002 spreadsheet	.xls	Microsoft Windows *
EXCEL4	Excel 4.0 spreadsheet	.xls	Microsoft Windows
EXCEL5	Excel 5.0 or 7.0 (95) spreadsheet	.xls	Microsoft Windows
EXCEL97	Excel 97 or 7.0 (95) spreadsheet	.xls	Microsoft Windows *
EXCEL2000	Excel 2000 spreadsheet	.xls	Microsoft Windows *
EXCELCS	Excel spreadsheet	.xls	UNIX
JMP	JMP table	.jmp	UNIX, Microsoft Windows
PCFS	Files on PC server	.*	UNIX

Identifier	Input Data Source	Extension	Host Availability
TAB	delimited file (tab-delimited values)	.txt	OpenVMS Alpha, UNIX, Microsoft Windows
WK1	Lotus 1-2-3 Release 2 spreadsheet	.wk1	Microsoft Windows
WK3	Lotus 1-2-3 Release 3 spreadsheet	.wk3	Microsoft Windows
WK4	Lotus 1-2-3 Release 4 or 5 spreadsheet	.wk4	Microsoft Windows

* Not available for Microsoft Windows 64-Bit Edition.

Restriction: The availability of an input data source depends on

- the operating environment, and in some cases the platform, as specified in the previous table.
- whether your site has a license to the SAS/ACCESS software for PC file formats. If you do not have a license, only delimited files are supported.

Featured in: Example 1 on page 514 and Example 4 on page 519

When you specify a value for DBMS=, consider the following:

- To import a Microsoft Access table, PROC IMPORT can distinguish whether the table is in Access 97, 2000, or 2002 format regardless of your specification. For example, if you specify DBMS=ACCESS and the table is an Access 97 table, PROC IMPORT will import the file.
- To import a Microsoft Excel spreadsheet, PROC IMPORT can distinguish some versions regardless of your specification. For example, if you specify DBMS=EXCEL and the spreadsheet is an Excel 97 spreadsheet, PROC IMPORT can import the file. However, if you specify DBMS=EXCEL4 and the spreadsheet is an Excel 2000 spreadsheet, PROC IMPORT cannot import the file. The following table lists the spreadsheets and whether PROC IMPORT can distinguish them based on the DBMS= specification:

Specification	Excel 2002	Excel 2000	Excel 97	Excel 5.0	Excel 4.0
EXCEL	yes	yes	yes	yes	yes
EXCEL2002	yes	yes	yes	yes	yes
EXCEL2000	yes	yes	yes	yes	yes
EXCEL97	yes	yes	yes	yes	yes
EXCEL5	no	no	no	yes	yes
EXCEL4	no	no	no	yes	yes

Note: Although Excel 4.0 and Excel 5.0 spreadsheets are often interchangeable, it is recommended that you specify the exact version. △

REPLACE
overwrites an existing SAS data set. If you do not specify REPLACE, PROC IMPORT does not overwrite an existing data set.

Featured in: Example 1 on page 514

Data Source Statements

Featured in: All examples

PROC IMPORT provides a variety of statements that are specific to the input data source.

Statements for PC Files, Spreadsheets, or Delimited External Files

The following table lists the statements that are available to import PC files, spreadsheets, and delimited external files, and it denotes which statements are valid for a specific data source. For example, Excel spreadsheets have optional statements to indicate whether column names are in the first row of data or which sheet and range of data to import, while a dBASE file (DBF) does not. For more information about PC file formats, see *SAS/ACCESS for PC Files: Reference*.

Data Source	Supported Syntax	Valid Values	Default Value
CSV/TAB	GETNAMES=	YES | NO	YES
	DATAROW=	1 to 32767	2
	GUESSING ROWS=	1 to 32767	none
DLM	GETNAMES=	YES | NO	YES
	DATAROW=	1 to 32767	2
	GUESSINGROWS=	1 to 32767	none
JMP			
DBF	GETDELETED=	YES | NO	NO
WK1 / WK3 / WK4	GETNAMES=	YES | NO	YES
	RANGE=	Range Name or Absolute Range Value, such as ’A1...C4’	
	SHEET=	Sheet Name	
EXCEL4 / EXCEL5	GETNAMES=	YES | NO	YES
	RANGE=	Range Name or Absolute Range Value, such as ’A1...C4’	
	SHEET=	Sheet Name	

Data Source	Supported Syntax	Valid Values	Default Value
EXCEL EXCEL97 EXCEL2000 EXCEL2002	GETNAMES=	YES | NO	YES
	RANGE=	Range Name or Absolute Range Value, such as 'A1...C4'	
	SHEET=	Sheet Name	
	MIXED=	YES | NO	NO
	SCANTEXT=	YES | NO	YES
	SCANTIME=	YES | NO	YES
	USEDATE=	YES | NO	YES
	TEXTSIZE=	1 to 32767	1024
	DBSASLABEL=	COMPAT | NONE	COMPAT
EXCELCS	VERSION=	'5' | '95' | '97' | '2000' | '2002'	97
	SERVER=	Server Name	
	SERVICE=	Service Name	
	PORT=	1 to 32767	
	RANGE=	Range Name or Absolute Range Value, such as 'A1...C4'	
	SHEET=	Sheet Name	
	SCANTEXT=	YES | NO	YES
	SCANTIME=	YES | NO	YES
	USEDATE=	YES | NO	YES
	TEXTSIZE=	1 to 32767	1024
	DBSASLABEL=	COMPAT | NONE	COMPAT

DATAROW=*n*;
starts reading data from row number *n* in the external file.

Default:

1	when GETNAMES=NO
2	when GETNAMES=YES (default for GETNAMES=)

Interaction: When GETNAMES=YES, DATAROW= must be equal to or greater than 2. When GETNAMES=NO, DATAROW must be equal to or greater than 1.

DBSASLABEL=COMPAT | NONE;
When DBSASLABEL=COMPAT, the data source's column names are saved as the corresponding SAS label names. This is the default value.

WHEN DBSASLABEL=NONE, the data source's column names are not saved as SAS label names. SAS label names are left as nulls.

Featured in: Example 1 on page 514

DELIMITER='*char*' | '*nn*'*x*;
for a delimited external file, specifies the delimiter that separates columns of data in the input file. You can specify the delimiter as a single character or as a hexadecimal value. For example, if columns of data are separated by an

ampersand, specify DELIMITER='&'. If you do not specify DELIMITER=, PROC IMPORT assumes that the delimiter is the blank. You can replace the equal sign with a blank.

GETDELETED=YES | NO;
: for a dBASE file (DBF), indicates whether to write records to the SAS data set that are marked for deletion but have not been purged. You can replace the equal sign with a blank.

GETNAMES=YES | NO;
: for spreadsheets and delimited external files, determines whether to generate SAS variable names from the column names in the input file's first row of data. You can replace the equal sign with a blank.

If you specify GETNAMES=NO or if the column names are not valid SAS names, PROC IMPORT uses default variable names. For example, for a delimited file, PROC IMPORT uses VAR1, VAR2, VAR3, and so on.

Note that if a column name contains special characters that are not valid in a SAS name, such as a blank, SAS converts the character to an underscore. For example, the column name **`Occupancy Code`** would become the variable name **`Occupancy_Code`**.

GUESSING ROWS=1 to 3276;
: scans data for its data type from row 1 to the row number that is specified.

Note: This number should be greater than the value that is specified for DATAROW=. △

MIXED=YES | NO;
: converts numeric data values into character data values for a column that contains mixed data types. This option is valid only while importing data from Excel. The default is NO, which means that numeric data will be imported as missing values in a character column. If MIXED=YES, then the engine will assign a SAS character type for the column and convert all numeric data values to character data values. This option is valid only while reading (importing) data into SAS.

PORT=1 to 3276;
: scans data for its data type from row 1 to the row number that is specified.

Note: This number should be greater than the value that is specified for DATAROW=. △

TEXTSIZE=1 to 32767
: specifies the field length that is allowed for importing Microsoft Excel 97, 2000, or 2002 Memo fields.

RANGE="*range-name* | *absolute-range*";
: subsets a spreadsheet by identifying the rectangular set of cells to import from the specified spreadsheet. The syntax for *range-name* and *absolute-range* is native to the file being read. You can replace the equal sign with a blank.

range-name
: is a name that has been assigned to represent a range, such as a range of cells within the spreadsheet.

Limitation: SAS supports range names up to 32 characters. If a range name exceeds 32 characters, SAS will notify you that the name is invalid.

Tip: For Microsoft Excel, range names do not contain special characters such as spaces or hyphens.

absolute-range identifies the top left cell that begins the range and the bottom right cell that ends the range. For Excel 4.0, 5.0, and 7.0 (95), the beginning and ending cells are separated by two periods; that is, **`C9..F12`** specifies a cell range that begins at cell C9, ends at cell F12, and includes all the cells in between. For Excel 97, 2000, and 2002, the beginning and ending cells are separated by a colon – that is, **`C9:F12`**.

Tip: For Excel 97, 2000, and 2002, you can include the spreadsheet name with an absolute range, such as **`range="North B$a1:d3"`**. If you do not include the spreadsheet name, PROC IMPORT uses the first sheet in the workbook or the spreadsheet name specified with SHEET=.

Default: The entire spreadsheet is selected.

Interaction: For Excel 97, 2000, and 2002 spreadsheets, when RANGE= is specified, a spreadsheet name specified with SHEET= is ignored when the conflict occurs.

SCANTEXT=YES | NO;
scans the length of text data for a data source column and uses the length of the longest string data that it finds as the SAS column width. However, if the maximum length that it finds is greater than what is specified in the TEXTSIZE= option, then the smaller value that is specified in TEXTSIZE= will be applied as the SAS variable width.

SCANTIME=YES | NO;
scans all row values for a DATETIME data type field and automatically determines the TIME data type if only time values (that is, no date or datetime values) exist in the column.

SERVER="*PC-server-name*";
specifies the name of the PC server. You must bring up the listener on the PC server before you can establish a connection to it. You can configure the service name, port number, maximum number of connections allowed, and use of data encryption on your PC server. This is a required statement. Refer to your PC server administrator for the information that is needed. Alias: SERVER_NAME=.

SERVICE="*service-name*";
specifies the service name that is defined on your service file for your client and server machines. This statement and the PORT= statement should not be used in the same procedure. Note that this service name must be defined on both your UNIX machine and your PC server. Alias: SERVER_NAME=, SERVICE_NAME=.

SHEET=*spreadsheet-name*;
identifies a particular spreadsheet in a group of spreadsheets. Use this statement with spreadsheets that support multiple spreadsheets within a single file. The naming convention for the spreadsheet name is native to the file being read.

Featured in: Example 2 on page 517

Default: The default depends on the type of spreadsheet. For Excel 4.0 and 5.0, PROC IMPORT reads the first spreadsheet in the file. For Excel 97 and later, PROC IMPORT reads the first spreadsheet from an ascending sort of the spreadsheet names. To be certain that PROC IMPORT reads the desired spreadsheet, you should identify the spreadsheet by specifying SHEET=.

Limitation: SAS supports spreadsheet names up to 31 characters. With the $ appended, the maximum length of a spreadsheet name is 32 characters.

USEDATE=YES | NO;

If USEDATE=YES, then DATE. format is used for date/time columns in the data source table while importing data from Excel workbook. If USEDATE=NO, then DATETIME. format is used for date/time.

VERSION="*file-version*";

specifies the version of file that you want to create with if the file does not exist on your PC server yet. The default version is data-source specific. For Microsoft Excel workbook, the valid values are '2002', '2000', '97', '95' and '5', and its default value is '97'.

Note: Always quote the version value. △

Note: If the file already exists in the PC server, then this value can be ignored. △

Statements for DBMS Tables

The following data source statements are available to establish a connection to the DBMS when you import a DBMS table.

Data Source	Supported Syntax	Valid Values	Default Value
ACCESS ACCESS97 ACCESS2000 ACCESS2002	DATABASE=	The complete path and filename for the MS ACCESS database file.	
	DBPWD=	Database password	
	UID=	User ID	
	PWD=	User password	
	WGDB=	The complete path and filename for the Workgroup Administration file.	
	SCANMEMO=	YES | NO	YES
	SCANTIME=	YES | NO	YES
	USEDATE=	YES | NO	NO
	MEMOSIZE=	1 to 32767	1024
	DBSASLABEL=	COMPAT | NONE	COMPAT
ACCESSCS	VERSION=	'97' | '2000' | '2002'	'2000'
	SERVER=	Server Name	
	SERVICE=	Service Name	
	PORT=	1 to 32767	
	DATABASE=	The complete path and filename for the MS ACCESS database file.	
	DBPWD=	Database password	
	UID=	User ID	
	PWD=	User password	
	WGDB=	The complete path and file name for the Workgroup Administration file.	
	SCANMEMO=	YES | NO	YES
	SCANTIME=	YES | NO	YES
	USEDATE=	YES | NO	YES
	MEMOSIZE=	1 to 32767	1024
	DBSASLABEL=	COMPAT | NONE	COMPAT

DATABASE="*database*";
: specifies the complete path and filename of the database that contains the specified DBMS table. If the database name does not contain lowercase characters, special characters, or national characters ($, #, or @), you can omit the quotation marks. You can replace the equal sign with a blank.

Note: A default may be configured in the DBMS client software; however, SAS does not generate a default value. △

DBPWD="*database password*";
specifies a password that allows access to a database. You can replace the equal sign with a blank.

DBSASLABEL=COMPAT | NONE;
When DBSASLABEL=COMPAT, the data source's column names are saved as the corresponding SAS label names. This is the default value.

WHEN DBSASLABEL=NONE, the data source's column names are not saved as SAS label names. SAS label names are left as nulls.

Featured in: Example 1 on page 514

MEMOSIZE="*field-length*";
specifies the field length for importing Microsoft Access Memo fields.

Range:

Default:

Tip:

1 to 32,767

1024

To prevent Memo
fields from being
imported, you can
specify
MEMOSIZE=0

Range: 1 - 32,767 Default: 1024 Tip: To prevent Memo fields from being imported, you can specify MEMOSIZE=0.

PORT=1 to 3276;
scans data for its data type from row 1 to the row number that is specified.

Note: This number should be greater than the value that is specified for DATAROW=. △

PWD="*password*";
specifies the user password used by the DBMS to validate a specific userid. If the password does not contain lowercase characters, special characters, or national characters, you can omit the quotation marks. You can replace the equal sign with a blank.

Note: The DBMS client software may default to the userid and password that were used to log in to the operating environment; SAS does not generate a default value. △

SCANMEMO=YES | NO;
scans the length of data for memo fields and uses the length of the longest string data that it finds as the SAS column width. However, if the maximum length that it finds is greater than what is specified in the MEMOSIZE= option, then the smaller value that is specified in MEMOSIZE= will be applied as the SAS variable width.

SCANTIME=YES | NO;
scans all row values for a DATETIME data type field and automatically determines the TIME data type if only time values (that is, no date or datetime values) exist in the column.

SERVER="*PC-server-name*";
: specifies the name of the PC server. You must bring up the listener on the PC server before you can establish a connection to it. You can configure the service name, port number, maximum number of connections allowed, and use of data encryption on your PC server. This is a required statement. Refer to your PC server administrator for the information that is needed. Alias: SERVER_NAME=.

SERVICE="*service-name*";
: specifies the service name that is defined on your service file for your client and server machines. This statement and the PORT= statement should not be used in the same procedure. Note that this service name must be defined on both your UNIX machine and your PC server. Alias: SERVER_NAME=.

UID= "*user-id*";
: identifies the user to the DBMS. If the userid does not contain lowercase characters, special characters, or national characters, you can omit the quotation marks. You can replace the equal sign with a blank.

  Note: The DBMS client software may default to the userid and password that were used to log in to the operating environment; SAS does not generate a default value. △

WGDB= "*workgroup-database-name*" ;
: specifies the workgroup (security) database name that contains the USERID and PWD data for the DBMS. If the workgroup database name does not contain lowercase characters, special characters, or national characters, you can omit the quotation marks. You can replace the equal sign with a blank.

  Note: A default workgroup database may be used by the DBMS; SAS does not generate a default value. △

USEDATE=YES | NO;
: If USEDATE=YES, then DATE. format is used for date/time columns in the data source table while importing data from Excel workbook. If USEDATE=NO, then DATETIME. format is used for date/time.

VERSION="*file-version*";
: specifies the version of file that you want to create with if the file does not exist on your PC server yet. The default version is data-source specific. For Microsoft Excel workbook, the valid values are '2002', '2000', '97', '95' and '5', and its default value is '97'.

  Note: Always quote the version value. △

  Note: If the file already exists in the PC Server, this value can be ignored. △

Security Levels for Microsoft Access Tables

Microsoft Access tables have the following levels of security, for which specific combinations of security statements must be used:

None
: Do not specify DBPWD=, PWD=, UID=, or WGDB=.

Password
: Specify only DBPWD=.

User-level
: Specify only PWD=, UID=, and WGDB=.

Full
: Specify DBPWD=, PWD=, UID=, and WGDB=.

Each statement has a default value; however, you may find it necessary to provide a value for each statement explicitly.

Examples: IMPORT Procedure

Example 1: Importing a Delimited External File

Procedure features:

PROC IMPORT statement arguments:

- DATAFILE=
- OUT=
- DBMS=
- REPLACE

Data source statements:

- DELIMITER=
- GETNAMES=

Other features:

PRINT procedure

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.

```
proc import datafile="C:\My Documents\myfiles\delimiter.txt"
```

Identify the output SAS data set.

```
   out=mydata
```

Specify that the input file is a delimited external file.

```
   dbms=dlm
```

Overwrite the data set if it exists.

```
   replace;
```

Specify the delimiter. The DELIMITER= option specifies that an & (ampersand) delimits data fields in the input file. The delimiter separates the columns of data in the input file.

```
   delimiter='&';
```

Generate the variable names from the first row of data in the input file.

```
   getnames=yes;
run;
```

Print the WORK.MYDATA data set. PROC PRINT produces a simple listing.

```
options nodate ps=60 ls=80;

proc print data=mydata;
run;
```

SAS Log

The SAS log displays information about the successful import. For this example, PROC IMPORT generates a SAS DATA step, as shown in the partial log that follows.

```
/**********************************************************************
79    *   PRODUCT:   SAS
80    *   VERSION:   9.00
81    *   CREATOR:   External File Interface
82    *   DATE:      24JAN02
83    *   DESC:      Generated SAS Datastep Code
84    *   TEMPLATE SOURCE:  (None Specified.)
85    ***********************************************************************/
86       data MYDATA                                      ;
87      %let _EFIERR_ = 0; /* set the ERROR detection macro variable */
88       infile 'C:\My Documents\myfiles\delimiter.txt' delimiter = '&' MISSOVER
88 !  DSD lrecl=32767 firstobs=2 ;
89          informat Region $8. ;
90          informat State $2. ;
91          informat Month $7. ;
92          informat Expenses best32. ;
93          informat Revenue best32. ;
94          format Region $8. ;
95          format State $2. ;
96          format Month $7. ;
97          format Expenses best12. ;
98          format Revenue best12. ;
99       input
100                  Region $
101                  State $
102                  Month $
103                  Expenses
104                  Revenue
105      ;
106      if _ERROR_ then call symput('_EFIERR_',1);  /* set ERROR detection
106! macro variable */
107      run;

NOTE: Numeric values have been converted to character
      values at the places given by: (Line):(Column).
      106:44
NOTE: The infile 'C:\My Documents\myfiles\delimiter.txt' is:
      File Name=C:\My Documents\myfiles\delimiter.txt,
      RECFM=V,LRECL=32767

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 31.
NOTE: The data set WORK.MYDATA has 7 observations and 5 variables.
NOTE: DATA statement used (Total process time):
      real time           0.04 seconds
      cpu time            0.05 seconds


7 rows created in MYDATA                                   from C:\My
Documents\myfiles\delimiter.txt.



NOTE: .MYDATA was successfully created.
```

Output

This output lists the output data set, MYDATA, created by PROC IMPORT 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 Spreadsheet from an Excel Workbook

Procedure features:

PROC IMPORT statement arguments:

DATAFILE=
OUT=

Data source statements:

SHEET=
GETNAMES=

Other features:

PRINT procedure option:

OBS=

This example imports a specific spreadsheet from an Excel workbook, which contains multiple spreadsheets, and creates a new, permanent SAS data set named SASUSER.ACCOUNTS.

Program

Specify the input file. The filename contains the extension .XLS, which PROC IMPORT recognizes as identifying an Excel 2000 spreadsheet.

```
proc import datafile="c:\myfiles\Accounts.xls"
```

Identify the output SAS data set.

```
   out=sasuser.accounts;
```

Import only the sheet PRICES that is contained in the file ACCOUNTS.XLS.

```
   sheet='Prices';
```

Do not generate the variable names from the input file. PROC IMPORT will use default variable names.

```
   getnames=no;
run;
```

Print the SASUSER.ACCOUNTS data set. PROC PRINT produces a simple listing. The OBS= data set option limits the output to the first 10 observations.

```
proc print data=sasuser.accounts(obs=10);
run;
```

Output

The following output displays the first 10 observations of the output data set, SASUSER.ACCOUNTS:

```
                                The SAS System                                1

OBS    F1                                    F2                               F3

  1    Dharamsala Tea                        10 boxes x 20 bags            18.00
  2    Tibetan Barley Beer                   24 - 12 oz bottles            19.00
  3    Licorice Syrup                        12 - 550 ml bottles           10.00
  4    Chef Anton's Cajun Seasoning          48 - 6 oz jars                22.00
  5    Chef Anton's Gumbo Mix                36 boxes                      21.35
  6    Grandma's Boysenberry Spread          12 - 8 oz jars                25.00
  7    Uncle Bob's Organic Dried Pears       12 - 1 lb pkgs.               30.00
  8    Northwoods Cranberry Sauce            12 - 12 oz jars               40.00
  9    Mishi Kobe Beef                       18 - 500 g pkgss.             97.00
 10    Fish Roe                              12 - 200 ml jars              31.00
```

Example 3: Importing a Subset of Records from an Excel Spreadsheet

Procedure features:

PROC IMPORT statement arguments:

DATAFILE=
OUT=

This example imports a subset of an Excel spreadsheet and creates a temporary SAS data set. The WHERE= SAS data set option is specified in order to import only a subset of records from the Excel spreadsheet.

Program

Specify the input file.

```
proc import datafile='c:\Myfiles\Class.xls'
```

Identify the output SAS data set, and request that only a subset of the records be imported.

```
   out=work.femaleclass (where=(sex='F'));
run;
```

Print the new SAS data set. PROC PRINT produces a simple listing.

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

Output

The following output displays the output SAS data set, WORK.FEMALECLASS:

```
                         The SAS System                          1

        Obs    Name       Sex    Age    Height    Weight

         1     Alice       F      13     56.5       84.0
         2     Barbara     F      13     65.3       98.0
         3     Carol       F      14     62.8      102.5
         4     Jane        F      12     59.8       84.5
         5     Janet       F      15     62.5      112.5
         6     Joyce       F      11     51.3       50.5
         7     Judy        F      14     64.3       90.0
         8     Louise      F      12     56.3       77.0
         9     Mary        F      15     66.5      112.0
```

Example 4: Importing a Microsoft Access Table

Procedure features:

PROC IMPORT statement arguments:

TABLE=
OUT=
DBMS=

Data source Statements:

DATABASE=
PWD=
UID=
WGDB=

This example imports a Microsoft Access 97 table and creates a permanent SAS data set named SASUSER.CUST. The Access table has user-level security, so it is necessary to specify values for the PWD=, UID=, and WGDB= statements.

Program

Specify the input DBMS table name.

```
proc import table="customers"
```

Identify the output SAS data set.

```
     out=sasuser.cust
```

Specify that the input file is a Microsoft Access 97 table.

```
     dbms=access97;
```

Identify the user ID to the DBMS.

```
     uid="userid";
```

Specify the DBMS password to access the table.

```
     pwd="mypassword";
```

Specify the path and filename of the database that contains the table.

```
     database="c:\myfiles\east.mdb";
```

Specify the workgroup (security) database name that contains the user ID and password data for the Microsoft Access table.

```
     wgdb="c:\winnt\system32\security.mdb";
```

Print the SASUSER.CUST data set. PROC PRINT produces a simple listing. The OBS= data set option limits the output to the first five observations.

```
proc print data=sasuser.cust(obs=5);
run;
```

Output

The following output displays the first five observations of the output data set, SASUSER.CUST.

```
                              The SAS System                              1

Obs    Name                                 Street                          Zipcode

  1    David Taylor                         124 Oxbow Street                72511
  2    Theo Barnes                          2412 McAllen Avenue             72513
  3    Lydia Stirog                         12550 Overton Place             72516
  4    Anton Niroles                        486 Gypsum Street               72511
  5    Cheryl Gaspar                        36 E. Broadway                  72515
```

Example 5: Importing a Specific Spreadsheet from an Excel Workbook on a PC Server

Procedure features:

PROC IMPORT statement arguments:

- DATAFILE=
- OUT=

Data Source Statements:

- SERVER=
- SERVICE=
- SHEET=
- GETNAMES=

Other features:

PRINT procedure option:

- OBS=

This example imports a specific spreadsheet from an Excel workbook on a PC server, which contains multiple spreadsheets, and creates a new, permanent SAS data set named WORK.PRICES.

Program

```
proc import dbms=excelcs
          datafile="c:\myfiles\Invoice.xls"
          out=work.prices;
     server='Sales';
     service='pcfiles';
     sheet='Prices';
     getnames=yes;
     usedate=no;
run;

proc print data=work.prices(obs=10);
run;
```

CHAPTER 27

The 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

- 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 46, "The SUMMARY Procedure," on page 1177 for an explanation of the differences.

What Types of Output Does PROC MEANS Produce?

PROC MEANS Default Output

Output 27.1 shows the default output that PROC MEANS displays. The data set that PROC MEANS analyzes contains the integers 1 through 10. The output reports the number of observations, the mean, the standard deviation, the minimum value, and the maximum value. The statements that produce the output follow:

```
proc means data=OnetoTen;
run;
```

Output 27.1 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

Output 27.2 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 on page 580.

Output 27.2 Specified Statistics for Class Levels and Identification of Maximum Values

```
                 Summary of Volunteer Work by School and Year                1

                             The MEANS Procedure

                         N
School            Year  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

                                      Most     Most       Money     Hours
  Obs  School    Year  _TYPE_  _FREQ_  Cash     Time      Raised  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.

Syntax: MEANS Procedure

Tip: Supports the Output Delivery System. See "Output Delivery System" on page 32 for details.

ODS Table Name: Summary

Reminder: You can use the ATTRIB, FORMAT, LABEL, and WHERE statements. See Chapter 3, "Statements with the Same Function in Multiple Procedures," on page 57 for details. You can also use any global statements. See "Global Statements" on page 18 for a list.

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*;

To do this	Use this statement
Calculate separate statistics for each BY group	BY
Identify variables whose values define subgroups for the analysis	CLASS
Identify a variable whose values represent the frequency of each observation	FREQ
Include additional identification variables in the output data set	ID
Create an output data set that contains specified statistics and identification variables	OUTPUT
Identify specific combinations of class variables to use to subdivide the data	TYPES
Identify the analysis variables and their order in the results	VAR

To do this	Use this statement
Specify the number of ways to make unique combinations of class variables	WAYS
Identify a variable whose values weight each observation in the statistical calculations	WEIGHT

PROC MEANS Statement

See also: Chapter 46, "The SUMMARY Procedure," on page 1177

PROC MEANS <*option(s)*> <*statistic-keyword(s)*>;

To do this	Use this option
Specify the input data set	DATA=
Disable floating point exception recovery	NOTRAP
Specify the amount of memory to use for data summarization with class variables	SUMSIZE=
Override the SAS system option THREADS | NOTHREADS	THREADS | NOTHREADS
Control the classification levels	
Specify a secondary data set that contains the combinations of class variables to analyze	CLASSDATA=
Create all possible combinations of class variable values	COMPLETETYPES
Exclude from the analysis all combinations of class variable values that are not in the CLASSDATA= data set	EXCLUSIVE
Use missing values as valid values to create combinations of class variables	MISSING
Control the statistical analysis	
Specify the confidence level for the confidence limits	ALPHA=
Exclude observations with nonpositive weights from the analysis	EXCLNPWGTS
Specify the sample size to use for the P2 quantile estimation method	QMARKERS=
Specify the quantile estimation method	QMETHOD=
Specify the mathematical definition used to compute quantiles	QNTLDEF=
Select the statistics	*statistic-keyword*
Specify the variance divisor	VARDEF=
Control the output	

To do this	Use this option
Specify the field width for the statistics	FW=
Specify the number of decimal places for the statistics	MAXDEC=
Suppress reporting the total number of observations for each unique combination of the class variables	NONOBS
Suppress all displayed output	NOPRINT
Order the values of the class variables according to the specified order	ORDER=
Display the output	PRINT
Display the analysis for all requested combinations of class variables	PRINTALLTYPES
Display the values of the ID variables	PRINTIDVARS
Control the output data set	
Specify that the _TYPE_ variable contain character values.	CHARTYPE
Order the output data set by descending _TYPE_ value	DESCENDTYPES
Select ID variables based on minimum values	IDMIN
Limit the output statistics to the observations with the highest _TYPE_ value	NWAY

Options

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 also: "Confidence Limits" on page 553

Featured in: Example 7 on page 573

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.

Main discussion: "Output Data Set" on page 557

Interaction: When you specify more than 32 class variables, _TYPE_ automatically becomes a character variable.

Featured in: Example 10 on page 578

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.

Featured in: Example 4 on page 564

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.

Featured in: Example 6 on page 570

DATA=*SAS-data-set*
identifies the input SAS data set.

Main discussion: "Input Data Sets" on page 19

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 also: "Output Data Set" on page 557

Featured in: Example 9 on page 577

EXCLNPWGTS
excludes observations with nonpositive weight values (zero or negative) from the analysis. By default, PROC MEANS treats observations with negative weights like those with zero weights and counts them in the total number of observations.

Alias: EXCLNPWGT

See also: WEIGHT= on page 548 and "WEIGHT Statement" on page 549

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.

Featured in: Example 4 on page 564

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.

Featured in: Example 1 on page 558, Example 4 on page 564, and Example 5 on page 567

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 also: "ID Statement" on page 540

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

Featured in: Example 2 on page 560 and Example 4 on page 564

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 also: *SAS Language Reference: Concepts* for a discussion of missing values that have special meaning.

Featured in: Example 6 on page 570

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 also: "The N Obs Statistic" on page 556

Featured in: Example 5 on page 567 and Example 6 on page 570

NOPRINT
See PRINT | NOPRINT on page 531.

NOTHREADS
See THREADS | NOTHREADS on page 534.

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, this 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 also: "Output Data Set" on page 557

Featured in: Example 10 on page 578

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 that 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.

Interaction: For multiway combinations of the class variables, PROC MEANS determines the order of a class variable combination from the individual class variable frequencies.

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.

Alias: UNFMT | INTERNAL

Default: UNFORMATTED

See also: "Ordering the Class Values" on page 551

PCTLDEF=

See QNTLDEF= on page 532.

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.

Featured in: For an example of NOPRINT, see Example 8 on page 575 and Example 12 on page 583

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.

Featured in: Example 4 on page 564

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 also: “ID Statement” on page 540

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 P50), *number* is 25. For the quantiles P1, P5, P10, 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.

Main Discussion: “Quantiles” on page 555

QMETHOD=OS|P2|HIST
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 is the same method that PROC UNIVARIATE uses.

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 MEANS will not compute weighted quantiles.

Tip: When QMETHOD=P2, reliable estimations of some quantiles (P1,P5,P95,P99) may not be possible for some data sets.

Main Discussion: “Quantiles” on page 555

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.

Default: 5

Alias: PCTLDEF=

Main discussion: “Quantile and Related Statistics” on page 1345

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	RANGE
CSS	SKEWNESS | SKEW
CV	STDDEV | STD
KURTOSIS | KURT	STDERR
LCLM	SUM
MAX	SUMWGT
MEAN	UCLM
MIN	USS
N	VAR
NMISS	
Quantile statistic keywords	
MEDIAN | P50	Q3 | P75
P1	P90
P5	P95
P10	P99
Q1 | P25	QRANGE
Hypothesis testing keywords	
PROBT	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.

Main discussion: The definitions of the keywords and the formulas for the associated statistics are listed in "Keywords and Formulas" on page 1340.

Featured in: Example 1 on page 558 and Example 3 on page 562

SUMSIZE=*value*

specifies the amount of memory that is available for data summarization when you use class variables. *value* may 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.

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 also: The SAS system option SUMSIZE= in *SAS Language Reference: Dictionary*.

Main discussion: "Computational Resources" on page 552

THREADS | NOTHREADS

enables or disables parallel processing of the input data set. This option overrides the SAS system option THREADS | NOTHREADS. See *SAS Language Reference: Concepts* for more information about parallel processing.

Default: value of SAS system option THREADS | NOTHREADS.

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.

VARDEF=*divisor*

specifies the divisor to use in the calculation of the variance and standard deviation. Table 27.1 on page 534 shows the possible values for *divisor* and associated divisors.

Table 27.1 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 $\sum (x_i - \overline{x})^2$. When you weight the analysis variables, CSS equals $\sum w_i (x_i - \overline{x}_w)^2$, where $\overline{x}_w$ 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=.

Tip: 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.

Tip: When you use the WEIGHT statement and VARDEF=WGT, the computed variance is asymptotically (for large *n*) an estimate of $\sigma^2/\overline{w}$, where $\overline{w}$ is the average weight. This yields an asymptotic estimate of the variance of an observation with average weight.

See also: "Weighted Statistics Example" on page 65

Main discussion: "Keywords and Formulas" on page 1340

BY Statement

Produces separate statistics for each BY group.

Main discussion: "BY" on page 58

See also: "Comparison of the BY and CLASS Statements" on page 539

Featured in: Example 3 on page 562

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 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*.

Options

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. 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.

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 20 and "Suppressing the Default BY Line" on page 20.)

CLASS Statement

Specifies the variables whose values define the subgroup combinations for the analysis.

Tip: You can use multiple CLASS statements.

Tip: 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 also: For information about how the CLASS statement groups formatted values, see "Formatted Values" on page 25.

Featured in: Example 2 on page 560, Example 4 on page 564, Example 5 on page 567, Example 6 on page 570, and Example 10 on page 578

CLASS *variable(s)* </ *options*>;

Required Arguments

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 that 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 also: "Using Class Variables" on page 550

Options

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.

Featured in: Example 10 on page 578

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.

Featured in: Example 6 on page 570

GROUPINTERNAL

specifies not to apply formats to the class variables when PROC MEANS groups the values to create combinations of class variables.

Interaction: If you specify the PRELOADFMT option, then PROC MEANS ignores the GROUPINTERNAL option and uses the formatted values.

Interaction: 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 also: "Computer Resources" on page 539

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 also: *SAS Language Reference: Concepts* for a discussion of missing values with special meanings.

Featured in: Example 10 on page 578

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.

Interaction: 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.

Interaction: Using MLF with ORDER=FREQ may not produce the order that you expect for the formatted values.

Tip: If you omit MLF, then PROC MEANS uses the primary format labels, which corresponds to using the first external format value, to determine the subgroup combinations.

See also: The MULTILABEL option in the VALUE statement of the FORMAT procedure on page 440.

Featured in: Example 5 on page 567

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). △

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 that you define them.

Featured in: Example 10 on page 578

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

Featured in: Example 5 on page 567

FREQ

orders values by descending frequency count so that levels with the most observations are listed first.

Interaction: For multiway combinations of the class variables, PROC MEANS determines the order of a level from the individual class variable frequencies.

Interaction: Use the ASCENDING option to order values by ascending frequency count.

Featured in: Example 5 on page 567

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 also: "Ordering the Class Values" on page 551

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.

Interaction: 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.

Interaction: 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.

Featured in: Example 6 on page 570

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 on page 562.

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 allows 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 552 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.

Main discussion: "FREQ" on page 61

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 *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 Also: Discussion of *id-group-specification* in “OUTPUT Statement” on page 540.

ID *variable(s)*;

Required Arguments

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.

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 that 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 1014 for information on 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.

Featured in: Example 8 on page 575, Example 9 on page 577, Example 10 on page 578, Example 11 on page 580, and Example 12 on page 583

OUTPUT <OUT=*SAS-data-set*> <*output-statistic-specification(s)*>
<*id-group-specification(s)*> <*maximum-id-specification(s)*>
<*minimum-id-specification(s)*> </ *option(s)*>;

Options

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. See "Data Set Options" on page 18 for a list.

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
N	USS
NMISS	VAR
Quantile statistics keyword	
MEDIAN | P50	Q3 | P75
P1	P90
P5	P95
P10	P99
Q1 | P25	QRANGE
Hypothesis testing keyword	
PROBT	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, SKEWNESS, and KURTOSIS will not inherit the analysis variable's format because this format may 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.

Featured in: Example 8 on page 575, Example 9 on page 577, Example 11 on page 580, and Example 12 on page 583

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*.

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.

Featured in: Example 8 on page 575

Default: 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.

Tip: Use the AUTONAME option to have PROC MEANS generate unique names for multiple variables and statistics.

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.

Interaction: If you use WHERE processing, then the value of _OBS_ might not correspond to the location of the observation in the input data set.

Interaction: 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 $\mathrm{O}\left(T * N \log_2 n\right)$, 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 $\mathrm{O}\left(N \log_2 N\right)$.

Default: 1

Range: an integer between 1 and 100

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).

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.

Alias: IDGRP

Requirement: You must specify the MIN|MAX selection criteria first and OUT(*id-variable-list*)= after the suboptions MISSING, OBS, and LAST.

Tip: 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.

Tip: 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)=;
```

Featured in: Example 8 on page 575 and Example 12 on page 583

CAUTION:
The IDGROUP syntax allows you to create output variables with the same name. When this 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 may 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.

Tip: 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.

Tip: Use the AUTONAME option to automatically resolve naming conflicts.

Limitation: 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.

Featured in: Example 11 on page 580

CAUTION:
The MAXID syntax allows you to create output variables with the same name. When this happens, only the first variable appears in the output data set. Use the AUTONAME option to automatically resolve these naming conflicts. △

Note: If you specify fewer new variable names than the combination of analysis variables and identification variables, then the remaining output variables use the corresponding names of the ID variables as soon as PROC MEANS exhausts the list of new variable names. △

minid-specification
See the description of maximum-id-specification on page 544. 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)*

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.

Featured in: Example 12 on page 583

AUTONAME
specifies that PROC MEANS creates a unique variable name for an output statistic when you do not explicitly assign the variable name in the OUTPUT statement. This is accomplished by appending the *statistic-keyword* to the end of the input variable name from which the statistic was derived. For example, the statement

```
output min(x)=/autoname;
```

produces the x_Min variable in the output data set.

AUTONAME activates the SAS internal mechanism to automatically resolve conflicts in the variable names in the output data set. Duplicate variables 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.

Featured in: Example 12 on page 583

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).

Main discussion: “Output Data Set” on page 557

Featured in: Example 8 on page 575

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.

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 may 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.

Main discussion: “Output Data Set” on page 557

See also: “WAYS Statement” on page 548

Featured in: Example 8 on page 575

TYPES Statement

Identifies which of the possible combinations of class variables to generate.

Main discussion: “Output Data Set” on page 557

Requirement: CLASS statement

Featured in: Example 2 on page 560, Example 5 on page 567, and Example 12 on page 583

TYPES *request(s)*;

Required Arguments

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 is composed of 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.

Tip: Use ( ) to request the overall total (_TYPE_=0).

Tip: 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.

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 557.

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 also: Chapter 46, “The SUMMARY Procedure,” on page 1177

Featured in: Example 1 on page 558

VAR *variable(s)* </ WEIGHT=*weight-variable*>;

Required Arguments

variable(s)
identifies the analysis variables and specifies their order in the results.

Option

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 MEANS…
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.

Restriction: To compute weighted quantiles, use QMETHOD=OS in the PROC statement.

Restriction: Skewness and kurtosis are not available with the WEIGHT option.

Tip: When you use the WEIGHT option, consider which value of the VARDEF= option is appropriate. See the discussion of VARDEF= on page 534.

Tip: Use the WEIGHT option in multiple VAR statements to specify different weights for the analysis variables.

Note: Prior to Version 7 of SAS, the procedure did not exclude the observations with missing weights from the count of observations. △

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.

Featured in: Example 6 on page 570

WAYS *list*;

Required Arguments

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

Example: To create the two-way types for the classification variables A, B, and C, use

```
class A B C ;
ways 2;
```

This WAYS statement is equivalent to specifying a*b, a*c, and b*c in the TYPES statement.

See also: WAYS option on page 546

WEIGHT Statement

Specifies weights for observations in the statistical calculations.

See also: For information on how to calculate weighted statistics and for an example that uses the WEIGHT statement, see “WEIGHT” on page 63

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. If the value of the weight variable is

Weight value…	PROC MEANS…
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.

Restriction: Skewness and kurtosis are not available with the WEIGHT statement.

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.

Tip: When you use the WEIGHT statement, consider which value of the VARDEF= option is appropriate. See the discussion of VARDEF= on page 534 and the calculation of weighted statistics in "Keywords and Formulas" on page 1340 for more information.

Note: Prior to Version 7 of SAS, the procedure did not exclude the observations with missing weights from the count of observations. △

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 those that 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. △

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 an 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.

```
                         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 employs the same memory allocation scheme across all operating environments. When class variables are involved, PROC MEANS must keep a copy of each unique value of each class variable in memory. You can estimate the memory requirements to group the class variable by calculating

$$Nc_1\,(Lc_1 + K) + Nc_2\,(Lc_2 + K) + ... + Nc_n\,(Lc_n + K)$$

where

Nc_i is the number of unique values for the class variable

Lc_i is the combined unformatted and formatted length of c_i

K is some constant on the order of 32 bytes (64 for 64-bit architectures).

When you use the GROUPINTERNAL option in the CLASS statement, Lc_i is simply the unformatted length of c_i.

Each unique combination of class variables, $c_{1_i}\ c_{2_j}$, for a given type forms a level in that type (see “TYPES Statement” on page 546). 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

$$W * Nc_1 * Nc_2 * \ ... \ * Nc_n$$

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).

$Nc_1 ... Nc_n$ are the number of unique levels for the active class variables of the given type.

Clearly, the memory requirements of the levels overwhelm those of the class variables. For this reason, PROC MEANS may 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 may be required to combine type levels in memory with those 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 SUMSIZE 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 and PROC option SUMSIZE=. Like the system 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 may internally round up the value of SUMSIZE=. SUMSIZE= has no effect unless you specify class variables.

As an alternative, you can set the SAS system option REALMEMSIZE= in the same way that you would set SUMSIZE=. 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.

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 on 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.

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 1340 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

$$\overline{x} \pm t_{(1-\alpha/2;n-1)} \frac{s}{\sqrt{n}}$$

where s is $\sqrt{\frac{1}{n-1}\sum(x_i-\overline{x})^2}$ 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

$$\overline{x} + t_{(1-\alpha;n-1)} \frac{s}{\sqrt{n}} \qquad \text{(upper)}$$
$$\overline{x} - t_{(1-\alpha;n-1)} \frac{s}{\sqrt{n}} \qquad \text{(lower)}$$

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.

When 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

$$\overline{y}_w \pm t_{(1-\alpha/2)} \frac{s_w}{\sqrt{\sum_{i=1}^{n} w_i}}$$

where $\overline{y}_w$ 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

$$t = \frac{\overline{x}-\mu_0}{s/\sqrt{n}}$$

where $\overline{x}$ 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

$$t_w = \frac{\overline{y}_w - \mu_0}{s_w / \sqrt{\sum_{i=1}^{n} w_i}}$$

where $\overline{y}_w$ 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 1345. 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) may not be possible for some data sets such as those 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 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, SKEWNESS, and KURTOSIS do not inherit the analysis variable's format because this format may 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 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 is composed of all input observations whose formatted class values match.

Figure 27.1 on page 558 shows the values of _TYPE_ and the number of observations in the data set when you specify one, two, and three class variables.

Figure 27.1 The Effect of Class Variables on the OUTPUT Data Set

three CLASS variables
two CLASS variables
one CLASS variable

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

Procedure features:

PROC MEANS statement options:

- statistic keywords
- FW=

VAR statement

This example

- □ specifies the analysis variables
- □ computes the statistics for the specified keywords and displays them in order
- □ specifies the field width of the statistics.

Program

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;
```

Output

PROC MEANS lists PresentScore first because this is the first variable that is specified in the VAR statement. A field width of eight truncates the statistics to four decimal places.

```
                  Summary of Presentation and Taste Scores                 1

                               The MEANS Procedure

Variable          N         Mean      Maximum      Minimum        Range      Std Dev
--------------------------------------------------------------------------------
PresentScore     20      76.1500      93.0000      56.0000      37.0000       9.3768
TasteScore       20      81.3500      94.0000      72.0000      22.0000       6.6116
--------------------------------------------------------------------------------
```

Example 2: Computing Descriptive Statistics with Class Variables

Procedure features:

PROC MEANS statement option:

MAXDEC=

CLASS statement

TYPES statement

This example

- □ 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

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).

```
          Final Exam Grades for Student Status and Year of Graduation         1

                              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

                 N
 Status  Year  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

Procedure features:

PROC MEANS statement option:

- statistic keywords

BY statement

CLASS statement

Other features:

SORT procedure

Data set: GRADE on page 561

This example

- 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

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

```
         Final Exam Scores for Student Status and Year of Graduation          1
                              Within Each Section

---------------------------------- Section=A -----------------------------------

                               The MEANS Procedure

                           Analysis Variable : Score

                        N
     Status    Year    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

                        N
     Status    Year    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

Procedure features:

PROC MEANS statement options:

- CLASSDATA=
- EXCLUSIVE
- FW=
- MAXDEC=
- PRINTALLTYPES

CLASS statement

Data set: CAKE on page 559

This example

- □ 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

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.

```
                 Taste Score For Number of Layers and Cake Flavor               1

                              The MEANS Procedure

                        Analysis Variable : TasteScore

                   N
                 Obs       Range      Median     Minimum     Maximum
                 ---------------------------------------------------
                  13          22          80          72          94
                 ---------------------------------------------------


                         Analysis Variable : TasteScore

                         N
             Layers    Obs       Range      Median     Minimum     Maximum
            ---------------------------------------------------------------
                  1      8          19          82          75          94

                  2      5          20          75          72          92

                  3      0           .           .           .           .
            ---------------------------------------------------------------


                        Analysis Variable : TasteScore

                          N
          Flavor        Obs       Range      Median     Minimum     Maximum
          ------------------------------------------------------------------
          Chocolate       8          20          81          72          92

          Vanilla         5          21          80          73          94
          ------------------------------------------------------------------


                        Analysis Variable : TasteScore

                                 N
     Flavor          Layers    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

Procedure 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 on page 559

This example

- 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

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=FREQ orders the levels of Flavor by descending frequency count. ORDER=FMT orders the levels of Age by ascending formatted values. MLF specifies that multilabel value formats be used for Age.

```
   class flavor/order=freq;
   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.

```
            Taste Score for Cake Flavors and Participant's Age              1
                             The MEANS Procedure
                       Analysis Variable : TasteScore

                 Flavor           N       Min       Max    Median
                 ------------------------------------------------
                 Chocolate        9     72.00     92.00     83.00

                 Vanilla          6     73.00     94.00     82.00

                 Other Flavor     4     72.00     91.00     82.00
                 ------------------------------------------------


                       Analysis Variable : TasteScore

      Flavor          Age                   N       Min       Max    Median
      ----------------------------------------------------------------------
      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

      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
```

Example 6: Using Preloaded Formats with Class Variables

Procedure 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 on page 559

This example

- 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

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.

```
                 Taste Score For Number of Layers and Cake Flavors              1
                                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

Procedure features:

PROC MEANS statement options:

- ALPHA=
- FW=
- MAXDEC=

CLASS statement

This example

- specifies the field width and number of decimal places of the statistics
- computes a two-sided 90 percent 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

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 on page 1378 creates this data set.

```
data charity;
   input School $ 1-7 Year 9-12 Name $ 14-20 MoneyRaised 22-26
         HoursVolunteered 28-29;
   datalines;
Monroe  1992 Allison 31.65 19
Monroe  1992 Barry   23.76 16
Monroe  1992 Candace 21.11  5

     . . . more data lines . . .

Kennedy 1994 Sid     27.45 25
Kennedy 1994 Will    28.88 21
Kennedy 1994 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 '1992-94';
run;
```

Output

PROC MEANS displays the lower and upper confidence limits for both variables for each year.

```
                 Confidence Limits for Fund Raising Statistics                1
                                   1992-94

                              The MEANS Procedure

           N                         Lower 90%    Upper 90%
    Year  Obs  Variable            CL for Mean  CL for Mean       Mean    Std Dev
 ----------------------------------------------------------------------------
    1992   31  MoneyRaised               25.21        32.40      28.80      11.79
               HoursVolunteered          17.67        23.17      20.42       9.01

    1993   32  MoneyRaised               25.17        31.58      28.37      10.69
               HoursVolunteered          15.86        20.02      17.94       6.94

    1994   46  MoneyRaised               26.73        33.78      30.26      14.23
               HoursVolunteered          19.68        22.63      21.15       5.96
 ----------------------------------------------------------------------------
```

Example 8: Computing Output Statistics

Procedure features:

PROC MEANS statement option:

NOPRINT

CLASS statement

OUTPUT statement options

statistic keywords
IDGROUP
LEVELS
WAYS

Other features:

PRINT procedure

Data set: GRADE on page 561

This example

- 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

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 on 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.

```
                Average Undergraduate and Graduate Course Grades               1
                                 For Two Years

                                                          Average     Best
Status    Year    _WAY_    _TYPE_    _LEVEL_    _FREQ_     Grade      Score      _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

Procedure 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 on page 561

This example

- 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

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.

```
                  Exam and Course Grades for Undergraduates Only               1
                               and for All Students

                                                                Final     Median
   Obs     Status     Year     _TYPE_     _FREQ_      Score      Grade      Grade

    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

Procedure features:

PROC MEANS statement options:

- CHARTYPE
- NOPRINT
- NWAY

CLASS statement options:

ASCENDING
MISSING
ORDER=

OUTPUT statement

Other features:

PRINT procedure

Data set: CAKE on page 559

This example

- □ 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

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 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 both class variables, Flavor and Layers. Therefore, the value of _TYPE_ is 3 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.

```
                 Maximum Taste Score for Flavor and Cake Layers              1

                                                              High
            Obs     Flavor       Layers    _TYPE_    _FREQ_   Score

             1      Rum            2         3         1        72
             2      Spice          2         3         2        83
             3      Spice          3         3         1        91
             4      Vanilla        .         3         1        84
             5      Vanilla        1         3         3        94
             6      Vanilla        2         3         2        87
             7      Chocolate      .         3         1        84
             8      Chocolate      1         3         5        85
             9      Chocolate      2         3         3        92
```

Example 11: Identifying an Extreme Value with the Output Statistics

Procedure features:

CLASS statement

OUTPUT statement options:

statistic keyword
MAXID

Other features:

PRINT procedure

Data set: CHARITY on page 573

This example

- identifies the observations with maximum values for two variables
- creates new variables for the maximum values
- displays the output data set.

Program

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 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.

```
                 Summary of Volunteer Work by School and Year                1

                             The MEANS Procedure

                          N
School           Year   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
------------------------------------------------------------------------------
```

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.

```
            Best Results: Most Money Raised and Most Hours Worked            2

                                          Most     Most      Money     Hours
   Obs  School   Year  _TYPE_  _FREQ_  Cash     Time      Raised  Volunteered

     1              .    00      109   Willard  Tonya     78.65       40
     2           1992    01       31   Tonya    Tonya     55.16       40
     3           1993    01       32   Cameron  Amy       65.44       31
     4           1994    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  1992    11       15   Thelma   Jay       52.63       35
     8  Kennedy  1993    11       20   Bill     Amy       42.23       31
     9  Kennedy  1994    11       18   Luther   Che-Min   72.22       33
    10  Monroe   1992    11       16   Tonya    Tonya     55.16       40
    11  Monroe   1993    11       12   Cameron  Myrtle    65.44       26
    12  Monroe   1994    11       28   Willard  L.T.      78.65       33
```

Example 12: Identifying the Top Three Extreme Values with the Output Statistics

Procedure 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 on page 573

This example

□ 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

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 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.

```
                    School Fund Raising Report                          1
                           Top Three Students

                                      Money    Money
                             Number   Raised_  Raised_  Money    Money    Money
 Obs School  Year _TYPE_ Students       Sum      Mean Raised_1 Raised_2 Raised_3

  1  All      All    0      109   $3192.75   $29.29   $78.65   $72.22   $65.44
  2  All     1992    1       31    $892.92   $28.80   $55.16   $53.76   $52.63
  3  All     1993    1       32    $907.92   $28.37   $65.44   $47.33   $42.23
  4  All     1994    1       46   $1391.91   $30.26   $78.65   $72.22   $56.87
  5  Kennedy  All    2       53   $1575.95   $29.73   $72.22   $52.63   $43.89
  6  Monroe   All    2       56   $1616.80   $28.87   $78.65   $65.44   $56.87



 Obs Name_1  Name_2  Name_3  School_1 School_2 School_3 Year_1 Year_2 Year_3

  1  Willard Luther  Cameron Monroe   Kennedy  Monroe    1994   1994   1993
  2  Tonya   Edward  Thelma  Monroe   Monroe   Kennedy   1992   1992   1992
  3  Cameron Myrtle  Bill    Monroe   Monroe   Kennedy   1993   1993   1993
  4  Willard Luther  L.T.    Monroe   Kennedy  Monroe    1994   1994   1994
  5  Luther  Thelma  Jenny   Kennedy  Kennedy  Kennedy   1994   1992   1992
  6  Willard Cameron L.T.    Monroe   Monroe   Monroe    1994   1993   1994
```

```
                    Contents of the PROC MEANS Output Data Set                  2

                              The DATASETS Procedure

Data Set Name        WORK.TOP3LIST                      Observations          6
Member Type          DATA                               Variables             18
Engine               V9                                 Indexes               0
Created              18:59 Thursday, March 14, 2002     Observation Length    144
Last Modified        18:59 Thursday, March 14, 2002     Deleted Observations  0
Protection                                              Compressed            NO
Data Set Type                                           Sorted                NO
Label
Data Representation  WINDOWS
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
File Name                   filename
Release Created             9.0000B0
Host Created                WIN_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
```

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.

References

Jain R. and Chlamtac I., (1985) "The P^2 Algorithm for Dynamic Calculation of Quantiles and Histograms Without Sorting Observations," *Communications of the Association of Computing Machinery*, 28:10.

CHAPTER 28

The MIGRATE Procedure

Information about the MIGRATE Procedure

See: The MIGRATE procedure is available specifically for migrating a SAS data library from a previous release to the most recent release. For complete information about PROC MIGRATE, see the Migration Community at **http://support.sas.com/rnd/migration**.

CHAPTER 29

The OPTIONS Procedure

Overview: OPTIONS Procedure

What Does the OPTIONS Procedure Do?

The OPTIONS procedure lists the current settings of SAS system options. The results are displayed in the SAS log.

SAS system options control how the SAS System 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 can change the settings of SAS system options

- in the SAS command
- in a configuration or autoexec file
- in the SAS OPTIONS statement
- by using the OPTLOAD and OPTSAVE procedures
- through the SAS System Options window
- in other ways, depending on your operating environment. See the companion for your operating environment for details.

For information about SAS system options, see the section on SAS system options in *SAS Language Reference: Dictionary*.

What Types of Output Does PROC OPTIONS Produce?

The log that results from running PROC OPTIONS can show both the portable and host system options, their settings, and short descriptions.

The following example shows a partial log that displays the settings of portable options.

```
proc options;
run;
```

Output 29.1 Log Showing a Partial Listing of SAS System Options

```
Portable Options:

 APPLETLOC=(system-specific pathname)
                   Location of Java applets
 ARMAGENT=         ARM Agent to use to collect ARM records
 ARMLOC=ARMLOC.LOG Identify location where ARM records are to be written
 ARMSUBSYS=(ARM_NONE)
                   Enable/Disable ARMing of SAS subsystems
 NOASYNCHIO        Do not enable asynchronous input/output
 AUTOSAVELOC=      Identifies the location where program editor contents are
                   auto saved
 NOAUTOSIGNON      SAS/CONNECT remote submit will not automatically attempt
                   to SIGNON
 NOBATCH           Do not use the batch set of default values for SAS system
                   options
 BINDING=DEFAULT   Controls the binding edge for duplexed output
 BOTTOMMARGIN=0.000
                   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
 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
 CPMLIB=           Identify previously compiled libraries of CMP subroutines
                   to use when linking
 CMPOPT=(NOEXTRAMATH NOMISSCHECK NOPRECISE NOGUARDCHECK)
            Enable SAS compiler performance optimizations
 NOCOLLATE         Do not collate multiple copies of printed output
 COLORPRINTING     Print in color if printer supports color
 COMAMID=TCP       Specifies the communication access method to be used for
                   SAS distributed products
 COMPRESS=NO       Specifies whether to compress observations in output SAS
                   data sets
```

To view the setting of a particular option, you can use the option parameter on PROC OPTIONS. The following example shows a log that PROC OPTIONS produces for a single SAS system option.

```
options pagesize=60;
proc options option=pagesize;
run;
```

Output 29.2 The Setting of a Single SAS System Option

```
25   options pagesize=60;
26   proc options option=pagesize;
27   run;
    SAS (r) Proprietary Software Release XXX

 PAGESIZE=60       Number of lines printed per page of output
```

Displaying the Settings of a Group of Options

You can display the settings of a group of SAS system options that have a specific functionality, such as error handling, by using the GROUP= option.

```
proc options group=errorhandling;
run;
```

Output 29.3 Sample Output Using the GROUP= Option

```
6    proc options group=errorhandling;
7    run;
    SAS (r) Proprietary Software Release XXX

 BYERR              Set the error flag if a null data set is input to the SORT
                    procedure
 CLEANUP            Attempt recovery from out-of-resources condition
 NODMSSYNCHK        Do not enable syntax check, in windowing mode, for a
                    submitted statement block
 DSNFERR            Generate error when SAS data set not found condition occurs
 NOERRORABEND       Do not abend on error conditions
 NOERRORBYABEND     Do not abend on By-group error condition
 ERRORCHECK=NORMAL Level of special error processing to be performed
 ERRORS=20          Maximum number of observations for which complete error
                    messages are printed
 FMTERR             Treat missing format or informat as an error
 QUOTELENMAX        Enable warning for quoted string length max
 VNFERR             Treat variable not found on _NULL_ SAS data set as an error
```

The following table lists the values that are available when you use the GROUP= option with PROC OPTIONS.

Values for Use with GROUP=		
COMMUNICATIONS	GRAPHICS	MACRO
DATAQUALITY	HELP	MEMORY
EMAIL	INPUTCONTROL	META
ENVDISPLAY	INSTALL	ODSPRINT
ENVFILES	LANGUAGECONTROL	PERFORMANCE

Values for Use with GROUP=		
ERRORHANDLING	LISTCONTROL	SASFILES
EXECMODES	LOG_LISTCONTROL	SORT
EXTFILES	LOGCONTROL	

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	IDMS	ORACLE
DATACOM	IMS	REXX
DB2	ISPF	

Operating Environment Information: Refer to the SAS documentation for your operating environment for more information about these host-specific options. △

Syntax: OPTIONS Procedure

See: OPTIONS procedure in the documentation for your operating environment.

PROC OPTIONS <*option(s)*>;

PROC OPTIONS Statement

PROC OPTIONS <*option(s)*>;

To do this	Use this option
Choose the format of the listing	
Specify the long form	LONG
Specify the short form	SHORT
Display the option's description, type and group	DEFINE
Display the option's value and scope	VALUE
Restrict the number of options displayed	
Display options belonging to a group	GROUP=
Display host options only	HOST
Display portable options only	NOHOST | PORT
Display a single option	OPTION=

Options

DEFINE
displays the short description of the option, the option group, and the option type. It displays information about when the option can be set, whether an option can be restricted, and whether the PROC OPTSAVE will save the option.

Interaction: This option has no effect when SHORT is specified.

GROUP=*group-name*
displays the options in the group specified by *group-name*. For more information on options groups, see "Displaying the Settings of a Group of Options" on page 593.

HOST | NOHOST
displays only host options (HOST) or displays only portable options (NOHOST).

Alias: PORTABLE is an alias for NOHOST.

LONG | SHORT
specifies the format for displaying the settings of the SAS system options. LONG lists each option on a separate line with a description; SHORT produces a compressed listing without the descriptions.

Default: LONG

Featured in: Example 1 on page 597

NOHOST | PORT
See HOST | NOHOST on page 595.

OPTION=*option-name*
displays a short description and the value (if any) of the option specified by *option-name*. DEFINE and VALUE 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 equals sign, such as PAGESIZE=, do not include the equals sign when specifying the option to OPTION=.

Featured in: Example 2 on page 598

SHORT
See LONG | SHORT.

VALUE
displays the option value and scope, as well as how the value was set.

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. △

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, but 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. △

Examples: OPTIONS Procedure

Example 1: Producing the Short Form of the Options Listing

Procedure features:
PROC OPTIONS statement option:
SHORT

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 "Overview: OPTIONS Procedure" on page 591.

Program

List all options and their settings. SHORT lists the SAS system options and their settings without any descriptions.

```
proc options short;
run;
```

Log (partial)

```
1    proc options short;
2    run;

    SAS (r) Proprietary Software Release XXX

Portable Options:

 APPLETLOC=(system-specific pathname) ARMAGENT= ARMLOC=ARMLOC.LOG ARMSUBSYS=
(ARM_NONE) NOASYNCHIO AUTOSAVELOC= NOAUTOSIGNON NOBATCH BINDING=DEFAULT
BOTTOMMARGIN=0.000 IN BUFNO=1 BUFSIZE=0 BYERR BYLINE BYSORTED NOCAPS
NOCARDIMAGE CATCACHE=0 CBUFNO=0 CENTER NOCHARCODE CLEANUP NOCMDMACCMPLIB=
CMPOPT=(NOEXTRAMATH NOMISSCHECK NOPRECISE NOGUARDCHECK) NOCOLLATE COLORPRINTING
COMAMID=TCP COMPRESS=NO CONNECTPERSIST CONNECTREMOTE= CONNECTSTATUS CONNECTWAIT
CONSOLELOG= COPIES=1 CPUCOUNT=1 CPUID DATASTMTCHK=COREKEYWORDS DATE DATESTYLE=MDY
DBSLICEPARM=(THREADED_APPS, 2) DBSRVTP=NONE NODETAILS DEVICE= DFLANG=ENGLISH
DKRICOND=ERROR DKROCOND=WARN DLDMGACTION=REPAIR NODMR DMS NODMSEXP DMSLOGSIZE=99999
DMSOUTSIZE=99999 NODMSSYNCHK DQLOCALE= DQSETUPLOC= DSNFERR NODTRESET NODUPLEX
NOECHOAUTO EMAILAUTHPROTOCOL=NONE EMAILHOST=LOCALHOST EMAILID= EMAILPORT=25 EMAILPW=
ENGINE=V9 NOERRORABEND NOERRORBYABEND ERRORCHECK=NORMAL ERRORS=20 NOEXPLORER
FIRSTOBS=1 FMTERR FMTSEARCH=(WORK LIBRARY) FONTSLOC=(system-specific pathname)
FORMCHAR=£$<>\^_{|}~+=|-/\<>* FORMDLIM= FORMS=DEFAULT GISMAPS= GWINDOW HELPENCMD
HELPINDEX=(/help/common.hlp/index.txt /help/common.hlp/keywords.htm common.hhk)
HELPTOC=(/help/helpnav.hlp/config.txt /help/common.hlp/toc.htm common.hhc)
IBUFSIZE=0 NOIMPLMAC INITCMD= INITSTMT= INVALIDDATA=. LABEL LEFTMARGIN=0.000 IN
LINESIZE=97 LOGPARM= MACRO MAPS=(system-specific pathname) NOMAUTOLOCDISPLAY
MAUTOSOURCE MAXSEGRATIO=75 MCOMPILENOTE=NONE MERGENOBY=NOWARN MERROR
METAAUTORESOURCES= METACONNECT= METAENCRYPTALG=NONE METAENCRYPTLEVEL=EVERYTHING
METAID= METAPASS= METAPORT=0 METAPROFILE= METAPROTOCOL=BRIDGE METAREPOSITORY=Default
METASERVER= METAUSER= NOMFILE MINDELIMITER= MINPARTSIZE=0 MISSING=. NOMLOGIC
NOMLOGICNEST NOMPRINT NOMPRINTNEST NOMRECALL MSGLEVEL=N NOMSTORED MSYMTABMAX=4194304
NOMULTENVAPPL MVARSIZE=4096 NONETENCRYPT NETENCRYPTALGORITHM= NETENCRYPTKEYLEN=0
NETMAC NEWS= NOTES NUMBER NOOBJECTSERVER OBS=9223372036854775807 ORIENTATION=PORTRAIT
NOOVP NOPAGEBREAKINITIAL PAGENO=1 PAGESIZE=55 PAPERDEST= PAPERSIZE=LETTER
PAPERSOURCE= PAPERTYPE=PLAIN PARM= PARMCARDS=FT15F001 PRINTERPATH= NOPRINTINIT
PRINTMSGLIST QUOTELENMAX REPLACE REUSE=NO RIGHTMARGIN=0.000 IN NORSASUSER S=0
S2=0 SASAUTOS=(system-specific pathname) SASCMD= SASFRSCR=
SASHELP=(system-specific pathname) SASMSTORE= SASSCRIPT=
SASUSER=(system-specific pathname) SEQ=8 SERROR NOSETINIT SIGNONWAIT SKIP=0
SOLUTIONS SORTDUP=PHYSICAL SORTEQUALS SORTSEQ= SORTSIZE=2097152 SOURCE NOSOURCE2
SPDEINDEXSORTSIZE=33554432 SPDEMAXTHREADS=0 SPDESORTSIZE=33554432 SPDEUTILLOC=
SPDEWHEVAL=COST NOSPOOL NOSSLCLIENTAUTH NOSSLCRLCHECK STARTLIB SUMSIZE=0
NOSYMBOLGEN SYNCHIO SYNTAXCHECK SYSPARM= SYSPRINTFONT= NOSYSRPUTSYNC TBUFSIZE=0
TCPPORTFIRST=0 TCPPORTLAST=0 TERMINAL TERMSTMT= TEXTURELOC=\\dntsrc\sas\m901\ods\misc
THREADS TOOLSMENU TOPMARGIN=0.000 IN TRAINLOC= TRANTAB= UNIVERSALPRINT USER= UTILLOC=
UUIDCOUNT=100 UUIDGENDHOST= V6CREATEUPDATE=NOTE VALIDFMTNAME=LONG VALIDVARNAME=V7
VIEWMENU VNFERR WORK=(system-specific pathname) WORKINIT WORKTERM YEARCUTOFF=1920
 _LAST_=_NULL_
```

Example 2: Displaying the Setting of a Single Option

Procedure features:

PROC OPTIONS statement option:

- OPTION=
- DEFINE
- VALUE

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 CENTER. The DEFINE and VALUE options display additional information.

Program

Set the CENTER SAS system option.OPTION=CENTER displays option value information. DEFINE and VALUE display additional information.

```
proc options option=center define value;
run;
```

Output 29.4 Log Output from Specifying the CENTER Option

```
29   proc options option=center define value;
30   run;
    SAS (r) Proprietary Software Release XXX

Option Value Information For SAS Option CENTER
    Option Value: CENTER
    Option Scope: Default
    How option value set:  Shipped Default
Option Definition Information for SAS Option CENTER
    Group= LISTCONTROL
    Group Description: Procedure output and display settings
    Description: Center SAS procedure output
    Type: The option value is of type BOOLEAN
    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.
```

CHAPTER 30

The OPTLOAD Procedure

Overview: OPTLOAD Procedure

What Does the OPTLOAD Procedure Do?

The OPTLOAD procedure reads SAS system option settings that are stored in the SAS registry or a SAS data set and puts them into effect.

You can load SAS system option settings from a SAS data set or registry key by using

- the 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.

Some SAS options will not be saved with PROC OPTSAVE and therefore cannot be loaded with OPTLOAD. The following is a list of these options:

- ARMAGENT system option
- ARMLOC system option
- ARMSUBSYS system option
- AWSDEF system option (for Windows only)
- FONTALIAS system option (for Windows only)
- SORTMSG system option (for z/OS only)
- STIMER system option
- TCPSEC system option
- all SAS system options that can be specified only during startup
- all SAS system options that identify a password.

Syntax: OPTLOAD Procedure

PROC OPTLOAD <*options*>;

PROC OPTLOAD Statement

PROC OPTLOAD <*options*>;

To do this	Use this option
Load SAS system option settings from an existing registry key	KEY=
Load SAS system option settings from an existing data set	DATA=

Options

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.

Requirement: The SAS library and data set must exist.

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.

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".

Requirement: *"SAS registry key"* must be an existing SAS registry key.

Requirement: 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"*.

CHAPTER 31

The OPTSAVE Procedure

Overview: OPTSAVE Procedure

What Does the OPTSAVE Procedure Do?

PROC OPTSAVE saves the current SAS system option settings in the SAS registry or in a SAS data set.

SAS system options can be saved across SAS sessions. You can save the settings of the SAS system options in a SAS data set or registry key by using

- the DMOPTSAVE command from a command line in the SAS windowing environment. Use the command like this: DMOPTSAVE <*save-location*>.
- the PROC OPTSAVE statement.

Some SAS options will not be saved with PROC OPTSAVE. The following is a list of these options:

- ARMAGENT system option
- ARMLOC system option
- ARMSUBSYS system option
- AWSDEF system option
- FONTALIAS system option
- SORTMSG system option
- STIMER system option
- TPSEC system option
- All SAS system options that can be specified only during startup
- All SAS system options that identify a password.

Syntax: OPTSAVE Procedure

Tip: The only statement that is used with the OPTSAVE procedure is the PROC statement.

PROC OPTSAVE <*options*>;

PROC OPTSAVE Statement

PROC OPTSAVE <*options* >;

To do this	Use this option
Save SAS system option settings to a registry key	KEY=
Save SAS system option settings to a SAS data set	OUT=

Options

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.

Requirement: Separate the names in a sequence of key names with a backslash (\). Individual key names can contain any character except a backslash.

Requirement: The length of a key name cannot exceed 255 characters (including the backslashes).

Requirement: 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.

Caution: If the data set already exists, it will be overwritten.

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.

CHAPTER

32

The 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.

Output 32.1 is a simple plot of the high values of the Dow Jones Industrial Average (DJIA) between 1954 and 1994. PROC PLOT determines the plotting symbol and the scales for the axes. These 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 1954 to 1994';
run;
```

Output 32.1 A Simple Plot

```
                  High Values of the Dow Jones                1
                       Industrial Average
                       from 1954 to 1994

      Plot of High*Year.  Legend: A = 1 obs, B = 2 obs, etc.

 4000 +                                              A
      |                                             A
      |                                           AA
 High |                                           A
      |                                       A A
      |                                        A
 2000 +                                      A
      |                                     A
      |                                   AA
      |                  AAAAAAAAAAAAAAAAAAA
      |          AAAAAAAA
      |        AA
    0 +
      ---+---------+---------+---------+---------+---------+--
       1950      1960      1970      1980      1990      2000

                                Year
```

You can also overlay two plots, as shown in Output 32.2. 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 Output 32.2 are shown in Example 3 on page 634.

Output 32.2 Plotting Two Sets of Values at Once

```
                     Plot of Highs and Lows                    1
              for the Dow Jones Industrial Average

            Plot of High*Year.  Symbol used is '*'.
            Plot of Low*Year.  Symbol used is 'o'.

      ---+---------+---------+---------+---------+---------+---
 4000 +                                              *        +
      |                                             *         |
      |                                            * o        |
      |                                           *oo         |
 High |                                          *            |
      |                                       * *             |
      |                                           o           |
      |                                        *oo            |
 2000 +                                      * o              +
      |                                       o               |
      |                                     *o                |
      |                                   **o                 |
      |                ****** ************oo                  |
      |           *****oooooo*o o oooooooo                    |
      |      *****oooo           o                            |
      |      o                                                |
    0 +                                                       +
      ---+---------+---------+---------+---------+---------+---
       1950      1960      1970      1980      1990      2000

                                Year

NOTE: 7 obs hidden.
```

PROC PLOT can also label points on a plot with the values of a variable, as shown in Output 32.3. The plotted data represents population density and crime rates for selected U.S. states. The SAS code that produces Output 32.3 is shown in Example 11 on page 654.

Output 32.3 Labeling Points on a Plot

```
                                A Plot of Population Density and Crime Rates                                      1

                        Plot of Density*CrimeRate$State.  Symbol is value of State.

         ---+------------+------------+------------+------------+------------+------------+------------+---
 Density |                                                                                                |
     500 +                                                                                                +
         |                                                                                                |
         |                                            Maryland                                            |
         |                                               M                                                |
         |                                                                                                |
         |                                                                                                |
         |                                                                                                |
         |                                                                                                |
         |                                     Delaware                                                   |
         |                                        D                                                       |
         |            Pennsylvania          Ohio                                                          |
         |                 P                 O                                                            |
     250 +                                                                                                +
         |                                           Illinois                                             |
         |                                              I                                     Florida     |
         |                                                                                       F        |
         |                              North Carolina             California                             |
         |                   New                   South               C                                  |
         |   West         Hampshire   Alabama N   Carolina                                                |
         | Virginia           N                T     S       G Georgia                                    |
         |    W           Mississippi       A Tennessee                  Washington Texas                 |
         |                     M  Vermont V    M Missouri    Oklahoma        W        T                   |
         |         South        Arkansas A    M Minnesota       O          Oregon                         |
         |         Dakota               I Idaho                    Nevada    O                            |
       0 +           S N North Dakota                                N                                    +
         ---+------------+------------+------------+------------+------------+------------+------------+---
          2000         3000         4000         5000         6000         7000         8000         9000

                                                     CrimeRate
```

Syntax: PLOT Procedure

Requirement: At least one PLOT statement is required.

Tip: Supports RUN-group processing

Tip: Supports the Output Delivery System. See "Output Delivery System" on page 32 for details.

ODS Table Names: See: "ODS Table Names" on page 629

Reminder: You can use the ATTRIB, FORMAT, LABEL, and WHERE statements. See Chapter 3, "Statements with the Same Function in Multiple Procedures," on page 57 for details. You can also use any global statements. See "Global Statements" on page 18 for a list.

PROC PLOT <*option(s)*>;

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

  PLOT *plot-request(s)* </ *option(s)*>;

To do this	Use this statement
Produce a separate plot for each BY group	BY
Describe the plots you want	PLOT

PROC PLOT Statement

Reminder: You can use data set options with the DATA= option. See "Data Set Options" on page 18 for a list.

PROC PLOT <*option(s)*>;

To do this	Use this option
Specify the input data set	DATA=
Control the axes	
Include missing character variable values	MISSING
Exclude observations with missing values	NOMISS
Uniformly scale axes across BY groups	UNIFORM
Control the appearance of the plot	
Specify the characters that construct the borders of the plot	FORMCHAR=
Suppress the legend at the top of the plot	NOLEGEND
Specify the aspect ratio of the characters on the output device	VTOH=
Control the size of the plot	
Specify the percentage of the available horizontal space for each plot	HPERCENT=
Specify the percentage of the available vertical space for each plot	VPERCENT=

Options

DATA=*SAS-data-set*
specifies the input SAS data set.

Main discussion: See Chapter 2, "Fundamental Concepts for Using Base SAS Procedures."

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.

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 that they are listed. For instance, 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.

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 instance, 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
```

Tip: 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

Featured in: Example 4 on page 636

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.

Interaction: The HAXIS= option overrides the effect of NOMISS on the horizontal axis. The VAXIS= option overrides the effect on the vertical axis.

Interaction: NOMISS is overridden by MISSING for character variables.

Featured in: Example 10 on page 652

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

Featured in: Example 4 on page 636

See also: HPERCENT= on page 610

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.

Minimum: 0

Interaction: VTOH= has no effect if you use the HSPACE= and the VSPACE= options in the PLOT statement.

See also: HAXIS= on page 616 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.

Main discussion: "BY" on page 58

Featured in: Example 8 on page 646

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 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*.

Options

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.

PLOT *plot-request(s)* </ *option(s)*>;

To do this	Use this option
Control the axes	
Specify the tick-mark values	HAXIS= and VAXIS=
Expand the axis	HEXPAND and VEXPAND
Specify the number of print positions	HPOS= and VPOS=
Reverse the order of the values	HREVERSE and VREVERSE
Specify the number of print positions between tick marks	HSPACE= and VSPACE=
Assign a value of zero to the first tick mark	HZERO and VZERO
Specify reference lines	
Draw a line perpendicular to the specified values on the axis	HREF= and VREF=
Specify a character to use to draw the reference line	HREFCHAR= and VREFCHAR=
Put a box around the plot	BOX
Overlay plots	OVERLAY
Produce a contour plot	
Draw a contour plot	CONTOUR
Specify the plotting symbol for one contour level	S*contour-level*=
Specify the plotting symbol for multiple contour levels	SLIST=
Label points on a plot	
List the penalty and the placement state of the points	LIST=
Force the labels away from the origin	OUTWARD=
Change default penalties	PENALTIES=
Specify locations for the placement of the labels	PLACEMENT=

To do this	Use this option
Specify a split character for the label	SPLIT=
List all placement states in effect	STATES

Required Arguments

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
```

See "Labeling Plot Points with Values of a Variable" on page 625 for more information. In addition, see Example 9 on page 649 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;
```

Specifying Variable Lists in Plot Requests

You can use SAS variable lists in plot requests. For example, the following are valid 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;
```

Options

BOX

draws a border around the entire plot, rather than just on the left side and bottom.

Featured in: Example 3 on page 634

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

Featured in: Example 7 on page 642

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 on page 639 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: 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,

```
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.

Featured in: Example 2 on page 632, Example 5 on page 639, and Example 6 on page 640

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 may 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 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= on page 616.

Featured in: Example 8 on page 646

HREFCHAR='*character*'

specifies the character to use to draw the horizontal reference line.

Default: vertical bar (|)

See also: FORMCHAR= option on page 610 and HREF= on page 618

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 also: "Understanding Penalties" on page 626

Featured in: Example 11 on page 654

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.

Featured in: Example 3 on page 634

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 also: "Understanding Penalties" on page 626

Featured in: Example 13 on page 661

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 instance, 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 may 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. Table 32.1 on page 621 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 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.

Alias: PLACE=

Defaults: 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 also: “Labeling Plot Points with Values of a Variable” on page 625

Featured in: Example 11 on page 654 and Example 12 on page 658

Table 32.1 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.
	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.

Expression	Placement state	Meaning
	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.
	. . .	
	S=CENTER L=3 H=– 2 V=–2	

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 grey-scale fill characters.

Range: 1 to the highest contour level (determined by the CONTOUR option).

See also: 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';
```

Default: 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 also: 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 also: "Labeling Plot Points with Values of a Variable" on page 625

STATES

lists all the placement states in effect. STATES prints the placement states in the order that you specify them in the PLACE= option.

VAXIS=*axis-specification*

specifies tick mark values for the vertical axis. VAXIS= follows the same rules as theHAXIS= option on page 616.

Featured in: Example 7 on page 642 and Example 12 on page 658

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 also: HEXPAND on page 618

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 8 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 or not plots are overlaid, and whether or not 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 also: HPOS= on page 618

VREF=*value-specification*
draws lines on the plot perpendicular to the specified values on the vertical axis. PROC PLOT includes the values 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= on page 616.

Featured in: Example 2 on page 632

VREFCHAR='*character*'
specifies the character to use to draw the vertical reference lines.

Default: horizontal bar (-)

See also: FORMCHAR= option on page 610, HREFCHAR= on page 618, and VREF= on page 624

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.

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 forth. 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 on page 654 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 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:

S=	V=	Symbol
LEFT	any	<
RIGHT	any	>
CENTER	>0	ˆ
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 may 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. Table 32.2 on page 626 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 32.3 on page 627.

Table 32.2 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/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
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

Table 32.3 on page 627 contains the index values from Table 32.2 on page 626 with a description of the corresponding penalty.

Table 32.3 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: $$[\mathrm{MAX}(\lvert H\rvert - fhs, 0) + vsw \times \mathrm{MAX}(\lvert V\rvert - (L{+}fvs{+}(V > 0))/2, 0)]/vhsd$$ 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 has the effect of removing any penalty for a large horizontal or vertical shift. Example 6 on page 640 illustrates a case in which removing the horizontal shift penalty is useful.
8	a label may collide with its own plotting symbol. If the plotting symbol is blank, then a collision state cannot occur. See "Collision States" on page 628 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 to 11, 10, and 8 and decrease the penalties for not printing the first and second characters to 2. The following PENALTIES= option accomplishes this change:

```
penalties(15 to 20)=2 2 11 10 8 2
```

This example extends the penalty list. The twentieth 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 may 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 roughly proportional to $n \times len$. The amount of time required to split the labels is roughly 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 24*p* bytes of memory for the internal placement state list. PROC PLOT uses $n\left(84 + 5len + 4s\left(1 + 1.5\left(s+1\right)\right)\right)$ bytes for the internal list of labels. PROC PLOT buildsall 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.

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 609.

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 *SAS Output Delivery System: User's Guide*.

Table 32.4 ODS Tables Produced by the PLOT Procedure

Table Name	Description	The PLOT procedure generates the table:
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 may 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

Procedure features:

PLOT statement

plotting symbol in plot request

This example expands on Output 32.1 by specifying a different plotting symbol.

Program

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. NUMBER enables printing of the page number. 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 number pageno=1 linesize=80 pagesize=35;
```

Create the DJIA data set. DJIA contains the high and low closing marks for the Dow Jones Industrial Average from 1954 to 1994. A DATA step on page 1383 creates this data set.

```
data djia;
      input Year @7 HighDate date7. High @24 LowDate date7. Low;
      format highdate lowdate date7.;
      datalines;
1954  31DEC54  404.39  11JAN54  279.87
1955  30DEC55  488.40  17JAN55  388.20
...more data lines...
1993  29DEC93 3794.33  20JAN93 3241.95
1994  31JAN94 3978.36  04APR94 3593.35
;
```

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='*';
```

Specify the titles.

```
   title 'High Values of the Dow Jones Industrial Average';
   title2 'from 1954 to 1994';
run;
```

Output

PROC PLOT determines the tick marks and the scale of both axes.

```
                 High Values of the Dow Jones Industrial Average                1
                               from 1954 to 1994

                     Plot of High*Year.  Symbol used is '*'.

          High |
               |
          4000 +                                             *
               |                                            *
               |
               |                                           *
               |                                          *
          3000 +                                         *
               |                                       * *
               |
               |
               |                                        *
          2000 +                                      *
               |
               |                                     *
               |
               |                                   **
          1000 +                *****  ***  ***  ***
               |            ****     *    **    *
               |       *****
               |     **
               |
             0 +
               |
               ---+---------+---------+---------+---------+---------+--
                 1950      1960      1970      1980      1990      2000

                                         Year
```

Example 2: Controlling the Horizontal Axis and Adding a Reference Line

Procedure features:

PLOT statement options:

HAXIS=
VREF=

Data set: DJIA on page 631

This example specifies values for the horizontal axis and draws a reference line from the vertical axis.

Program

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=35;
```

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 1950 to 1995 in five-year increments. VREF= draws a reference line that extends from the value 3000 on the vertical axis.

```
                      / haxis=1950 to 1995 by 5 vref=3000;
```

Specify the titles.

```
   title 'High Values of Dow Jones Industrial Average';
   title2 'from 1954 to 1994';
run;
```

Output

```
                  High Values of Dow Jones Industrial Average                  1
                               from 1954 to 1994

                    Plot of High*Year.  Symbol used is '*'.

High |
     |
4000 +                                                                      *
     |                                                                     *
     |
     |                                                                   *
     |                                                                  *
3000 +----------------------------------------------------------------*---------
     |                                                           *  *
     |
     |
     |                                                             *
2000 +                                                          *
     |
     |                                                        *
     |                                                     **
1000 +                        * ** **   ** *    ** *  * **
     |                  ** **         *     * *     *
     |          ** ** *
     |      * *
     |
   0 +
     |
     -+-------+-------+-------+-------+-------+-------+-------+-------+-------+-
    1950    1955    1960    1965    1970    1975    1980    1985    1990    1995

                                         Year
```

Example 3: Overlaying Two Plots

Procedure features:

PLOT statement options

BOX

OVERLAY

Data set: DJIA on page 631

This example overlays two plots and puts a box around the plot.

Program

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=30;
```

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.

```
proc plot data=djia;
   plot high*year='*'
        low*year='o' / overlay box;
```

Specify the titles.

```
   title 'Plot of Highs and Lows';
   title2 'for the Dow Jones Industrial Average';
run;
```

Output

```
                     Plot of Highs and Lows                          1
              for the Dow Jones Industrial Average

            Plot of High*Year.  Symbol used is '*'.
            Plot of Low*Year.  Symbol used is 'o'.

      ---+---------+---------+---------+---------+---------+---
 4000 +                                              *        +
      |                                             *         |
      |                                            * o        |
      |                                           *oo         |
 High |                                          *            |
      |                                       * *             |
      |                                           o           |
      |                                        *oo            |
 2000 +                                      * o              +
      |                                       o               |
      |                                     *o                |
      |                                   **o                 |
      |                ****** ************oo                  |
      |           *****oooooo*o o oooooooo                    |
      |      *****oooo           o                            |
      |      o                                                |
    0 +                                                       +
      ---+---------+---------+---------+---------+---------+---
       1950      1960      1970      1980      1990      2000

                                Year

NOTE: 7 obs hidden.
```

Example 4: Producing Multiple Plots per Page

Procedure features:
PROC PLOT statement options
HPERCENT=
VPERCENT=

Data set: DJIA on page 631

This example puts three plots on one page of output.

Program

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=60;
```

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 1954 to 1994';
run;
```

Output

```
                                       Plots of the Dow Jones Industrial Average                                       1
                                                   from 1954 to 1994

         Plot of High*Year.  Symbol used is '*'.                        Plot of Low*Year.  Symbol used is 'o'.

4000 +                                            *           4000 +
     |                                           *                 |
     |                                                             |                                            o
     |                                          *                  |                                           o
High |                                         *               Low |                                          o
     |                                       **                    |
     |                                     *                       |
     |                                                             |                                        oo
     |                                      *                      |                                       o
2000 +                                    *                   2000 +
     |                                                             |                                     oo
     |                                   *                         |                                    o
     |                                 **                          |
     |                      **  *   ***                            |                                 ooo
     |              ********  ** ***                               |               o oo  ooo  oo o o
     |        ******                                               |         ooo oo o  oo   oo  o o o
     |    ****                                                     |     oooo   o
     |                                                             |    o
   0 +                                                           0 +
     -+---------+---------+---------+---------+---------+-         -+---------+---------+---------+---------+---------+-
    1950      1960      1970      1980      1990      2000        1950      1960      1970      1980      1990      2000

                              Year                                                          Year



         Plot of High*Year.  Symbol used is '*'.
         Plot of Low*Year.  Symbol used is 'o'.

     -+---------+---------+---------+---------+---------+-
4000 +                                            *      +
     |                                           *       |
     |                                          * o      |
     |                                         *oo       |
High |                                        *          |
     |                                     * *           |
     |                                         o         |
     |                                      *oo          |
2000 +                                    * o            +
     |                                     o             |
     |                                   *o              |
     |                                 **o               |
     |              ****** ************oo                |
     |         *****oooooo*o o oooooooo                  |
     |    *****oooo           o                          |
     |    o                                              |
   0 +                                                   +
     -+---------+---------+---------+---------+---------+-
    1950      1960      1970      1980      1990     2000

                             Year

NOTE: 7 obs hidden.
```

Example 5: Plotting Data on a Logarithmic Scale

Procedure 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

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 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;
    plot y*x / haxis=10 100 1000;
```

Specify the titles.

```
    title 'Two Plots with Different';
    title2 'Horizontal Axis Specifications';
run;
```

Output

```
                            Two Plots with Different                           1
                         Horizontal Axis Specifications

     Plot of Y*X.  A=1, B=2, etc.              Plot of Y*X.  A=1, B=2, etc.


  Y |                                       Y |
    |                                         |
3.0 +                                A    3.0 +                                A
2.9 +                         A           2.9 +                              A
2.8 +                    A                2.8 +                             A
2.7 +                A                    2.7 +                           A
2.6 +             A                       2.6 +                          A
2.5 +          A                          2.5 +                        A
2.4 +        A                            2.4 +                      A
2.3 +      A                              2.3 +                     A
2.2 +     A                               2.2 +                   A
2.1 +    A                                2.1 +                  A
2.0 +   A                                 2.0 +                A
1.9 +   A                                 1.9 +              A
1.8 +  A                                  1.8 +             A
1.7 +  A                                  1.7 +           A
1.6 + A                                   1.6 +          A
1.5 + A                                   1.5 +        A
1.4 + A                                   1.4 +      A
1.3 + A                                   1.3 +     A
1.2 + A                                   1.2 +   A
1.1 +A                                    1.1 +  A
1.0 +A                                    1.0 +A
    |                                         |
    -+---------------+---------------+        -+---------------+---------------+
     0              500          1000          10              100          1000

                     X                                         X
```

Example 6: Plotting Date Values on an Axis

Procedure features:

PLOT statement option

HAXIS=

This example shows how you can specify date values on an axis.

Program

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;
```

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;
```

Format the DATE values. The FORMAT statement assigns the DATE7. format to Date.

```
   format date date7.;
```

Specify the titles.

```
   title 'Calls to City Emergency Services Number';
   title2 'Sample of Days for 1994';
run;
```

Output

PROC PLOT uses the variables' labels on the axes.

```
                                        Calls to City Emergency Services Number                                        1
                                                Sample of Days for 1994

                                Plot of Calls*Date.  Legend: A = 1 obs, B = 2 obs, etc.

      |
      |
  600 +                                                                                                   A
      |
      |
      |
      |                                                                                                     A
N 500 +                                                             A     A
u     |               A                                   A
m     |                        A
b     |
e     |                                    A                                                                      A
r 400 +                                                                                  A
      |                                A                              A
o     |                              A  A    A                                         A
f     |                                                               A
      |                                                                         A                      A
C 300 +                                              A        A                     A                 A
a     |                    A                                     A             A
l     |
l     |                                       A  A                           A
s     |                                                                                     A
  200 +                  A       A
      |   A                       A                    A
      |                                                   A                                                     A
      |                             A
      |      A
  100 +
      |
      ---+--------+--------+--------+--------+--------+--------+--------+--------+--------+--------+--------+--------+--
      01JAN94  01FEB94  01MAR94  01APR94  01MAY94  01JUN94  01JUL94  01AUG94  01SEP94  01OCT94  01NOV94  01DEC94  01JAN95

                                                             Date
```

Example 7: Producing a Contour Plot

Procedure 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

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=25;
```

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 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 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. NOOVP ensures that overprinting is not used in the plot.

```
options nodate pageno=1 linesize=120 pagesize=60 noovp;
```

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.

```
                                  A Contour Plot                                          1

                               Contour plot of Y*X.

       Y |
         |
     350 +  ======++++++OOOOOOOOXXXXXXXXXXXWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWXXXXXXXXXXXOOOOOOOO
     340 +  ====++++++OOOOOOOXXXXXXXXXXWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWXXXXXXXXXXOOOOOOO
     330 +  =++++++OOOOOOOXXXXXXXXXWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWXXXXXXXXXOOOOO
     320 +  +++++OOOOOOOXXXXXXXXWWWWWWWWWWWWWW********************WWWWWWWWWWWWWWXXXXXXXXXOOOO
     310 +  +++OOOOOOXXXXXXXXWWWWWWWWWWWW*****************************WWWWWWWWWWXXXXXXXXXOOOO
     300 +  +OOOOOOXXXXXXXXWWWWWWWWWW***********************************WWWWWWWWWXXXXXXXXXOOO
     290 +  OOOOOXXXXXXXWWWWWWWWWW****************************************WWWWWWWWWXXXXXXXOOO
     280 +  OOOXXXXXXXWWWWWWWWW********************###*********************WWWWWWWWWXXXXXXXOO
     270 +  OXXXXXXXWWWWWWWWW**************##################***************WWWWWWWXXXXXXXXOO
     260 +  XXXXXXWWWWWWWW*************#########################************WWWWWWWXXXXXXXXOO
     250 +  XXXXWWWWWWWW************#############################************WWWWWWWXXXXXXXOO
     240 +  XXXWWWWWWW***********#################################***********WWWWWWWXXXXXXXOO
     230 +  XWWWWWWWW**********####################################**********WWWWWWXXXXXXXXOO
     220 +  WWWWWWW**********######################################**********WWWWWWXXXXXXXOOO
     210 +  WWWWWW*********########################################**********WWWWWWXXXXXXXOOO
     200 +  WWWWW*********#########################################*********WWWWWWWXXXXXXOOOO
     190 +  WWW**********##########################################*********WWWWWWWXXXXXXOOOO
     180 +  WW*********###########################################*********WWWWWWWXXXXXXOOOOO
     170 +  W*********############################################*********WWWWWWWXXXXXXOOOOO
     160 +  W*********###########################################*********WWWWWWWXXXXXXOOOOO+
     150 +  *********###########################################*********WWWWWWWXXXXXXOOOOO++
     140 +  ********###########################################*********WWWWWWWXXXXXXOOOOO+++
     130 +  ********##########################################*********WWWWWWWXXXXXXOOOOO++++
     120 +  ********########################################**********WWWWWWWXXXXXXOOOOO+++++
     110 +  ********#######################################**********WWWWWWWXXXXXXOOOOO+++++=
     100 +  ********#####################################**********WWWWWWWXXXXXXOOOOOO+++++==
      90 +  ********###################################**********WWWWWWWWXXXXXXOOOOO+++++====
      80 +  *********################################***********WWWWWWWXXXXXXXOOOOO+++++====-
      70 +  **********############################************WWWWWWWWXXXXXXOOOOOO+++++====--
      60 +  ************######################**************WWWWWWWWXXXXXXXOOOOO+++++=====---
      50 +  ***************###############***************WWWWWWWWWXXXXXXXOOOOOO+++++====----'
      40 +  W******************************************WWWWWWWWWXXXXXXXOOOOOO+++++=====----''
      30 +  WW**************************************WWWWWWWWWWXXXXXXXOOOOOO+++++=====----''''
      20 +  WWWW********************************WWWWWWWWWWWXXXXXXXXOOOOOO++++++====-----''''.
      10 +  WWWWWW**************************WWWWWWWWWWWWWXXXXXXXXOOOOOO++++++=====----''''...
       0 +  WWWWWWWWWW*****************WWWWWWWWWWWWWWWXXXXXXXXOOOOOOO++++++=====----'''''....
         |
         ---+---------+---------+---------+---------+---------+---------+---------+---------+--
            0        50        100       150       200       250       300       350       400

                                                 X

 Symbol           z     Symbol           z     Symbol           z     Symbol           z     Symbol           z

 .....     2.2 -  8.1   -----   14.0 - 19.9    +++++   25.8 - 31.7    XXXXX   37.6 - 43.5    *****   49.4 - 55.4
 '''''     8.1 - 14.0   =====   19.9 - 25.8    OOOOO   31.7 - 37.6    WWWWW   43.5 - 49.4    #####   55.4 - 61.3
```

Example 8: Plotting BY Groups

Procedure features:
PLOT statement option
HREF=
Other features:
BY statement

This example shows BY group processing in PROC PLOT.

Program

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=35;
```

Create the EDUCATION data set. EDUCATION contains educational data* 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. A DATA step on page 1384 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
;
```

* Source: U.S. Department of Education.

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 extende from 28.6 on the horizontal axis. The reference line represents the national average.

```
    plot expenditures*dropoutrate='*' / href=28.6;
```

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.

```
                 Plot of Dropout Rate and Expenditure Per Pupil                1

---------------------------------- Region=MW -----------------------------------

             Plot of Expenditures*DropoutRate.  Symbol used is '*'.

    Expenditures |                                                  |
            5500 +                                                  |
                 |                                                  |
                 |                                                  |
                 |                                                  |
                 |                                                  | *
            5000 +                                                  |
                 |                                *                 |
                 |      *                                           |
                 |                                                  |
                 |                                       *          |
            4500 +                                                  |
                 |            *          *                          |
                 |           **                                *    |
                 |                                                  |
                 |                                                  |
            4000 +       *                                          |
                 |                                                  |
                 |                                                  |
                 |                                                  |
                 |                                                  |
            3500 +                                                  |
                 |                                                  |
                 ---+------------+------------+------------+------------+--
                   10           15           20           25           30

                                  Dropout Percentage - 1989
```

```
                 Plot of Dropout Rate and Expenditure Per Pupil                2

---------------------------------- Region=NE -----------------------------------

             Plot of Expenditures*DropoutRate.  Symbol used is '*'.

    Expenditures |                                     |
            8000 +                                     |
                 |                                     |
                 |                *                    |
                 |                                     |
                 |                                     |
            7000 +                                     |
                 |       *                             |
                 |                                     |
                 |                                     |
                 |                                     |
            6000 +                                    *|
                 |                               *     |
                 |                                     |
                 |                                     *
                 |                                     |
            5000 +                                     |
                 |                      *       *      |
                 |                                     |
                 |                                     |
                 |                                     |
            4000 +                                     |
                 |                                     |
                 ---+------------+------------+------------+------------+--
                   15           20           25           30           35

                                  Dropout Percentage - 1989

NOTE: 1 obs had missing values.
```

Example 9: Adding Labels to a Plot

Procedure features:

PLOT statement

label variable in plot request

Data set: EDUCATION on page 646

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 on page 646.

Program

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=35;
```

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.

```
   plot expenditures*dropoutrate='*' $ state / href=28.6;
```

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.

```
                  Plot of Dropout Rate and Expenditure Per Pupil                1

 ---------------------------------- Region=MW -----------------------------------

           Plot of Expenditures*DropoutRate$State.  Symbol used is '*'.

     Expenditures |                                                   |
             5500 +                                                   |
                  |                                                   |
                  |                                                   |
                  |                                                   |
                  |                                           Michigan *
             5000 +                                                   |
                  |                                  * Illinois       |
                  |      * Minnesota                                  |
                  |                                                   |
                  |                                         * Ohio    |
             4500 +                                                   |
                  |            * Nebraska * Kansas                    |
                  |      Iowa ** Indiana                            * Missouri
                  |                                                   |
                  |                                                   |
             4000 +       * North Dakota                              |
                  |                                                   |
                  |                                                   |
                  |                                                   |
                  |                                                   |
             3500 +                                                   |
                  |                                                   |
                  ---+------------+------------+------------+------------+--
                    10           15           20           25           30

                                 Dropout Percentage - 1989
```

```
                 Plot of Dropout Rate and Expenditure Per Pupil                 2

---------------------------------- Region=NE -----------------------------------

          Plot of Expenditures*DropoutRate$State.  Symbol used is '*'.

    Expenditures |                                      |
            8000 +                                      |
                 |                                      |
                 |                * New Jersey          |
                 |                                      |
                 |                                      |
            7000 +                                      |
                 |       * Connecticut                  |
                 |                                      |
                 |                                      |
                 |                                      |
            6000 +                                     *|Massachusetts
                 |                                * Maryland
                 |                                      |
                 |                                      * Delaware
                 |                                      |
            5000 +                                      |
                 |                      * Maine * New Hampshire
                 |                                      |
                 |                                      |
                 |                                      |
            4000 +                                      |
                 |                                      |
                 ---+------------+------------+------------+------------+--
                   15           20           25           30           35

                                 Dropout Percentage - 1989

NOTE: 1 obs had missing values.
```

Example 10: Excluding Observations That Have Missing Values

Procedure features:

PROC PLOT statement option

NOMISS

Data set: EDUCATION on page 646

This example shows how missing values affect the calculation of the axes.

Program

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=35;
```

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.

```
    plot expenditures*dropoutrate='*' $ state / href=28.6;
```

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 on page 649.

```
                   Plot of Dropout Rate and Expenditure Per Pupil                1

 ---------------------------------- Region=NE ----------------------------------

            Plot of Expenditures*DropoutRate$State.  Symbol used is '*'.

 Expenditures |                                                           |
         8000 +                                                           |
              |                                                           |
              |                      * New Jersey                         |
              |                                                           |
              |                                                           |
         7000 +                                                           |
              |     * Connecticut                                         |
              |                                                           |
              |                                                           |
              |                                                           |
         6000 +                                         Massachusetts *   |
              |                                              * Maryland   |
              |                                                           |
              |                                                Delaware * |
              |                                                           |
         5000 +                                                           |
              |                               * Maine        * New Hampshire
              |                                                           |
              |                                                           |
              |                                                           |
         4000 +                                                           |
              |                                                           |
              --+--------+--------+--------+--------+--------+--------+--------+-
               16       18       20       22       24       26       28       30

                                    Dropout Percentage - 1989

 NOTE: 1 obs had missing values.
```

Example 11: Adjusting Labels on a Plot with the PLACEMENT= Option

Procedure features:

PLOT statement options

label variable in plot request
LIST=
PLACEMENT=

Other features:

RUN group processing

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 variable in the data set.*

This example also shows RUN group processing in PROC PLOT.

Program

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=37;
```

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 on page 1377 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;
```

* Source: U.S. Bureau of the Census and the 1987 Uniform Crime Reports, FBI.

Specify the title.

```
   title 'A Plot of Population Density and Crime Rates';
run;
```

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. Seven label characters are hidden.

```
                                      A Plot of Population Density and Crime Rates                                     1

                              Plot of Density*CrimeRate$State.  Symbol is value of State.

               ---+------------+------------+------------+------------+------------+------------+------------+---
       Density |                                                                                                |
           500 +                                                                                                +
               |                                                                                                |
               |                                                                                                |
               |                                               M Maryland                                       |
               |                                                                                                |
               |                                                                                                |
               |                                                                                                |
               |                                                                                                |
               |                                                                                                |
               |                                        D Delaware                                              |
               |                                                                                                |
               |                 P Pennsylvania    O Ohio                                                       |
           250 +                                                                                                +
               |                                                                                                |
               |                                              I Illinois                                        |
               |                                                                                       F Florida|
               |                                                                                                |
               |                              North Carolina                 C California                       |
               |                                TennNssee       Georgia                                         |
               |                    N New Hampshire  T     S South Garolina                                     |
               |    W West Virginia               A Alabama                                                     |
               |         Mississippi M  Vermont V    M Missouri         Washington W        T Texas             |
               |                          MinneAoArkMnsas             O Oklahoma                                |
               |        North Dakota          I Idaho                              O Oregon                     |
             0 +           S Nouth Dakota                                  N Nevada                             +
               ---+------------+------------+------------+------------+------------+------------+------------+---
                2000         3000         4000         5000         6000         7000         8000         9000

                                                           CrimeRate

NOTE: 7 label characters hidden.
```

A Plot of Population Density and Crime Rates 2

List of Point Locations, Penalties, and Placement States

Label	Vertical Axis	Horizontal Axis	Penalty	Starting Position	Lines	Vertical Shift	Horizontal Shift
Tennessee	111.60	4665.6	2	Center	1	1	-1
South Carolina	103.40	5161.9	2	Right	1	0	2
Arkansas	43.90	4245.2	6	Right	1	0	2
Minnesota	51.20	4615.8	7	Left	1	0	-2
South Dakota	9.10	2678.0	11	Right	1	0	2

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
```

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

No collisions occur in the plot.

```
                               A Plot of Population Density and Crime Rates                                     3

                       Plot of Density*CrimeRate$State.  Symbol is value of State.

        ---+------------+------------+------------+------------+------------+------------+------------+---
Density |                                                                                                |
    500 +                                                                                                +
        |                                                                                                |
        |                                            Maryland                                            |
        |                                               M                                                |
        |                                                                                                |
        |                                                                                                |
        |                                                                                                |
        |                                                                                                |
        |                                     Delaware                                                   |
        |                                        D                                                       |
        |            Pennsylvania          Ohio                                                          |
        |                 P                 O                                                            |
    250 +                                                                                                +
        |                                           Illinois                                             |
        |                                              I                                     Florida     |
        |                                                                                       F        |
        |                              North Carolina             California                             |
        |                   New                   South               C                                  |
        |   West         Hampshire   Alabama N   Carolina                                                |
        | Virginia           N                T     S       G Georgia                                    |
        |    W           Mississippi       A Tennessee                  Washington Texas                 |
        |                     M  Vermont V    M Missouri    Oklahoma        W        T                   |
        |         South        Arkansas A    M Minnesota       O          Oregon                         |
        |         Dakota               I Idaho                    Nevada    O                            |
      0 +           S N North Dakota                                N                                    +
        ---+------------+------------+------------+------------+------------+------------+------------+---
         2000         3000         4000         5000         6000         7000         8000         9000

                                                    CrimeRate
```

Example 12: Adjusting Labeling on a Plot with a Macro

Procedure features:

PLOT statement options

label variable in plot request
PLACEMENT=

Data set: CENSUS on page 655

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

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=37;
```

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
           %place(4);
```

Specify the title.

```
   title 'A Plot of Population Density and Crime Rates';
run;
```

Output

No collisions occur in the plot.

```
                        A Plot of Population Density and Crime Rates                            1

                Plot of Density*CrimeRate$State.  Symbol is value of State.

         ---+------------+------------+------------+------------+------------+------------+------------+---
 Density |                                                                                                |
     500 +                                                                                                +
         |                                                                                                |
         |                                                                                                |
         |                                             M Maryland                                         |
         |                                                                                                |
         |                                                                                                |
         |                                                                                                |
         |                                                                                                |
         |                                                                                                |
         |                                      D Delaware                                                |
         |                                                                                                |
         |                 P Pennsylvania    O Ohio                                                       |
     250 +                                                                                                +
         |                                                                                                |
         |                                            I Illinois                                          |
         |                                                                                       F Florida|
         |                                                                                                |
         |                              North Carolina                 C California                       |
         |                                    N Tennessee                                                 |
         |                    N New Hampshire  T     S       G Georgia                                    |
         |    W West Virginia       Alabama A  South Carolina                                             |
         |         Mississippi M  Vermont V    M Missouri         Washington W        T Texas             |
         |                      Arkansas A    M Minnesota       O Oklahoma                                |
         |      South Dakota            I Idaho                              O Oregon                     |
       0 +           S N North Dakota                          N Nevada                                   +
         ---+------------+------------+------------+------------+------------+------------+------------+---
           2000         3000         4000         5000         6000         7000         8000         9000

                                                   CrimeRate
```

Example 13: Changing a Default Penalty

Procedure features:
PLOT statement option
PENALTIES=

Data set: CENSUS on page 655

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

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=37;
```

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 labels each point with the corresponding state name.

```
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;
```

Specify the title.

```
   title 'A Plot of Population Density and Crime Rates';
run;
```

Output

```
                              A Plot of Population Density and Crime Rates                                    1

                      Plot of Density*CrimeRate$State.  Symbol is value of State.

 Density |
     500 +
         |
         |
         |
         |                                        M                                                  Maryland
     400 +
         |
         |
         |
         |
     300 +                                    D                                                      Delaware
         |
         |                    P           O                                                 Pennsylvania Ohio
         |
         |
     200 +                                       I                                                   Illinois
         |Florida                                                        F
         |                                                C                                        California
         |
         |                                    T                                    North Carolina Tennessee
     100 +Georgia                 N             S    G                         New Hampshire South Carolina
         |                  W                A M                             Alabama Missouri West Virginia
         |Washington Texas         M     V  M                  W     T          Vermont Minnesota Mississippi
         |Oklahoma                       A             O                                           Arkansas
         |Oregon                        I                      O                                      Idaho
       0 +                     S N                        N                  North Dakota South Dakota Nevada
         ---+-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+--
            0      1000    2000    3000    4000    5000    6000    7000    8000    9000    10000   11000   12000   13000

                                                     CrimeRate

NOTE: 1 obs hidden.
```

A Plot of Population Density and Crime Rates 2

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 33

The PMENU Procedure

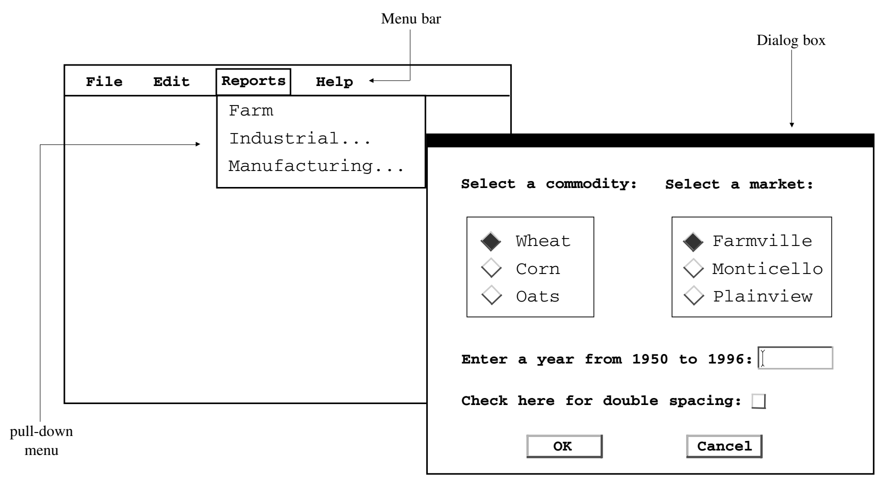
Overview: PMENU Procedure

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 upon 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 33.1 Menu Bar, Pull-Down Menu, and Dialog Box



Note: A menu bar in some operating environments may appear as a popup menu or may 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.

Syntax: PMENU Procedure

Restriction: You must use at least one MENU statement followed by at least one ITEM statement.

Tip: Supports RUN group processing

Reminder: You can also use appropriate global statements with this procedure. See Chapter 2, “Fundamental Concepts for Using Base SAS Procedures,” on page 15 for a list.

See: PMENU Procedure in the documentation for your operating environment.

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*;

To do this	Use this statement
Define choices a user can make in a dialog box	CHECKBOX
Describe a dialog box that is associated with an item in a pull-down menu	DIALOG
Identify an item to be listed in a menu bar or in a pull-down menu	ITEM
Name the catalog entry or define a pull-down menu	MENU
List and define mutually exclusive choices within a dialog box	RADIOBOX and RBUTTON
Define a command that is submitted when an item is selected	SELECTION
Draw a line between items in a pull-down menu	SEPARATOR
Define a common submenu associated with an item	SUBMENU
Specify text and the input fields for a dialog box	TEXT

PROC PMENU Statement

Invokes the PMENU procedure and specifies where to store all PMENU catalog entries that are created in the PROC PMENU step.

PROC PMENU <CATALOG=<*libref.*>*catalog*>
<DESC *'entry-description'*>;

Options

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 data library. If you omit CATALOG=, then the entries are stored in the SASUSER.PROFILE catalog.

Featured in: Example 1 on page 682

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.

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.

Options

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.

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 pull-down menu.

Restriction: Must be followed by at least one TEXT statement.

Featured in: Example 2 on page 685, Example 3 on page 688, and Example 4 on page 694

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. △

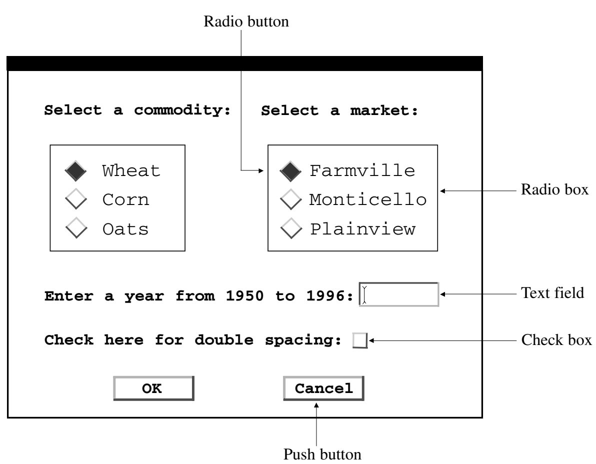
&1...&*n*
are optional CHECKBOX statement numbers that can add information to the command before it is submitted. Numbers preceded by an ampersand (&) correspond to CHECKBOX statements following the DIALOG statement.

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 pull-down 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.
- Figure 33.2 on page 670 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 box*, 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 33.2 A Typical Dialog Box



Dialog boxes have two or more *buttons*, such as OK and Cancel, automatically built into the box.* A button causes an action to occur.

ITEM Statement

Identifies an item to be listed in a menu bar or in a pull-down menu.

Featured in: Example 1 on page 682

* The actual names of the buttons vary in different windowing environments.

ITEM *command* <*option(s)*><*action-options*>;

ITEM *'menu-item'* <*option(s)*><*action-options*>;

To do this	Use this option
Specify the action for the item	
Associate the item with a dialog box	DIALOG=
Associate the item with a pull-down menu	MENU=
Associate the item with a command	SELECTION=
Associate the item with a common submenu	SUBMENU=
Specify help text for an item	HELP=
Define a key that can be used instead of the pull-down menu	ACCELERATE=
Indicate that the item is not an active choice in the window	GRAY
Provide an ID number for an item	ID=
Define a single character that can select the item	MNEMONIC=
Place a check box or a radio button next to an item	STATE=

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 (%).

Options

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 pull-down menu.

Restriction: The functionality of this option is limited to only a few characters. For details, see the SAS documentation for your operating environment.

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.

action-option
is one of the following:

DIALOG=*dialog-box*
the name of an associated DIALOG statement, which displays a dialog box when the user selects this item.

Featured in: Example 3 on page 688

MENU=*pull-down-menu*
the name of an associated MENU statement, which displays a pull-down menu when the user selects this item.

Featured in: Example 1 on page 682

SELECTION=*selection*
the name of an associated SELECTION statement, which submits a command when the user selects this item.

Featured in: Example 1 on page 682

SUBMENU=*submenu*
the name of an associated SUBMENU statement, which displays a pmenu entry when the user selects this item.

Featured in: Example 1 on page 682

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*
a value that is used as an identifier for an item in a pull-down 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.

Minimum: 3001

Restriction: Integers from 0 to 3000 are reserved for operating environment and SAS use.

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: ID= is useful with the WINFO function in SAS Component Language.

Tip: You can use the same ID for more than one item.

See also: STATE= option on page 673

MNEMONIC=*character*

underlines the first occurrence of *character* in the text string that appears on the pull-down 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.

Tip: STATE= is used with the ID= option and the WINFO function in SAS Component Language.

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.

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 pull-down 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 DIA0LOG, 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 pull-down menu. See “SEPARATOR Statement” on page 677 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 pull-down menu.

Featured in: Example 1 on page 682

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 pull-down 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.

Defining Pull-Down Menus

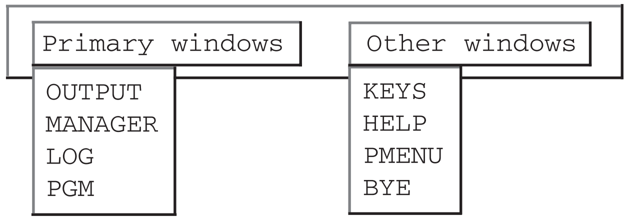
When used to define a pull-down menu, the MENU statement must follow an ITEM statement that specifies the MENU= option. Both the ITEM statement and the MENU statement for the pull-down menu must be in the same RUN group as the MENU statement that defines the menu bar for the PMENU catalog entry.

For both menu bars and pull-down 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 pull-down 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 pull-down menu */
   menu prime;
   item output;
   item manager;
   item log;
   item pgm;

      /* create second pull-down 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 33.3 Pull-Down Menu



RADIOBOX Statement

Defines a box that contains mutually exclusive choices within a dialog box.

Restriction: Must be used after a DIALOG statement.

Restriction: Must be followed by one or more RBUTTON statements.

Featured in: Example 3 on page 688

RADIOBOX DEFAULT=*button-number*;

Required Arguments

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.

Featured in: Example 3 on page 688

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, you will get an error message. Also, specify space between other text and a radio button. △

Options

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.

Featured in: Example 3 on page 688

SELECTION Statement

Defines a command that is submitted when an item is selected.

Restriction: Must be used after an ITEM statement

Featured in: Example 1 on page 682 and Example 4 on page 694

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.

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 pull-down 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, and you must return control to the PROGRAM EDITOR window. △

SEPARATOR Statement

Draws a line between items on a pull-down menu.

Restriction: Must be used after an ITEM statement.

Restriction: Not available in all operating environments.

SEPARATOR;

SUBMENU Statement

Specifies the SAS file that contains a common submenu associated with an item.

Featured in: Example 1 on page 682

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.

Featured in: Example 2 on page 685

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 prefixed by an at sign (@).

Featured in: Example 2 on page 685

’*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.

Options

ATTR=*attribute*
defines the attribute for the text or input field. Valid attribute values are
- BLINK
- HIGHLIGH
- REV_VIDE
- UNDERLIN

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.

Restriction: Your hardware may not support all of these attributes.

COLOR=*color*
defines the color for the text or input field characters. These are the color values that you can use:

BLACK	BROWN
GRAY	MAGENTA
PINK	WHITE
BLUE	CYAN
GREEN	ORANGE
RED	YELLOW

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.

Restriction: Your hardware may not support all of these colors.

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 680.

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, pull-down menus and other features that you want. Store the output of PROC PMENU in a SAS catalog.

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. See "Associating a Menu with a Window" on page 697.
   - 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 on page 700.
   - the PMENU function in SAS/AF and SAS/FSP software.
   - the SETPMENU command in SAS/FSP software. See Example 1 on page 682.

4 Activate the window you have 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 pull-down 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 that which 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;
```

Examples: PMENU Procedure

The windows in these examples were produced in the UNIX environment and may 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.

Example 1: Building a Menu Bar for an FSEDIT Application

Procedure features:

PROC PMENU statement option:

CATALOG=

ITEM statement options:

MENU=
SELECTION=
SUBMENU=

MENU statement

SELECTION statement

SUBMENU statement

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 pull-down menus do not enable end users to delete or duplicate observations.

Program

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

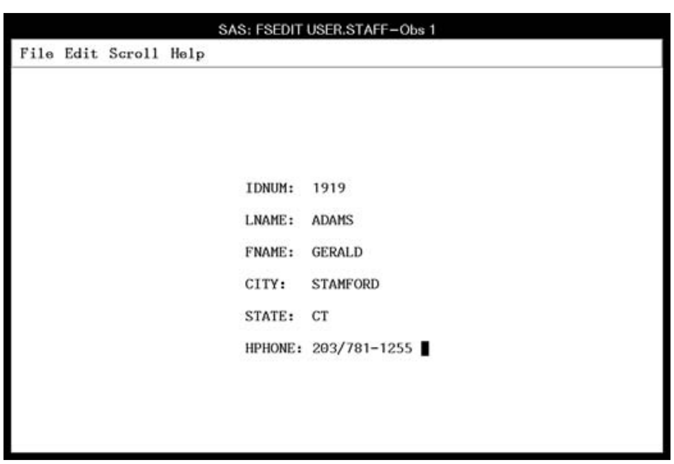
The following SETPMENU command associates the customized menu bar with the FSEDIT window.

```
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).

See "Associating a Menu Bar with an FSEDIT Session" on page 691 for other methods of associating the customized menu bar with the FSEDIT window.

The FSEDIT window shows the menu bar.



Example 2: Collecting User Input in a Dialog Box

Procedure features:

DIALOG statement

TEXT statement option:

LEN=

This example adds a dialog box to the menus created in Example 1 on page 682. The dialog box enables the user to use a WHERE clause to subset the SAS data set.

Tasks include

- collecting user input in a dialog box
- creating customized menus for an FSEDIT application.

Program

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

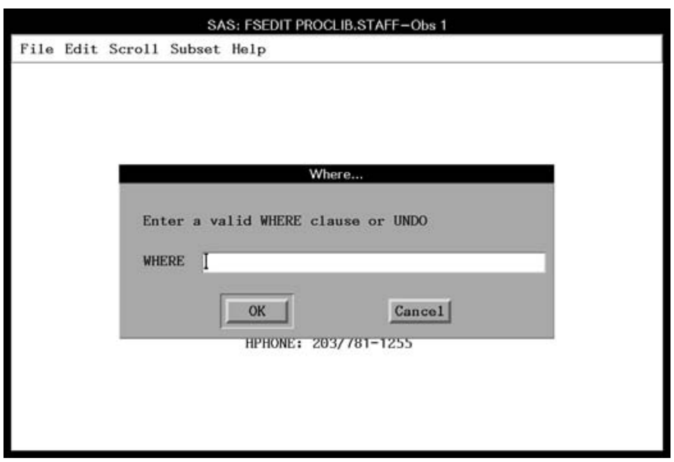
The following SETPMENU command associates the customized menu bar with the FSEDIT window.

```
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 Component Language: Reference* for complete documentation on SCL.

See "Associating a Menu Bar with an FSEDIT Session" on page 691 for other methods of associating the customized menu bar with the FSEDIT window.

This dialog box appears when the user chooses **Subset** and then **Where**.



Example 3: Creating a Dialog Box to Search Multiple Variables

Procedure features:

DIALOG statement

SAS macro invocation

ITEM statement

DIALOG= option

RADIOBOX statement option:

DEFAULT=

RBUTTON statement option:

SUBSTITUTE=

Other features: SAS macro invocation

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

- associating customized menus with an FSEDIT session
- searching multiple variables with a WHERE clause
- extending PROC PMENU functionality with a SAS macro.

Program

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 boxes. The field number @1 equates to the search value that the user enters. See "How the WBUILD Macro Works" on page 693.

```
dialog d1 '%%wbuild(%1,%2,@1,%3)';
```

Add a radio box 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 box 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 box 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 box 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';
```

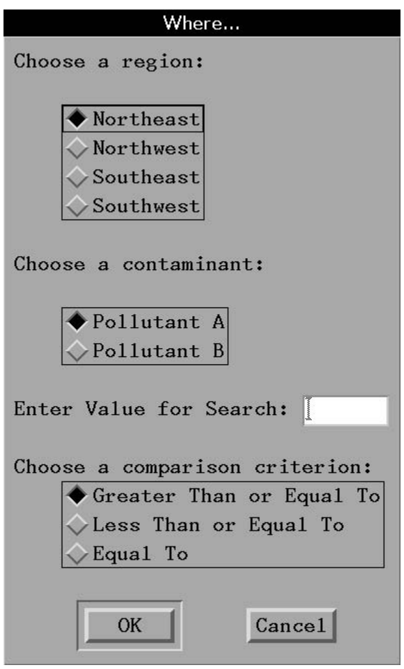
Add an input field. The first TEXT statement specifies text for the dialog box that appears on line 13 and begins in column 1. The second TEXT statement specifies an input field that is 6 bytes long that appears on line 13 and begins in column 25. The value that the user enters in the field is substituted for the @1 in the preceding DIALOG statement.

```
          text #13 @1 'Enter Value for Search:';
          text #13 @25 len=6;
```

Add a radio box 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 box 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;
```

This dialog box appears when the user selects **Subset** and then **Where**.



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:

Output 33.1

```
                              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
```

A DATA step on page 1393 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 if 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 see if 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

Procedure features:
DIALOG statement
SELECTION statement
Other features: FILENAME statement

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

- associating customized menus with a DATA step window
- creating menus for a DATA step window
- submitting SAS code from a menu selection
- creating a pull-down menu selection that calls a dialog box.

Program

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. (See "Using a Data Entry Program" on page 698.) 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. (See "Using a Data Entry Program" on page 698.) 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 pull-down 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. (See "Printing a Program" on page 699.) 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. (See “Printing a Program” on page 699.) 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;
```

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:

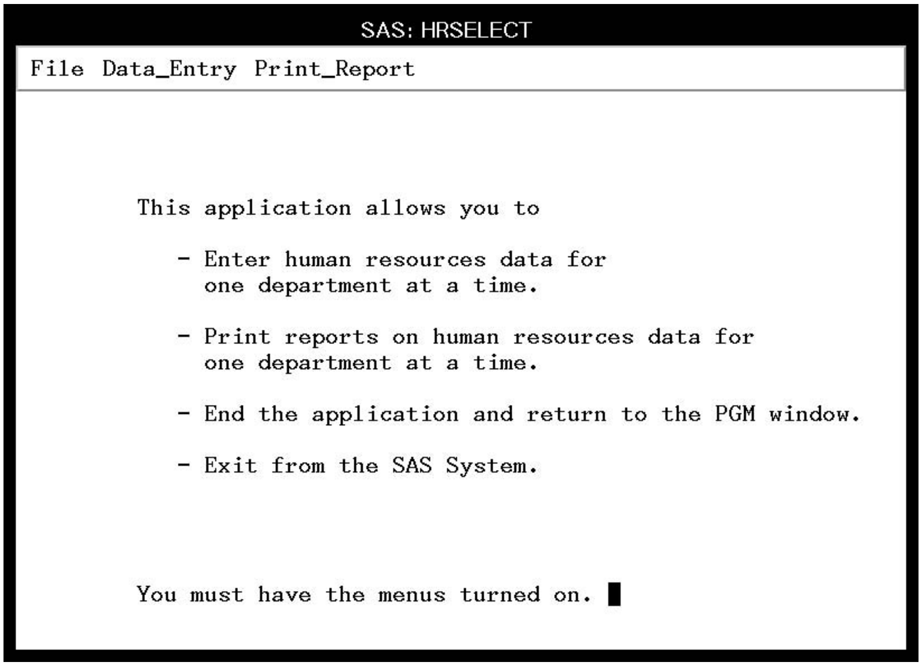
The WINDOW statement creates the HRSELECT window. MENU= associates the PROCLIB.MENUS.SELECT.PMENU entry with this window.

```
   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 window HRSELECT.

```
      display hrselect;
   run;
```

Primary window, HRSELECT.



Using a Data Entry Program

When the user selects **Data_Entry** from the menu bar in the HRSELECT window, a pull-down 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;
```

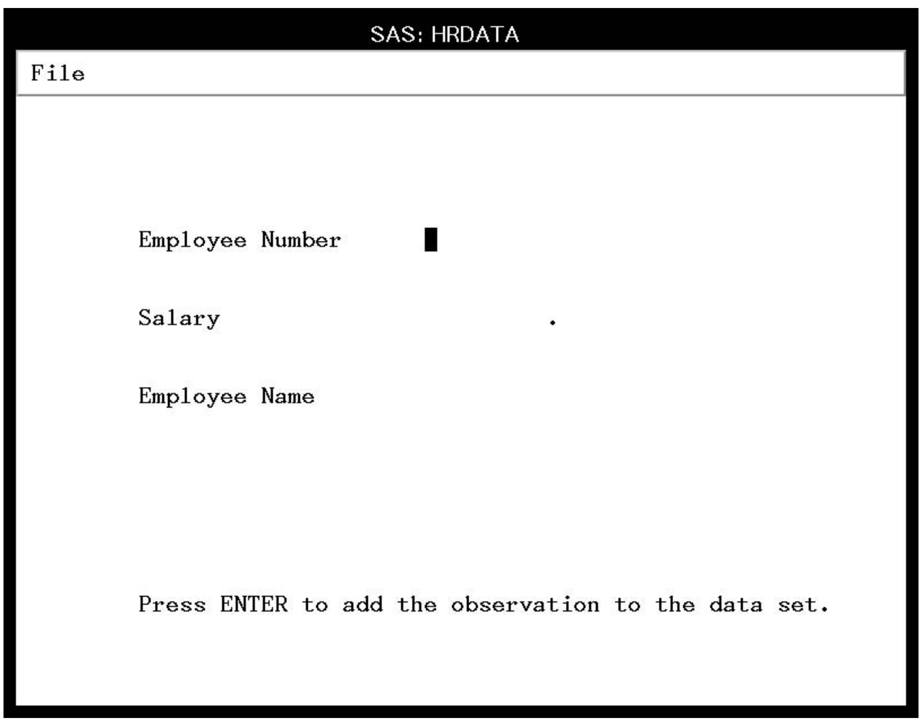
The %INCLUDE statement recalls the statements in the file HRWDW. The statements in HRWDW redisplay the primary window. See the HRSELECT window on page 698.

```
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;
```

Data entry window, HRDATA.



Printing a Program

When the user selects **`Print_Report`** from the menu bar, a pull-down 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;
```

The **xx**'s are changed to the appropriate department number by the CHANGE command in the SELECTION or DIALOG statement in the PROC PMENU step. PROC PRINT prints that data set.

```
libname proclib 'SAS-data-library';

proc print data=proclib.deptxx;
   title 'Information for deptxx';
run;
```

This PROC PRINTTO steps restores the default output destination. See Chapter 35, “The PRINTTO Procedure,” on page 771 for documentation on PROC PRINTTO.

```
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

Procedure features:
ITEM statement
MENU statement
Other features: SAS/AF software

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

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 opens.

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 that the menus are now associated with the FRAME.

Refer to *Getting Started with the FRAME Entry: Developing Object-Oriented Applications* for more information on SAS programming with FRAME entries.

CHAPTER 34

The 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

Output 34.1 illustrates the simplest kind of report that you can produce. The statements that produce the output follow. Example 1 on page 723 creates the data set EXPREV.

```
options nodate pageno=1 linesize=64 pagesize=60;


proc print data=exprev;
run;
```

Output 34.1 Simple Listing Report Produced with PROC PRINT

```
                    The SAS System                           1

      Obs     Region      State     Month     Expenses    Revenues

       1     Southern      GA       JAN95       2000         8000
       2     Southern      GA       FEB95       1200         6000
       3     Southern      FL       FEB95       8500        11000
       4     Northern      NY       FEB95       3000         4000
       5     Northern      NY       MAR95       6000         5000
       6     Southern      FL       MAR95       9800        13500
       7     Northern      MA       MAR95       1500         1000
```

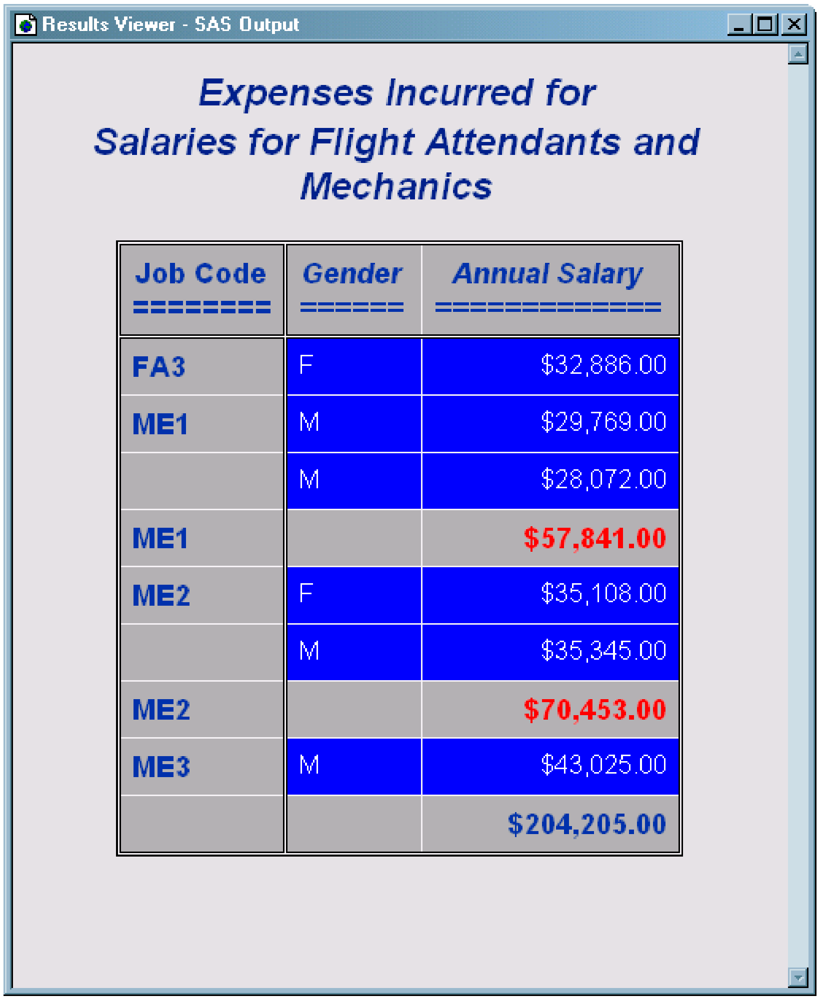
Customized Report

The following HTML report is a customized report that is produced by PROC PRINT using ODS. The statements that create this report

- 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 765.

Display 34.1 Customized Report Produced by PROC PRINT Using ODS


Results Viewer - SAS Output

Expenses Incurred for
Salaries for Flight Attendants and
Mechanics

Job Code ========	Gender ======	Annual Salary =============
FA3	F	$32,886.00
ME1	M	$29,769.00
	M	$28,072.00
ME1		$57,841.00
ME2	F	$35,108.00
	M	$35,345.00
ME2		$70,453.00
ME3	M	$43,025.00
		$204,205.00

Syntax: PRINT Procedure

Tip: Supports the Output Delivery System. See “Output Delivery System” on page 32 for details.

ODS Table Name: Print

Reminder: You can use the ATTRIB, FORMAT, LABEL, and WHERE statements. See Chapter 3, “Statements with the Same Function in Multiple Procedures,” on page 57 for details. You can also use any global statements. See “Global Statements” on page 18 for a list.

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>*;

To do this	Use this statement
Produce a separate section of the report for each BY group	BY
Identify observations by the formatted values of the variables that you list instead of by observation numbers	ID
Control page ejects that occur before a page is full	PAGEBY
Limit the number of sums that appear in the report	SUMBY
Total values of numeric variables	SUM
Select variables that appear in the report and determine their order	VAR

PROC PRINT Statement

PROC PRINT <*option(s)*>;

To do this	Use this option
Specify text for the HTML contents link to the output	CONTENTS=
Specify the input data set	DATA=
Control general format	
Write a blank line between observations	DOUBLE
Print the number of observations in the data set, in BY groups, or both, and specify explanatory text to print with the number	N=
Suppress the column in the output that identifies each observation by number	NOOBS
Specify a column header for the column that identifies each observation by number	OBS=
Round unformatted numeric values to two decimal places	ROUND
Control page format	
Format the rows on a page	ROWS=
Use each variable's formatted width as its column width on all pages	WIDTH=UNIFORM
Control column format	
Control the orientation of the column headings	HEADING=
Use variables' labels as column headings	LABEL or SPLIT=
Specify the split character, which controls line breaks in column headings	SPLIT=
Specify one or more style elements for the Output Delivery System to use for different parts of the report	STYLE
Determine the column width for each variable	WIDTH=

Options

CONTENTS=*link-text*

specifies the text for the links in the HTML contents file to the output produced by the PROC PRINT statement. For information on HTML output, see *SAS Output Delivery System: User's Guide*.

Restriction: CONTENTS= does not affect the HTML body file. It affects only the HTML contents file.

DATA=*SAS-data-set*
specifies the SAS data set to print.

Main discussion: "Input Data Sets" on page 19

DOUBLE
writes a blank line between observations.

Alias: D

Restriction: This option has no effect on the HTML output.

Featured in: Example 1 on page 723

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

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 dictates whether column headings are vertical or horizontal.
- If you use LABEL and at least one variable has a label, all headings are horizontal.

LABEL
uses variables' labels as column headings.

Alias: L

Default: If you omit LABEL, PROC PRINT uses the variable's name as the column heading even if the PROC PRINT step contains a LABEL statement. If a variable does not have a label, PROC PRINT uses the variable's name as the column heading.

Interaction: By default, if you specify LABEL and at least one variable has a label, PROC PRINT prints all column headings horizontally. Therefore, using LABEL may increase the number of pages of output. (Use HEADING=VERTICAL in the PROC PRINT statement to print vertical column headings.)

Interaction: 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=.

Tip: To create a blank column header for a variable, use this LABEL statement in your PROC PRINT step:

```
label variable-name='00'x;
```

See also: For information on using the LABEL statement to create temporary labels in procedures see Chapter 3, "Statements with the Same Function in Multiple Procedures," on page 57.

For information on using the LABEL statement in a DATA step to create permanent labels, see the section on statements in *SAS Language Reference: Dictionary*.

Featured in: Example 3 on page 731

Note: The SAS system option LABEL must be in effect in order for any procedure to use labels. For more information see the section on system options in *SAS Language Reference: Dictionary* △

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.

If you use the N option …	PROC PRINT …
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*.

Featured in: Example 2 on page 727 (alone)
Example 3 on page 731 (with a BY statement)
Example 4 on page 737 (with a BY statement and a SUM statement)

NOOBS
suppresses the observation number in the output.

Featured in: Example 3 on page 731

OBS="*column-header*"
specifies a column header for the column that identifies each observation by number.

Tip: OBS= honors the split character (see the discussion of SPLIT= on page 710).

Featured in: Example 2 on page 727

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 may 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 may 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 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 34.1 Three Methods of Summing Variables

```
               PROC PRINT without          PROC PRINT with
Actual Values  the ROUND option            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     ||------------
   .009      ||    3         0.01     ||    3         0.01     ||   0.0160000
  =====      ||             =====     ||             =====     ||------------
   .016      ||              0.02     ||              0.01     ||
             ||                       ||                       ||
==============================================================================
```

Notice that the sum produced without the ROUND option (.02) is closer to the actual result (0.16) than the sum produced with ROUND (0.01). However, the sum produced with ROUND reflects the numbers 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 may lead to unexpected results. △

ROWS=*page-format*

formats 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: Physical page size does not mean the same thing in HTML output as it does in traditional procedure output. Therefore, HTML output from PROC PRINT appears the same whether or not 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 also: “Page Layout” on page 720 for discussion of the default layout.

Featured in: Example 7 on page 754

SPLIT='*split-character*'

specifies the split character, which controls line breaks in column headers. It also uses labels as column headers. 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=

Interaction: You do not need to use both LABEL and SPLIT= because SPLIT= implies the use of labels.

Interaction: The OBS= option honors the split character. (See the discussion of OBS= on page 709.)

Featured in: Example 2 on page 727

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. △

STYLE

<(*location(s)*)>=<*style-element-name*><[*style-attribute-specification(s)*]>

specifies the style element to use for the specified locations in the report.

Note: You can use braces ({ and }) instead of square brackets ([and]). △

location

identifies the part of the report that the STYLE option affects. The following table shows the available locations and the other statements in which you can specify them.

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 instance, if you specify a blue background and a white foreground for all column headers in the PROC PRINT statement, and you specify a gray background for the column headers of a variable in the VAR statement, the background for that particular column header is gray, and the foreground is white (as specified in the PROC PRINT statement). △

Table 34.1 Specifying Locations in the STYLE Option

This location	Affects this part of the report	And can also be specified for individual items in this statement
BYLABEL	the label for the BY variable on the line containing the SUM totals	none
DATA	the cells of all columns	VAR ID SUM
GRANDTOTAL	the SUM line containing the grand totals for the whole report	SUM
HEADER	all column headers	VAR ID SUM

This location	Affects this part of the report	And can also be specified for individual items in this statement
N	N= table and contents	none
OBS	the data in the OBS column	none
OBSHEADER	the header of the OBS column	none
TABLE	the 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	the 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 34.2 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
TOTAL	TOT BYSUMLINE BYLINE BYSUM

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 PROC TEMPLATE.

When style elements are processed, more specific style elements override less specific style elements.

Default: The following table shows the default style element for each location.

Table 34.3 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:

BACKGROUND=
BACKGROUNDIMAGE=
BORDERCOLOR=
BORDERCOLORDARK=
BORDERCOLORLIGHT=
BORDERWIDTH=
CELLPADDING=
CELLSPACING=
FONT=*
FONT_FACE=*
FONT_SIZE=*
FONT_STYLE=*
FONT_WEIGHT=*
FONT_WIDTH=*
FOREGROUND=*
FRAME=
HTMLCLASS=
JUST=
OUTPUTWIDTH=
POSTHTML=
POSTIMAGE=
POSTTEXT=
PREHTML=
PREIMAGE=
PRETEXT=
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=	FONT_WIDTH=
BACKGROUND=	HREFTARGET=
BACKGROUNDIMAGE=	HTMLCLASS=
BORDERCOLOR=	JUST=
BORDERCOLORDARK=	NOBREAKSPACE=
BORDERCOLORLIGHT=	POSTHTML=
BORDERWIDTH=	POSTIMAGE=
CELLHEIGHT=	POSTTEXT=
CELLWIDTH=	PREHTML=
FLYOVER=	PREIMAGE=
FONT=	PRETEXT=
FONT_FACE=	PROTECTSPECIALCHARS=
FONT_SIZE=	TAGATTR=
FONT_STYLE=	URL=
FONT_WEIGHT=	VJUST=

For information about style attributes, see DEFINE STYLE statement in *SAS Output Delivery System: User's Guide*.

Restriction: This option affects all destinations except Listing and Output.

UNIFORM

See WIDTH=UNIFORM on page 715.

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

Tip: 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.

Tip: WIDTH=UNIFORM is the same as UNIFORM.

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.

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 may differ from one page to another.

Tip: Column width is affected not only by variable width but also by the length of column headings. Long column headings may lessen the usefulness of WIDTH=.

See also: For a discussion of default column widths, see "Column Width" on page 723.

BY Statement

Produces a separate section of the report for each BY group.

Main discussion: "BY" on page 58

Featured in: Example 3 on page 731, Example 4 on page 737, Example 5 on page 742, Example 6 on page 748, and Example 8 on page 761

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, 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*.

Options

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.

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 on page 761.)

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.

ID Statement

Identifies observations by using the formatted values of the variables that you list instead of by using observation numbers.

Featured in: Example 7 on page 754 and Example 8 on page 761

ID *variable(s)* </ STYLE <(*location(s)*)>
=<*style-element-name*><[*style-attribute-specification(s)*]>>;

Required Arguments

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.

Options

STYLE <(*location(s)*)>=<*style-element-name*><[*style-attribute-specification(s)*]>
specifies the style element to use for ID columns created with the ID statement. For information about the arguments of this option and how it is used, see STYLE on page 711 in the PROC PRINT 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.

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 on page 761.)

PAGEBY Statement

Controls page ejects that occur before a page is full.

Requirements: BY statement

Featured in: Example 3 on page 731

PAGEBY *BY-variable*;

Required Arguments

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 20.)

SUM Statement

Totals values of numeric variables.

Featured in: Example 4 on page 737, Example 5 on page 742, Example 6 on page 748, and Example 8 on page 761

SUM *variable(s)* </ STYLE <(*location(s)*)>
=<*style-element-name*><[*style-attribute-specification(s)*]>>;

Required Arguments

variable(s)
identifies the numeric variables to total in the report.

Option

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. For information about the arguments of this option and how it is used, see STYLE on page 711 in the PROC PRINT statement.

Tip: 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.

Tip: 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.

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 on page 737).

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 on page 742.)

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.

Requirements: BY statement

Featured in: Example 6 on page 748

SUMBY *BY-variable*;

Required Arguments

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.

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 those 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.

Featured in: Example 1 on page 723 and Example 8 on page 761

VAR *variable(s)* </ STYLE <(*location(s)*)>
=<*style-element-name*><[*style-attribute-specification(s)*]>>;

Required Arguments

variable(s)

identifies the variables to print. PROC PRINT prints the variables in the order that 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.

Option

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. For information about the arguments of this option and how it is used, see STYLE on page 711 in the PROC PRINT 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.

Results: Print Procedure

Procedure Output

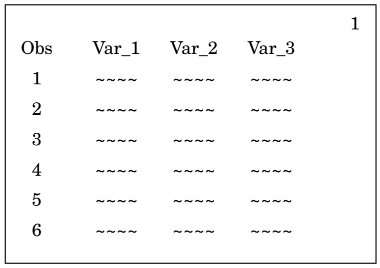
PROC PRINT always produces a printed report. You control the appearance of the report with statements and options. See "Examples: PRINT Procedure "on page 723 for a sampling of the types of reports that the procedure produces.

Page Layout

Observations

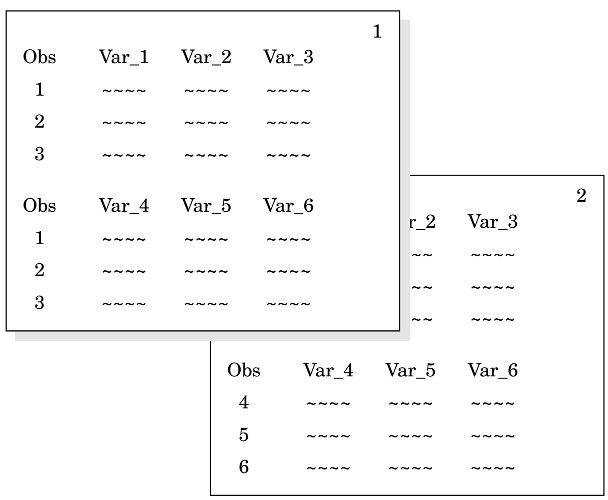
By default, PROC PRINT uses an identical layout for all observations on a page of output. First, it attempts to print observations on a single line (see Figure 34.2 on page 721).

Figure 34.2 Printing Observations on a Single Line



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 Figure 34.3 on page 721, 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 34.3 Splitting Observations into Multiple Sections on One Page



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 Figure 34.4 on page 722, 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 34.4 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= on page 710). △

Note: PROC PRINT may 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 dictates whether PROC PRINT prints column headings horizontally or vertically. Figure 34.2 on page 721, Figure 34.3 on page 721, and Figure 34.4 on page 722 all illustrate horizontal headings. Figure 34.5 on page 722 illustrates vertical headings.

Figure 34.5 Using Vertical Headings

```
                                1
       V      V      V
       a      a      a
 O     r      r      r
 b     _      _      _
 s     1      2      3

 1    ~~~~   ~~~~   ~~~~
 2    ~~~~   ~~~~   ~~~~
 3    ~~~~   ~~~~   ~~~~
 4    ~~~~   ~~~~   ~~~~
 5    ~~~~   ~~~~   ~~~~
 6    ~~~~   ~~~~   ~~~~
```

Note: If you use LABEL and at least one variable has a label, PROC PRINT prints all column headings horizontally unless you specify HEADING=VERTICAL. △

Column Width

By default, PROC PRINT uses a variable's formatted width as the column width. (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 linesize minus the length of all the ID variables, PROC PRINT may truncate the value. Consider the following situation:

- The linesize 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 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 may lessen the usefulness of WIDTH=. △

Examples: PRINT Procedure

Example 1: Selecting Variables to Print

Procedure features:

PROC PRINT statement options:

DOUBLE
STYLE

VAR statement

Other Features:

ODS HTML statement

This example

- selects three variables for the report
- uses variable labels as column headings
- double spaces between rows of the report.

Program: Creating a Listing Report

Set the SAS system options.

```
options nodate pageno=1 linesize=70 pagesize=60;
```

Create the input data set. EXPREV contains information about a company's monthly expenses and revenues for two regions of the United States.

```
data exprev;
   input Region $ State $ Month monyy5.
         Expenses Revenues;
   format month monyy5.;
   datalines;
Southern GA JAN95 2000  8000
Southern GA FEB95 1200  6000
Southern FL FEB95 8500 11000
Northern NY FEB95 3000  4000
Northern NY MAR95 6000  5000
Southern FL MAR95 9800 13500
Northern MA MAR95 1500  1000
;
```

Print the data set EXPREV. DOUBLE inserts a blank line between observations. (This 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 Month, State, and Expenses, in that order.

```
   var month state expenses;
```

Specify a title. The TITLE statement specifies a title for the report.

```
   title 'Monthly Expenses for Offices in Each State';
run;
```

Output: Listing

Output 34.2 Selecting Variables: Listing Output

By default, PROC PRINT identifies each observation by number under the column heading **Obs.**

```
              Monthly Expenses for Offices in Each State            1

                  Obs    Month    State    Expenses

                   1     JAN95     GA        2000

                   2     FEB95     GA        1200

                   3     FEB95     FL        8500

                   4     FEB95     NY        3000

                   5     MAR95     NY        6000

                   6     MAR95     FL        9800

                   7     MAR95     MA        1500
```

Program: Creating an HTML Report

You can easily create HTML output by adding ODS statements. In the following example, ODS statements were added to produce HTML output.

```
options nodate pageno=1 linesize=70 pagesize=60;
```

Create HTML output and specify the file to store the output in. The ODS HTML statement opens the HTML destination. FILE= specifies the external file that you want to contain the HTML output.

```
ods html file='your_file.html';

proc print data=exprev double;

   var month state expenses;
   title 'Monthly Expenses for Offices in Each State';

run;
```

Close the HTML destination. The ODS HTML CLOSE statement closes the HTML destination.

```
ods html close;
```

Output: HTML

Display 34.2 Selecting Variables: Default HTML Output

Monthly Expenses for Offices in Each State

Obs	Month	State	Expenses
1	JAN95	GA	2000
2	FEB95	GA	1200
3	FEB95	FL	8500
4	FEB95	NY	3000
5	MAR95	NY	6000
6	MAR95	FL	9800
7	MAR95	MA	1500

Program: Creating an HTML Report with the STYLE Option

You can go a step further and add more formatting to your HTML output. The following example uses the STYLE option to add shading to your HTML report.

```
options nodate pageno=1 linesize=70 pagesize=60;

ods html file='your_file.html';
```

Create stylized HTML output. The first STYLE option specifies that the column headers be written in white italic font.

The second STYLE option specifies that SAS change the color of the background of the observations column to red.

```
Proc Print data=exprev double
   style(HEADER) = {font_style=italic foreground = white}
   style(OBS) = {background=red};

   var month state expenses;

   title 'Monthly Expenses for Offices in Each State';

run;
```

Close the HTML destination. The ODS HTML CLOSE statement closes the HTML destination.

```
ods html close;
```

Output: HTML Output with Styles

Display 34.3 Selecting Variables: HTML Output Using Styles

Monthly Expenses for Offices in Each State

Obs	Month	State	Expenses
1	JAN95	GA	2000
2	FEB95	GA	1200
3	FEB95	FL	8500
4	FEB95	NY	3000
5	MAR95	NY	6000
6	MAR95	FL	9800
7	MAR95	MA	1500

Example 2: Customizing Text in Column Headers

Procedure features:

PROC PRINT statement options:

N
OBS=
SPLIT=
STYLE

VAR statement option:

STYLE

Other features:

LABEL statement

ODS PDF statement

Data set: EXPREV on page 724

This example

- customizes and underlines the text in column headings for variables
- customizes the column header for the column that identifies observations by number
- shows the number of observations in the report
- writes the values of Expenses with commas.

Program: Creating a Listing Report

```
options nodate pageno=1 linesize=70 pagesize=60;
```

Print the report and define the column headings. SPLIT= identifies the asterisk as the character that starts a new line in column headers. The N option prints the number of observations at the end of the report. OBS= specifies the column header for the column that identifies each observation by number. The split character (*) starts a new line in the column heading. Therefore, the equal signs (=) in the value of OBS= underline the column header.

```
proc print data=exprev split='*' n obs='Observation*Number*===========';
```

Select the variables to include in the report. The VAR statement creates columns for Month, State, and Expenses, in that order.

```
   var month state expenses;
```

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 SPLIT= in the PROC PRINT statement, the procedure uses labels for column headers. The split character (*) starts a new line in the column heading. Therefore, the equal signs (=) in the labels underline the column headers.

```
   label month='Month**====='
         state='State**====='
         expenses='Expenses**========';
```

Specify a title for the report, and format any variable containing numbers. The FORMAT statement assigns a format to use for Expenses in the report. The TITLE statement specifies a title.

```
   format expenses comma10.;
   title 'Monthly Expenses for Offices in Each State';
run;
```

Output: Listing

Output 34.3 Customizing Text in Column Headers: Listing Output

```
 Monthly Expenses for Offices in Each State            1

Observation    Month    State      Expenses
   Number
===========    =====    =====      ========

     1         JAN95     GA           2,000
     2         FEB95     GA           1,200
     3         FEB95     FL           8,500
     4         FEB95     NY           3,000
     5         MAR95     NY           6,000
     6         MAR95     FL           9,800
     7         MAR95     MA           1,500

                   N = 7
```

Program: Creating a PDF Report

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=70 pagesize=60;
```

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 your external file that contains the PDF output.

```
ods pdf file='your_file.pdf';
```

```
proc print data=exprev split='*' n obs='Observation*Number*===========';
   var month state expenses;
   label month='Month**====='
         state='State**====='
         expenses='Expenses**========';
                format expenses comma10.;

  title 'Monthly Expenses for Offices in Each State';
run;
```

Close the PDF destination. The ODS PDF CLOSE statement closes the PDF destination.

```
ods pdf close;
```

Output: PDF

Display 34.4 Customizing Text in Column Headers: Default PDF Output

Monthly Expenses for Offices in Each State

Observation Number ===========	Month =====	State =====	Expenses ========
1	JAN95	GA	2,000
2	FEB95	GA	1,200
3	FEB95	FL	8,500
4	FEB95	NY	3,000
5	MAR95	NY	6,000
6	MAR95	FL	9,800
7	MAR95	MA	1,500
N = 7			

Program: Creating a PDF Report with the STYLE Option

```
options nodate pageno=1 linesize=70 pagesize=60;

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 white.

```
proc print data=exprev split='*' n obs='Observation*Number*==========='
    style(N) = {font_style=italic background= blue}
    Style(HEADER OBS OBSHEADER) = {background=white};
```

Create stylized PDF output. The STYLE option changes the color of the cells containing data to gray.

```
   var month state expenses / style (DATA)= [ background = gray ] ;
   label month='Month**====='
         state='State**====='
         expenses='Expenses**========';
         format expenses comma10.;
```

```
   title 'Monthly Expenses for Offices in Each State';
run;
```

Close the PDF destination. The ODS PDF CLOSE statement closes the PDF destination.

```
ods pdf close;
```

Output: PDF Report with Styles

Display 34.5 Customizing Text in Column Headers: PDF Output Using Styles

Monthly Expenses for Offices in Each State

Observation Number ===========	Month =====	State =====	Expenses ========
1	JAN95	GA	2,000
2	FEB95	GA	1,200
3	FEB95	FL	8,500
4	FEB95	NY	3,000
5	MAR95	NY	6,000
6	MAR95	FL	9,800
7	MAR95	MA	1,500
N = 7			

Example 3: Creating Separate Sections of a Report for Groups of Observations

Procedure features:

PROC PRINT statement options:

LABEL
N=
NOOBS
STYLE

BY statement

PAGEBY statement

Other features:
SORT procedure
LABEL statement
ODS RTF statement

Data set: EXPREV on page 724

This example

- suppresses the printing of observation numbers at the beginning of each row
- presents the data for each state in a separate section of the report
- begins a new page for each region.

Program: Creating a Listing Report

```
options pagesize=60 pageno=1 nodate linesize=70;
```

Sort the EXPREV data set. PROC SORT sorts the observations by Region, State, and Month.

```
proc sort data=exprev;
   by region state month;
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 state: '
           noobs label;
```

Specify the variables to include in the report. The VAR statement creates columns for Month, Expenses, and Revenues, in that order.

```
   var month expenses revenues;
```

Create a separate section for each region of the state and specify page breaks for each BY group of Region. 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 Region changes.

```
   by region state;
   pageby region;
```

Establish the column headings. The LABEL statement associates a label with the variable Region 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 region='Sales Region';
```

Format the columns that contain numbers and specify a title. The FORMAT statement assigns a format to Expenses and Revenues for this report. The TITLE statement specifies a title.

```
   format revenues expenses comma10.;
   title 'Sales Figures Grouped by Region and State';
run;
```

Output: Listing

Output 34.4 Creating Separate Sections of a Report for Groups of Observations: Listing Output

```
              Sales Figures Grouped by Region and State             1

------------------- Sales Region=Northern State=MA -------------------

                  Month      Expenses      Revenues

                  MAR95         1,500         1,000

               Number of observations for the state: 1


------------------- Sales Region=Northern State=NY -------------------

                  Month      Expenses      Revenues

                  FEB95         3,000         4,000
                  MAR95         6,000         5,000

               Number of observations for the state: 2
```

```
              Sales Figures Grouped by Region and State             2

------------------- Sales Region=Southern State=FL -------------------

                  Month      Expenses      Revenues

                  FEB95         8,500        11,000
                  MAR95         9,800        13,500

               Number of observations for the state: 2


------------------- Sales Region=Southern State=GA -------------------

                  Month      Expenses      Revenues

                  JAN95         2,000         8,000
                  FEB95         1,200         6,000

               Number of observations for the state: 2
```

Program: Creating an RTF Report

```
options pagesize=60 pageno=1 nodate linesize=70;
```

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 your external file that contains the RTF output. The STARTPAGE=NO option specifies that no new pages be inserted within the PRINT procedure, even if new pages are requested by the procedure code.

```
ods rtf startpage=no file='your_file.rtf';
```

```
proc sort data=exprev;
by region state month;
run;
```

```
proc print data=exprev n='Number of observations for the state: '
    noobs label;
    var month expenses revenues;
    by region state;
    pageby region;
    label region='Sales Region';
    format revenues expenses comma10.;
title 'Sales Figures Grouped by Region
and State';
run;
```

Close the RTF destination. The ODS RTF CLOSE statement closes the RTF destination.

```
ods rtf close;
```

Output: RTF

Display 34.6 Creating Separate Sections of a Report for Groups of Observations: Default RTF Output

Sales Figures Grouped by Regionand State

Sales Region=Northern State=MA

Month	Expenses	Revenues
MAR95	1,500	1,000
Number of observations for the state: 1		

Sales Region=Northern State=NY

Month	Expenses	Revenues
FEB95	3,000	4,000
MAR95	6,000	5,000
Number of observations for the state: 2		

Sales Region=Southern State=FL

Month	Expenses	Revenues
FEB95	8,500	11,000
MAR95	9,800	13,500
Number of observations for the state: 2		

Sales Region=Southern State=GA

Month	Expenses	Revenues
JAN95	2,000	8,000
FEB95	1,200	6,000
Number of observations for the state: 2		

Program: Creating an RTF Report with the STYLE Option

```
options pagesize=60 pageno=1 nodate linesize=70;

ods rtf file='your_file.rtf';

proc sort data=exprev;
by region state month;
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 header for the variable MONTH be changed to white.

The third STYLE option specifies that the background color of the column header for the variable EXPENSES be changed to blue and the font color be changed to white.

The fourth STYLE option specifies that the background color of the column header for the variable REVENUES be changed to gray.

```
proc print data=exprev n='Number of observations for the state: '
           noobs label style(N) = {background=gray};
   var month / style(HEADER) = [background = white];
   var expenses / style(HEADER) = [background = blue foreground=white];
   var revenues / style(HEADER) = [background = gray];
    by region state;
    pageby region;
    label region='Sales Region';
    format revenues expenses comma10.;
title 'Sales Figures Grouped by Region
and State';
run;
```

```
ods rtf close;
```

Output: RTF with Styles

Display 34.7 Creating Separate Sections of a Report for Groups of Observations: RTF Output Using Styles

Sales Figures Grouped by Regionand State

Sales Region=Northern State=MA

Month	Expenses	Revenues
MAR95	1,500	1,000
Number of observations for the state: 1		

Sales Region=Northern State=NY

Month	Expenses	Revenues
FEB95	3,000	4,000
MAR95	6,000	5,000
Number of observations for the state: 2		

Sales Region=Southern State=FL

Month	Expenses	Revenues
FEB95	8,500	11,000
MAR95	9,800	13,500
Number of observations for the state: 2		

Sales Region=Southern State=GA

Month	Expenses	Revenues
JAN95	2,000	8,000
FEB95	1,200	6,000
Number of observations for the state: 2		

Example 4: Summing Numeric Variables with One BY Group

Procedure features:

PROC PRINT statement options:

N=

BY statement

SUM statement

Other features:

ODS MARKUP statement

SORT procedure

TITLE statement

#BYVAL specification

SAS system options:

BYLINE
NOBYLINE

Data set: EXPREV on page 724

This example

- 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.

Program: Creating a Listing Report

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 NOBYLINE, each BY group starts on a new page.

```
options nodate pageno=1 linesize=70 pagesize=60 nobyline;
```

Sort the data set. PROC SORT sorts the observations by Region.

```
proc sort data=exprev;
   by region;
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. 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
           n='Number of observations for the state: '
             'Number of observations for the data set: ';
```

Sum the values for the selected variables. The SUM statement alone sums the values of Expenses and Revenues for the entire data set. Because the PROC PRINT step contains a BY statement, the SUM statement also sums the values of Expenses and Revenues for each region that contains more than one observation.

```
   sum expenses revenues;
   by region;
```

Format the numeric values for a specified column. The FORMAT statement assigns the COMMA10. format to Expenses and Revenues for this report.

```
   format revenues expenses comma10.;
```

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 Region 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 'Revenue and Expense Totals for the
#byval(region) Region';
run;
```

Generate the default BY line. The SAS system option BYLINE resets the printing of the default BY line.

```
options byline;
```

Output: Listing

Output 34.5 Summing Numeric Variables with One BY Group: Listing Output

```
          Revenue and Expense Totals for the Northern Region         1

             State     Month      Expenses      Revenues

             NY        FEB95         3,000         4,000
             NY        MAR95         6,000         5,000
             MA        MAR95         1,500         1,000
             ------             ----------    ----------
             Region                 10,500        10,000

               Number of observations for the state: 3
```

```
          Revenue and Expense Totals for the Southern Region         2

             State     Month      Expenses      Revenues

             GA        JAN95         2,000         8,000
             GA        FEB95         1,200         6,000
             FL        FEB95         8,500        11,000
             FL        MAR95         9,800        13,500
             ------              ----------    ----------
             Region                 21,500        38,500
                                 ==========    ==========
                                    32,000        48,500

   Number of observations for the state: 4
Number of observations for the data set: 7
```

Program: Creating an XML File

The following example opens the MARKUP destination. The output file will contain only XML tagging unless you have a browser that reads XML.

```
options nodate pageno=1 linesize=70 pagesize=60 nobyline;
```

Produce output that is tagged with Extensible Markup Language (XML) tags and specify the file to store it in. The ODS MARKUP statement opens the MARKUP destination and creates a file containing output that is tagged with XML tags. The FILE= argument specifies your external file that contains the XML output.

```
ods markup file='your_file.xml';
```

```
proc sort data=exprev;
   by region;
run;
```

```
proc print data=exprev noobs
           n='Number of observations for the state: '
             'Number of observations for the data set: ';
```

```
   sum expenses revenues;
   by region;
```

```
   format revenues expenses comma10.;
```

```
   title 'Revenue and Expense Totals for the
#byval(region) Region';
run;
```

```
options byline;
```

Close the MARKUP destination. The ODS RTF CLOSE statement closes the MARKUP destination.

```
ods markup close;
```

Output: XML file

Output 34.6 Summing Numeric Variables with One BY Group: Partial XML Output Viewed with a Text Editor

```
<?xml version="1.0" encoding="windows-1252"?>

<odsxml>
<head>
<meta operator="user"/>
</head>
<body>
<proc name="Print">
<label name="IDX"/>
<title class="SystemTitle" toc-level="1">Revenue and Expense Totals for the Northern Region</title>
<branch name="Print" label="The Print Procedure" class="ContentProcName" toc-level="1">
<bygroup>
<branch name="ByGroup1" label="ByGroup1" class="ByContentFolder" toc-level="2">
<leaf name="Print" label="Data Set WORK.EXPREV" class="ContentItem" toc-level="3">
<output name="Print" label="Data Set WORK.EXPREV" clabel="Data Set WORK.EXPREV">
<output-object type="table" class="Table">

  <style>
    <border spacing="1" padding="7" rules="groups" frame="box"/>
  </style>
<colspecs columns="4">
<colgroup>
<colspec name="1" width="6" type="string"/>
<colspec name="2" width="5" type="string"/>
<colspec name="3" width="10" type="string"/>
<colspec name="4" width="10" type="string"/>
</colgroup>

... more lines of XML output ...

<row>
<data type="string" class="NoteContent" row="8" column="1" column-end="4">
  <style>
    <span columns="4"/>
  </style>
<value>Number of observations for the state: 4<br/>Number of observations for the data set: 7</value>
</data>
</row>
</output-body>
</output-object>
</output>
</leaf>
</bygroup>
</branch>
</branch>
</proc>
</body>
</odsxml>
```

Example 5: Summing Numeric Variables with Multiple BY Variables

Procedure features:
BY statement
SUM statement

Other features: SORT procedure

Data set: EXPREV on page 724

This example

- sums expenses and revenues for
  - each region
  - each state 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.

Program: Creating a Listing Report

```
options nodate pageno=1 linesize=70 pagesize=60;
```

Sort the data set. PROC SORT sorts the observations by Region and State.

```
proc sort data=exprev;
   by region state;
run;
```

Print the report, suppress the printing of observation numbers, and print the total number of observations for the selected variables. 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.

```
proc print data=exprev n noobs;
```

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 Expenses and Revenues for the entire data set. Because the program contains a BY statement, the SUM statement also sums the values of Expenses and Revenues for each BY group that contains more than one observation.

```
   by region state;
   sum expenses revenues;
```

Establish a label for a selected variable, format the values of specified variables, and create a title. The LABEL statement associates a label with the variable Region for the duration of the PROC PRINT step. The BY line at the beginning of each BY group uses the label. The FORMAT statement assigns a format to the variables Expenses and Revenues for this report. The TITLE statement specifies a title.

```
   label  region='Sales Region';
   format revenues expenses comma10.;
   title 'Revenue and Expense Totals for Each State and Region';
run;
```

Output: Listing

Output 34.7 Summing Numeric Variables with Multiple BY Variables: Listing Output

The report uses default column headers (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 name of a BY variable identifies the subtotals in the report.

PROC PRINT sums Expenses and Revenues 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 third BY group, where the sales region is **Southern** and the state is **FL**, Expenses and Revenues are summed only for the state because the next BY group is for the same region.

```
          Revenue and Expense Totals for Each State and Region        1

------------------- Sales Region=Northern State=MA -------------------

                   Month      Expenses      Revenues

                   MAR95         1,500         1,000

                              N = 1

------------------- Sales Region=Northern State=NY -------------------

                   Month      Expenses      Revenues

                   FEB95         3,000         4,000
                   MAR95         6,000         5,000
                  ------    ----------    ----------
                   State         9,000         9,000
                  Region        10,500        10,000

                              N = 2

------------------- Sales Region=Southern State=FL -------------------

                   Month      Expenses      Revenues

                   FEB95         8,500        11,000
                   MAR95         9,800        13,500
                  ------    ----------    ----------
                   State        18,300        24,500

                              N = 2

------------------- Sales Region=Southern State=GA -------------------

                   Month      Expenses      Revenues

                   JAN95         2,000         8,000
                   FEB95         1,200         6,000
                  ------    ----------    ----------
                   State         3,200        14,000
                  Region        21,500        38,500
                            ==========    ==========
                                32,000        48,500

                              N = 2
                        Total N = 7
```

Program: Creating an HTML Report

```
options nodate pageno=1 linesize=70 pagesize=60;
```

Produce HTML output and specify the file to store the output in. The ODS HTML statement opens the HTML destination and creates a file that contains HTML output. The FILE= argument specifies your external file that contains the HTML output.

```
ods html file='your_file.html';

proc sort data=exprev;
   by region state;
run;

proc print data=exprev n noobs;
   by region state;
   sum expenses revenues;

   label  region='Sales Region';
   format revenues expenses comma10.;
   title 'Revenue and Expense Totals for Each State and Region';
run;
```

Close the HTML destination. The ODS HTML CLOSE statement closes the HTML destination.

```
ods html close;
```

Output: HTML

Display 34.8 Summing Numeric Variables with Multiple BY Variables: Default HTML Output

Revenue and Expense Totals for Each State and Region

Sales Region=Northern State=MA

Month	Expenses	Revenues
MAR95	1,500	1,000
N = 1		

Sales Region=Northern State=NY

Month	Expenses	Revenues
FEB95	3,000	4,000
MAR95	6,000	5,000
State	**9,000**	**9,000**
Region	**10,500**	**10,000**
N = 2		

Sales Region=Southern State=FL

Month	Expenses	Revenues
FEB95	8,500	11,000
MAR95	9,800	13,500
State	**18,300**	**24,500**
N = 2		

Sales Region=Southern State=GA

Month	Expenses	Revenues
JAN95	2,000	8,000
FEB95	1,200	6,000
State	**3,200**	**14,000**
Region	**21,500**	**38,500**
	32,000	**48,500**
N = 2 Total N = 7		

Program: Creating an HTML Report with the STYLE Option

```
options nodate pageno=1 linesize=70 pagesize=60;

ods html file='your_file.html';

proc sort data=exprev;
   by region state;
run;

proc print data=exprev n noobs;
```

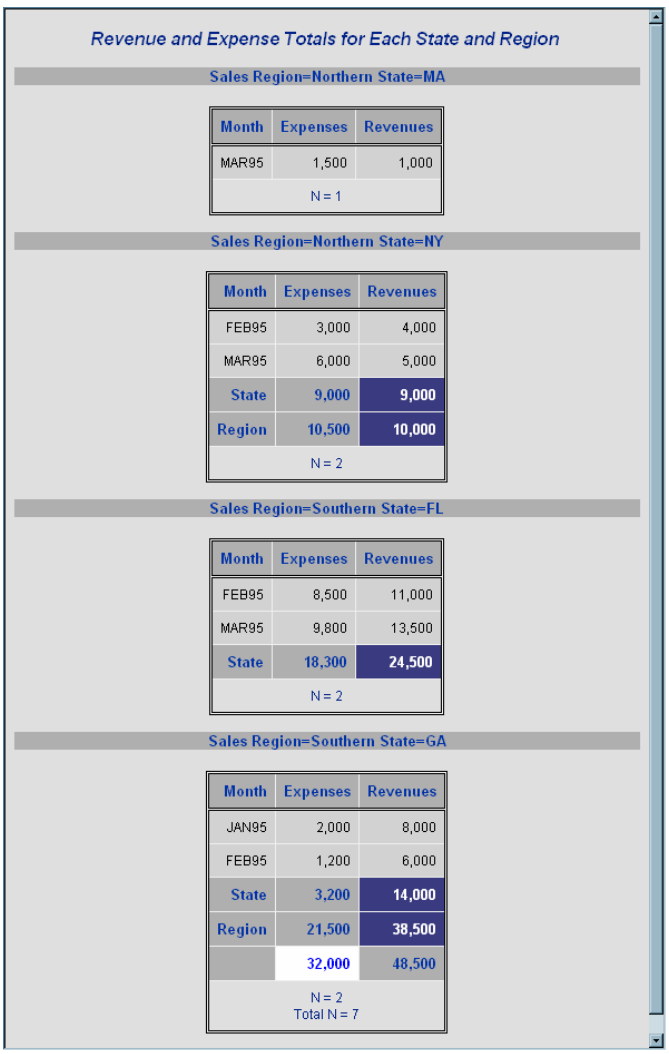
Create stylized HTML output. The STYLE option in the first SUM statement specifies that the background color of the cell containing the grand total for the variable EXPENSES be changed to white and the font color be changed to dark gray.

The STYLE option in the second SUM statement specifies that the background color of cells containing totals for the variable REVENUES be changed to blue and the font color be changed to white.

```
   by region state;
sum expenses / style(GRANDTOTAL) = [background =white foreground=blue];
sum revenues / style(TOTAL) = [background =dark gray foreground=white];

   label  region='Sales Region';
   format revenues expenses comma10.;
   title 'Revenue and Expense Totals for Each State and Region';
run;

ods html close;
```

Output: HTML with Styles

Display 34.9 Summing Numeric Variables with Multiple BY Variables: HTML Output Using Styles



Example 6: Limiting the Number of Sums in a Report

Features:

BY statement

SUM statement

SUMBY statement

Other features:

SORT procedure

LABEL statement

Data set: EXPREV on page 724

This example

- creates a separate section of the report for each combination of state and region
- sums expenses and revenues only for each region and for all regions, not for individual states.

Program: Creating a Listing Report

```
options nodate pageno=1 linesize=70 pagesize=60;
```

Sort the data set. PROC SORT sorts the observations by Region and State.

```
proc sort data=exprev;
   by region state;
run;
```

Print the report and remove the observation numbers. NOOBS suppresses the printing of observation numbers at the beginning of the rows.

```
proc print data=exprev noobs;
```

Sum the values for each region. The SUM and BY statements work together to sum the values of Revenues and Expenses for each BY group as well as for the whole report. The SUMBY statement limits the subtotals to one for each region.

```
   by region state;
   sum revenues expenses;
   sumby region;
```

Assign labels to specific variables. The LABEL statement associates a label with the variable Region for the duration of the PROC PRINT step. This label is used in the BY lines.

```
   label region='Sales Region';
```

Assign a format to the necessary variables and specify a title. The FORMAT statement assigns the COMMA10. format to Expenses and Revenues for this report.

```
   format revenues expenses comma10.;
   title 'Revenue and Expense Figures for Each Region';
run;
```

Output: Listing

Output 34.8 Limiting the Number of Sums in a Report: Listing Output

The report uses default column headers (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 name of a BY variable identifies the subtotals in the report.

```
            Revenue and Expense Figures for Each Region            1

------------------- Sales Region=Northern State=MA -------------------

                  Month      Expenses      Revenues

                  MAR95         1,500         1,000


------------------- Sales Region=Northern State=NY -------------------

                  Month      Expenses      Revenues

                  FEB95         3,000         4,000
                  MAR95         6,000         5,000
                 ------    ----------    ----------
                 Region        10,500        10,000


------------------- Sales Region=Southern State=FL -------------------

                  Month      Expenses      Revenues

                  FEB95         8,500        11,000
                  MAR95         9,800        13,500


------------------- Sales Region=Southern State=GA -------------------

                  Month      Expenses      Revenues

                  JAN95         2,000         8,000
                  FEB95         1,200         6,000
                 ------    ----------    ----------
                 Region        21,500        38,500
                           ==========    ==========
                               32,000        48,500
```

Program: Creating a PostScript file

```
options nodate pageno=1 linesize=70 pagesize=60;
```

Produce PostScript output and specify the file to store the output in. The ODS PS statement opens the PS destination and creates a file that contains PostScript output. The FILE= argument specifies your external file that contains the PostScript output.

```
ods ps file='your_file.ps';
```

```
proc sort data=exprev;
   by region state;
run;
```

```
proc print data=exprev noobs;
```

```
by region state;
   sum revenues expenses;
   sumby region;
```

```
   label region='Sales Region';
```

```
   format revenues expenses comma10.;
   title 'Revenue and Expense Figures for Each Region';
run;
```

Close the PS destination. The ODS PS CLOSE statement closes the PS destination.

```
ods ps close;
```

Output: PostScript

Display 34.10 Limiting the Number of Sums in a Report: PostScript Output

Revenue and Expense Figures for Each Region

Sales Region=Northern State=MA

Month	Expenses	Revenues
MAR95	1,500	1,000

Sales Region=Northern State=NY

Month	Expenses	Revenues
FEB95	3,000	4,000
MAR95	6,000	5,000
Region	**10,500**	**10,000**

Sales Region=Southern State=FL

Month	Expenses	Revenues
FEB95	8,500	11,000
MAR95	9,800	13,500

Sales Region=Southern State=GA

Month	Expenses	Revenues
JAN95	2,000	8,000
FEB95	1,200	6,000
Region	**21,500**	**38,500**
	32,000	**48,500**

Program: Creating a PostScript Report with the STYLE Option

```
options nodate pageno=1 linesize=70 pagesize=60;


ods ps file='your_file.ps';


proc sort data=exprev;
   by region state;
run;


proc print data=exprev noobs;


   by region state;
```

Create stylized PostScript output. The STYLE option in the first SUM statement specifies that the background color of cells containing totals for the variable REVENUES 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 EXPENSES variable be changed to white and the font color be changed to dark gray.

```
   sum revenues / style(TOTAL) = [background =blue foreground=white];
   sum expenses / style(GRANDTOTAL) = [background =white foreground=dark gray];


   label region='Sales Region';


   format revenues expenses comma10.;
   title 'Revenue and Expense Figures for Each Region';
run;


ods ps close;
```

Output: PostScript with Styles

Display 34.11 Limiting the Number of Sums in a Report: PostScript Output Using Styles

Revenue and Expense Figures for Each Region

Sales Region=Northern State=MA

Month	Expenses	Revenues
MAR95	1,500	1,000

Sales Region=Northern State=NY

Month	Expenses	Revenues
FEB95	3,000	4,000
MAR95	6,000	5,000
State	**9,000**	**9,000**
Region	**10,500**	**10,000**

Sales Region=Southern State=FL

Month	Expenses	Revenues
FEB95	8,500	11,000
MAR95	9,800	13,500
State	**18,300**	**24,500**

Sales Region=Southern State=GA

Month	Expenses	Revenues
JAN95	2,000	8,000
FEB95	1,200	6,000
State	**3,200**	**14,000**
Region	**21,500**	**38,500**
	32,000	**48,500**

Example 7: Controlling the Layout of a Report with Many Variables

Procedure features:

PROC PRINT statement options:

ROWS=

ID statement options:

STYLE

Other features:

ODS RTF statement

SAS data set options:

OBS=

This example shows two ways of printing a data set with a large number of variables: one is the default, and the other uses ROWS=. For detailed explanations of the layouts of these two reports, see the ROWS= option on page 710 and see “Page Layout” on page 720.

These reports use a pagesize of 24 and a linesize of 64 to help illustrate the different layouts.

Note: When the two reports are written as HTML output, they do not differ. △

Program: Creating a Listing Report

```
options nodate pageno=1 linesize=64 pagesize=24 ;
```

Create the EMPDATA data set. The data set EMPDATA contains personal and job-related information about a company’s employees. A DATA step on page 1385 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     15SEP1948      07JUN1975    203/781-1255
1653    Alexander  Susan     Bridgeport   CT
F       ME2        35108     18OCT1952      12AUG1978    203/675-7715

. . . more lines of data . . .

1407    Grant      Daniel    Mt. Vernon   NY
M       PT1        68096     26MAR1957      21MAR1978    914/468-1616
1114    Green      Janice    New York     NY
F       TA2        32928     21SEP1957      30JUN1975    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 Example 7 on page 754.

```
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 Example 7 on page 754.

```
proc print data=empdata(obs=12) rows=page;
   id idnumber;
   title 'Personnel Data';
run;
```

Output: Listing

Output 34.9 Default Layout for a Report with Many Variables: Listing Output

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 headers at a change in capitalization if the entire name does not fit in the column. Compare, for example, the column headers for LastName (which fits in the column) and FirstName (which does not fit in the column).

```
                          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    15SEP48    07JUN75    203/781-1255
  1653     ME2      35108    18OCT52    12AUG78    203/675-7715
  1400     ME1      29769    08NOV55    19OCT78    212/586-0808
  1350     FA3      32886    03SEP53    01AUG78    718/383-1549
  1401     TA3      38822    16DEC38    20NOV73    201/732-8787
  1499     ME3      43025    29APR42    10JUN68    201/812-5665
  1101     SCP      18723    09JUN50    04OCT78    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    02APR49    13FEB69    203/781-1777
 1402     TA2      32615    20JAN51    05DEC78    718/384-2849
 1479     TA3      38785    25DEC56    08OCT77    718/384-8816
 1403     ME1      28072    31JAN57    24DEC79    203/675-3434
 1739     PT1      66517    28DEC52    30JAN79    212/587-1247
```

Output 34.10 Layout Produced by the ROWS=PAGE Option: Listing Output

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.

```
                         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
 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                         2

  Id      Job
Number    Code    Salary      Birth      Hired     HomePhone

 1919     TA2      34376    15SEP48    07JUN75    203/781-1255
 1653     ME2      35108    18OCT52    12AUG78    203/675-7715
 1400     ME1      29769    08NOV55    19OCT78    212/586-0808
 1350     FA3      32886    03SEP53    01AUG78    718/383-1549
 1401     TA3      38822    16DEC38    20NOV73    201/732-8787
 1499     ME3      43025    29APR42    10JUN68    201/812-5665
 1101     SCP      18723    09JUN50    04OCT78    212/586-8060
 1333     PT2      88606    02APR49    13FEB69    203/781-1777
 1402     TA2      32615    20JAN51    05DEC78    718/384-2849
 1479     TA3      38785    25DEC56    08OCT77    718/384-8816
 1403     ME1      28072    31JAN57    24DEC79    203/675-3434
 1739     PT1      66517    28DEC52    30JAN79    212/587-1247
```

Program: Creating an RTF Report

```
options nodate pageno=1 linesize=64 pagesize=24;


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     15SEP1948      07JUN1975    203/781-1255
1653    Alexander   Susan     Bridgeport   CT
F       ME2         35108     18OCT1952      12AUG1978    203/675-7715

. . . more lines of data . . .

1407    Grant       Daniel    Mt. Vernon   NY
M       PT1         68096     26MAR1957      21MAR1978    914/468-1616
1114    Green       Janice    New York     NY
F       TA2         32928     21SEP1957      30JUN1975    212/588-1092
;
```

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 your external file that contains the RTF output.

```
ods rtf file='your_file.rtf';
```

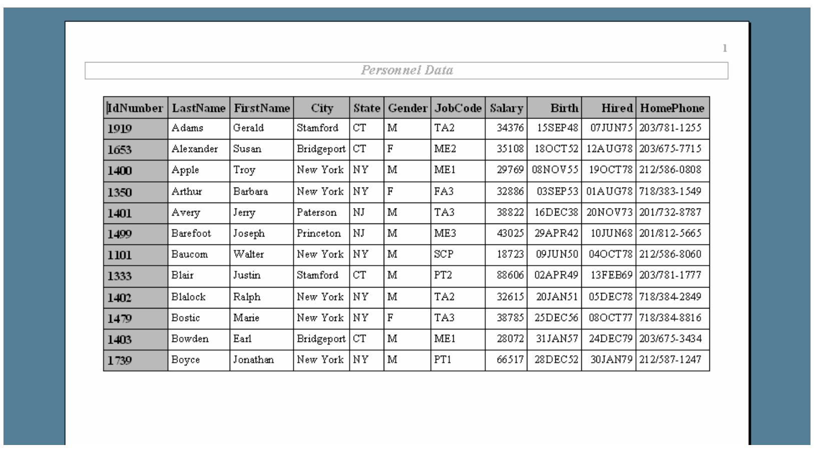
```
proc print data=empdata(obs=12);
   id idnumber;
   title 'Personnel Data';
run;
```

Close the RTF destination. The ODS RTF CLOSE statement closes the RTF destination.

```
ods rtf close;
```

Output: RTF

Display 34.12 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	15SEP48	07JUN75	203/781-1255
1653	Alexander	Susan	Bridgeport	CT	F	ME2	35108	18OCT52	12AUG78	203/675-7715
1400	Apple	Troy	New York	NY	M	ME1	29769	08NOV55	19OCT78	212/586-0808
1350	Arthur	Barbara	New York	NY	F	FA3	32886	03SEP53	01AUG78	718/383-1549
1401	Avery	Jerry	Paterson	NJ	M	TA3	38822	16DEC38	20NOV73	201/732-8787
1499	Barefoot	Joseph	Princeton	NJ	M	ME3	43025	29APR42	10JUN68	201/812-5665
1101	Baucom	Walter	New York	NY	M	SCP	18723	09JUN50	04OCT78	212/586-8060
1333	Blair	Justin	Stamford	CT	M	PT2	88606	02APR49	13FEB69	203/781-1777
1402	Blalock	Ralph	New York	NY	M	TA2	32615	20JAN51	05DEC78	718/384-2849
1479	Bostic	Marie	New York	NY	F	TA3	38785	25DEC56	08OCT77	718/384-8816
1403	Bowden	Earl	Bridgeport	CT	M	ME1	28072	31JAN57	24DEC79	203/675-3434
1739	Boyce	Jonathan	New York	NY	M	PT1	66517	28DEC52	30JAN79	212/587-1247

Program: Creating an RTF Report with the STYLE Option

```
options nodate pageno=1 linesize=64 pagesize=24;

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     15SEP1948      07JUN1975    203/781-1255
1653    Alexander  Susan     Bridgeport   CT
F       ME2        35108     18OCT1952      12AUG1978    203/675-7715

. . . more lines of data . . .
```

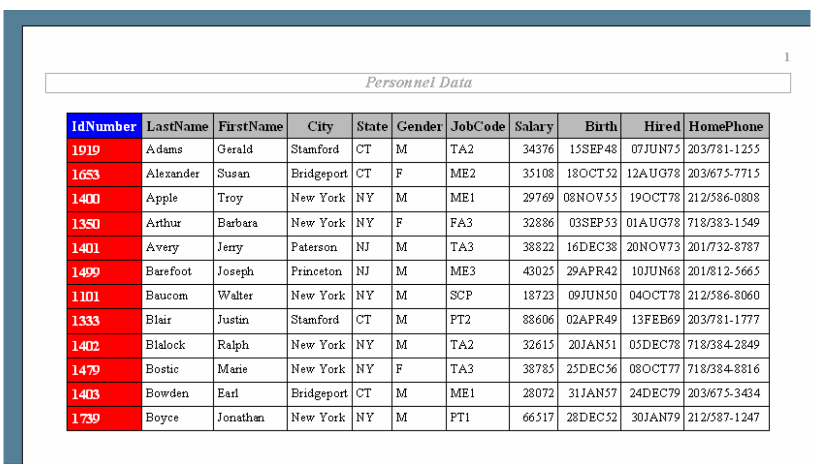
```
1407    Grant     Daniel    Mt. Vernon   NY
M       PT1       68096     26MAR1957      21MAR1978    914/468-1616
1114    Green     Janice    New York     NY
F       TA2       32928     21SEP1957      30JUN1975    212/588-1092
;

ods rtf file='your_file.rtf';

proc print data=empdata(obs=12);
```

Create stylized output for Microsoft Word.

```
id idnumber / style(DATA) =
            {background = red foreground = white}
            style(HEADER) =
            {background = blue foreground = white};

   title 'Personnel Data';
run;

ods rtf close;
```

Output: RTF with Styles

Display 34.13 Layout for a Report with Many Variables: RTF Output Using Styles



1

Personnel Data

IdNumber	LastName	FirstName	City	State	Gender	JobCode	Salary	Birth	Hired	HomePhone
1919	Adams	Gerald	Stamford	CT	M	TA2	34376	15SEP48	07JUN75	203/781-1255
1653	Alexander	Susan	Bridgeport	CT	F	ME2	35108	18OCT52	12AUG78	203/675-7715
1400	Apple	Troy	New York	NY	M	ME1	29769	08NOV55	19OCT78	212/586-0808
1350	Arthur	Barbara	New York	NY	F	FA3	32886	03SEP53	01AUG78	718/383-1549
1401	Avery	Jerry	Paterson	NJ	M	TA3	38822	16DEC38	20NOV73	201/732-8787
1499	Barefoot	Joseph	Princeton	NJ	M	ME3	43025	29APR42	10JUN68	201/812-5665
1101	Baucom	Walter	New York	NY	M	SCP	18723	09JUN50	04OCT78	212/586-8060
1333	Blair	Justin	Stamford	CT	M	PT2	88606	02APR49	13FEB69	203/781-1777
1402	Blalock	Ralph	New York	NY	M	TA2	32615	20JAN51	05DEC78	718/384-2849
1479	Bostic	Marie	New York	NY	F	TA3	38785	25DEC56	08OCT77	718/384-8816
1403	Bowden	Earl	Bridgeport	CT	M	ME1	28072	31JAN57	24DEC79	203/675-3434
1739	Boyce	Jonathan	New York	NY	M	PT1	66517	28DEC52	30JAN79	212/587-1247

Example 8: Creating a Customized Layout with BY Groups and ID Variables

Procedure features:
BY statement
ID statement
SUM statement
VAR statement

Other features:
SORT procedure

Data set: EMPDATA on page 755

This customized report

- selects variables to include in the report and controls their order
- 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

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 headers. SPLIT= identifies the asterisk as the character that starts a new line in column headers.

```
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 'Expenses Incurred for';
   title2 'Salaries for Flight Attendants and Mechanics';
run;
```

Output: Listing

Output 34.11 Creating a Customized Layout with BY Groups and ID Variables: Listing Output

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.

```
                  Expenses Incurred for                         1
         Salaries for Flight Attendants and Mechanics

            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;
```

Produce HTML output and specify the file to store the output in. The ODS HTML statement opens the HTML destination and creates a file that contains HTML output. The FILE= argument specifies your 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*=============';
```

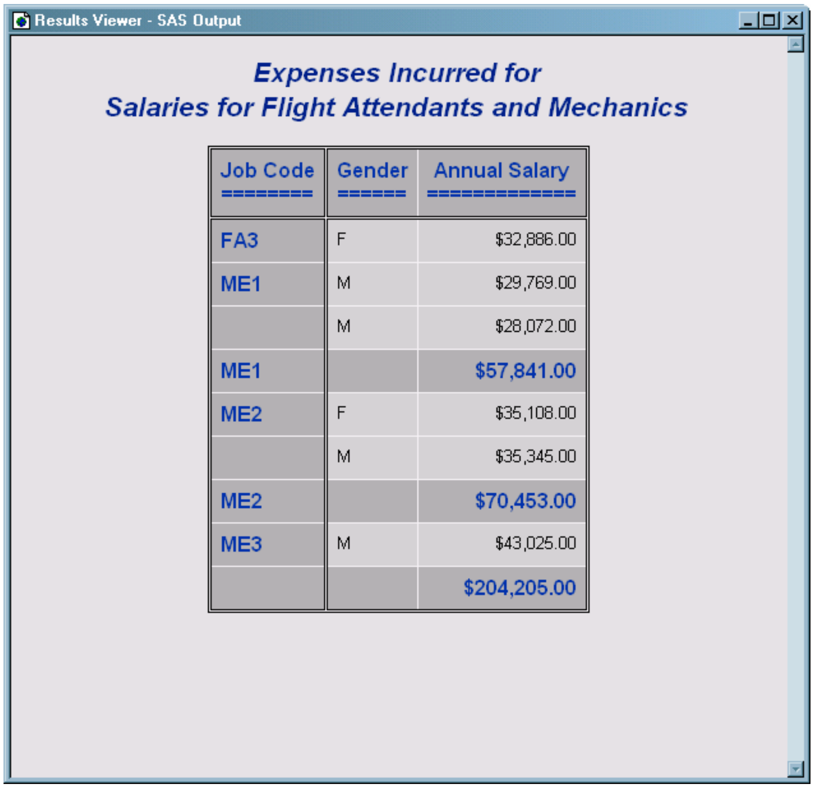
```
   format salary dollar11.2;
   where jobcode contains 'FA' or jobcode contains 'ME';
   title 'Expenses Incurred for';
   title2 'Salaries for Flight Attendants and Mechanics';
run;
```

Close the HTML destination. The ODS HTML CLOSE statement closes the HTML destination.

```
ods html close;
```

Output: HTML

Display 34.14 Creating a Customized Layout with BY Groups and ID Variables: Default HTML Output



Expenses Incurred for
Salaries for Flight Attendants and Mechanics

Job Code ========	Gender ======	Annual Salary =============
FA3	F	$32,886.00
ME1	M	$29,769.00
	M	$28,072.00
ME1		$57,841.00
ME2	F	$35,108.00
	M	$35,345.00
ME2		$70,453.00
ME3	M	$43,025.00
		$204,205.00

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';
```

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) =
                                  {font_style=italic}
                                  style(DATA) =
                                  {background=blue foreground = white};
```

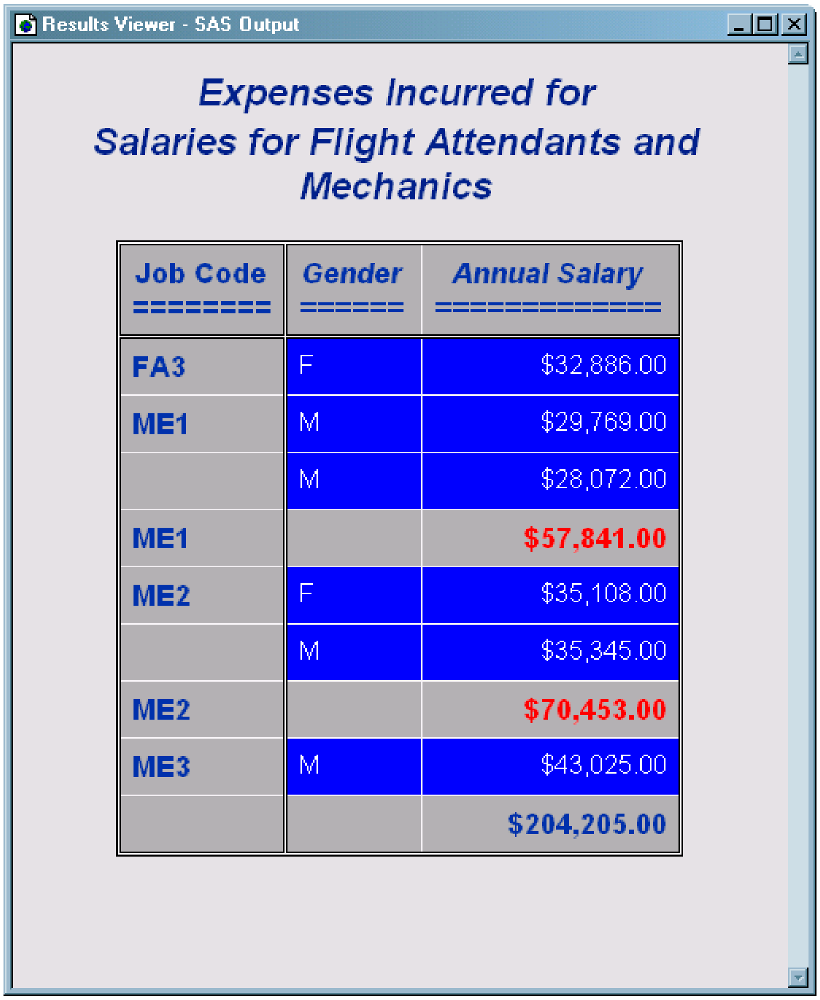
```
 id jobcode;
   by jobcode;
   var gender salary;
```

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)= [foreground=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;


ods html close;
```

Output: HTML with Styles

Display 34.15 Creating a Customized Layout with BY Groups and ID Variables: HTML Output Using Styles



Expenses Incurred for
Salaries for Flight Attendants and Mechanics

Job Code ========	Gender ======	Annual Salary =============
FA3	F	$32,886.00
ME1	M	$29,769.00
	M	$28,072.00
ME1		$57,841.00
ME2	F	$35,108.00
	M	$35,345.00
ME2		$70,453.00
ME3	M	$43,025.00
		$204,205.00

Example 9: Printing All the Data Sets in a SAS Library

Features:

Macro facility

DATASETS procedure

PRINT procedure

Data set: EXPREV on page 724 and LIST

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 Guide to Macro Processing, Version 6, Second Edition*.

Program

```
libname printlib 'SAS-data-library'
options nodate pageno=1 linesize=80 pagesize=60;
```

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=work memtype=data nolist;
   copy out=printlib;
      select list exprev;
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

Output 34.12 Printing All the Data Sets in a SAS Library: Listing Output

```
                     Data Set printlib.EXPREV                      1

          Region      State     Month     Expenses    Revenues

         Northern      MA       MAR95       1500         1000
         Northern      NY       FEB95       3000         4000
         Northern      NY       MAR95       6000         5000
         Southern      FL       FEB95       8500        11000
         Southern      FL       MAR95       9800        13500
         Southern      GA       JAN95       2000         8000
         Southern      GA       FEB95       1200         6000
```

```
                        Data Set printlib.LIST                       2

Name                 Street                City          State    Zip

Gabrielli, Theresa   24 Ridgetop Rd.       Westboro       MA     01581
Clayton, Aria        314 Bridge St.        Hanover        NH     03755
Dix, Martin L.       4 Shepherd St.        Norwich        VT     05055
Slater, Emily C.     2009 Cherry St.       York           PA     17407
Ericson, Jane        211 Clancey Court     Chapel Hill    NC     27514
An, Ing              95 Willow Dr.         Charlotte      NC     28211
Jacobson, Becky      7 Lincoln St.         Tallahassee    FL     32312
Misiewicz, Jeremy    43-C Lakeview Apts.   Madison        WI     53704
Ahmadi, Hafez        5203 Marston Way      Boulder        CO     80302
Archuleta, Ruby      Box 108               Milagro        NM     87429
```

CHAPTER
35

The PRINTTO Procedure

Overview: PRINTTO Procedure

The PRINTTO procedure defines 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. See Table 35.1 on page 771. You can store the SAS log or procedure output in an external file or in a SAS catalog entry. With additional programming, you can use SAS output as input data within the same job.

Table 35.1 Default Destinations for SAS Log and Procedure Output

Method of running the SAS System	SAS log destination	Procedure output destination
windowing environment	the LOG window	the OUTPUT window
interactive line mode	the display monitor (as statements are entered)	the 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 appropriate SAS Companion or technical report. △

Syntax: PRINTTO Procedure

See: PRINTTO Procedure in the documentation for your operating environment.

PROC PRINTTO <*option(s)*>;

PROC PRINTTO Statement

Tip: To reset the destination for the SAS log and procedure output to the default, use the PROC PRINTTO statement without options.

Tip: To route the SAS log and procedure output to the same file, specify the same file with both the LOG= and PRINT= options.

Restriction: To route SAS log and procedure output directly to a printer, you must use a FILENAME statement with the PROC PRINTTO statement. See Example 4 on page 785.

PROC PRINTTO <*option(s)*>;

To do this	Use this option
provide a description for a SAS log or procedure output stored in a SAS catalog entry	LABEL=
route the SAS log to a permanent external file or SAS catalog entry	LOG=
combine the SAS log and procedure output into a single file	LOG= and PRINT= with same destination
replace the file instead of appending to it	NEW
route procedure output to a permanent external file or SAS catalog entry or printer.	PRINT=

Without Options

Using a PROC PRINTTO statement with no options

- 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.

Featured in: Example 1 on page 776 and Example 2 on page 779

Options

LABEL='*description*'
provides a description for a catalog entry that contains a SAS log or procedure output.

Range: 1 to 256 characters

Interaction: Use the LABEL= option only when you specify a catalog entry as the value for the LOG= or the PRINT= option.

Featured in: Example 2 on page 779

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.

log-filename
is an unquoted alphanumeric text string. SAS creates a log that uses *log-filename*.log as the log filename.

Operating Environment Information: For more information about using *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 data library and SAS catalog specified.

catalog.entry<.LOG>
a SAS catalog entry stored in the specified SAS catalog in the default SAS data 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.

Tip: 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.

Tip: 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.)

Interaction: The SAS log and procedure output cannot be routed to the same catalog entry at the same time.

Interaction: The NEW option replaces the existing contents of a file with the new log. Otherwise, the new log is appended to the file.

Interaction: To route the SAS log and procedure output to the same file, specify the same file with both the LOG= and PRINT= options.

Interaction: 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.

Featured in: Example 1 on page 776, Example 2 on page 779, and Example 3 on page 782

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.

Featured in: Example 1 on page 776, Example 2 on page 779, and Example 3 on page 782

PRINT= PRINT | *file-specification* | *SAS-catalog-entry*

routes procedure output to one of three locations:

PRINT

routes procedure output to its default destination. 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. It is one of the following:

'*external-file*'

the name of an external file specified in quotation marks.

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: See your operating environment documentation for additional information about *file-specification* for the PRINT option. △

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 data library and SAS catalog specified.

catalog.entry<.OUTPUT>

a SAS catalog entry stored in the specified SAS catalog in the default SAS data 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.

Aliases: FILE=, NAME=

Default: PRINT

Interaction: The procedure output and the SAS log cannot be routed to the same catalog entry at the same time.

Interaction: 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.

Interaction: To route the SAS log and procedure output to the same file, specify the same file with both the LOG= and PRINT= options.

Interaction: 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.

Featured in: Example 3 on page 782

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 to 99, integer only.

Tip: You can define this fileref yourself; however, some operating systems predefine certain filerefs in this form.

Concepts: PRINTTO Procedure

Page Numbering

- When the SAS system option NUMBER 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 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 on page 785.

For more information see the FILENAME statement in *SAS Language Reference: Dictionary*.

Operating Environment Information: For examples of printer names, see the documentation for your operating system. △

Examples: PRINTTO Procedure

Example 1: Routing to External Files

Procedure features:
- PRINTTO statement:
  - Without options
  - Options:
    - LOG=
    - NEW
    - PRINT=

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

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page. The SOURCE option writes lines of source code to the default destination for the SAS log.

```
options nodate pageno=1 linesize=80 pagesize=60 source;
```

Route the SAS log to an external file. PROC PRINTTO uses the LOG= option to route the SAS log to an external file. By default, this log is appended to the current contents of **log-file**.

```
proc printto log='log-file';
   run;
```

Create the NUMBERS data set. The DATA step uses list input to create the NUMBERS data set.

```
data numbers;
   input x y z;
   datalines;
 14.2   25.2   96.8
 10.8   51.6   96.8
  9.5   34.2  138.2
  8.8   27.6   83.2
 11.5   49.4  287.0
  6.3   42.0  170.7
;
```

Route the procedure output to an external file. PROC PRINTTO routes output to an external file. Because NEW is specified, any output written to **`output-file`** will overwrite the file's current contents.

```
proc printto print='output-file' new;
run;
```

Print the NUMBERS data set. The PROC PRINT output is written to the specified external file.

```
proc print data=numbers;
   title 'Listing of NUMBERS Data Set';
run;
```

Reset the SAS log and procedure output destinations to default. PROC PRINTTO routes subsequent logs and procedure output to their default destinations and closes both of the current files.

```
proc printto;
run;
```

Log

Output 35.1 Portion of Log Routed to the Default Destination

```
1          options nodate pageno=1 linesize=80 pagesize=60 source;
2          proc printto log='log-file';
3          run;
```

Output 35.2 Portion of Log Routed to an External File

```
5
6          data numbers;
7             input x y z;
8             datalines;

NOTE: The data set WORK.NUMBERS has 6 observations and 3 variables.
NOTE: DATA statement used:
      real time           0.00 seconds
      cpu time            0.00 seconds


15         ;
16         proc printto print='output-file' new;
16
17         run;

NOTE: PROCEDURE PRINTTO used:
      real time           0.00 seconds
      cpu time            0.00 seconds


18
19         proc print data=numbers;
20            title 'Listing of NUMBERS Data Set';
21         run;

NOTE: The PROCEDURE PRINT printed page 1.
NOTE: PROCEDURE PRINT used:
      real time           0.00 seconds
      cpu time            0.00 seconds


22
23         proc printto;
24         run;
```

Output

Output 35.3 Procedure Output Routed to an External File

```
                           Listing of NUMBERS Data Set                         1

                           OBS      x        y        z

                            1      14.2     25.2     96.8
                            2      10.8     51.6     96.8
                            3       9.5     34.2    138.2
                            4       8.8     27.6     83.2
                            5      11.5     49.4    287.0
                            6       6.3     42.0    170.7
```

Example 2: Routing to SAS Catalog Entries

Procedure features:

PRINTTO statement:

- Without options
- Options:
  - LABEL=
  - LOG=
  - NEW
  - PRINT=

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

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 source;
```

Assign a libname.

```
libname lib1 'SAS-data-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 1996 3070 20411  spring 1997
3070 20412  spring 1997 3070 20413  spring 1997
3070 20414  spring 1996 3070 20416  spring 1995
```

```
3071 20500  spring 1994 3071 20501  spring 1995
3071 20502  spring 1996 3071 20503  spring 1996
3071 20505  spring 1994 3071 20506  spring 1994
3071 20507  spring 1994 3071 20424  spring 1994
;
```

Route the procedure output to a SAS catalog entry. 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.

```
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.

```
proc printto;
run;
```

Log

Output 35.4 SAS Log Routed to SAS Catalog Entry SASUSER.PROFILE.TEST.LOG.

You can view this catalog entry in the BUILD window of the SAS Explorer.

```
8
9    data lib1.inventry;
10      length Dept $ 4 Item $ 6 Season $ 6 Year 4;
11      input dept item season year @@;
12      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:
      real time           0.00 seconds
      cpu time            0.00 seconds


20   ;
21
22   proc printto print=lib1.cat1.inventry.output
23      label='Inventory program' new;
24   run;

NOTE: PROCEDURE PRINTTO used:
      real time           0.00 seconds
      cpu time            0.00 seconds


25
26   proc report data=lib1.inventry nowindows headskip;
27      column dept item season year;
28      title 'Current Inventory Listing';
29   run;

NOTE: PROCEDURE REPORT used:
      real time           0.00 seconds
      cpu time            0.00 seconds


30
31   proc printto;
32   run;
```

Output

Output 35.5 Procedure Output Routed to SAS Catalog Entry LIB1.CAT1.INVENTRY.OUTPUT.

You can view this catalog entry in the BUILD window of the SAS Explorer.

```
                        Current Inventory Listing                          1

                     Dept  Item    Season       Year

                     3070  20410   spring       1996
                     3070  20411   spring       1997
                     3070  20412   spring       1997
                     3070  20413   spring       1997
                     3070  20414   spring       1996
                     3070  20416   spring       1995
                     3071  20500   spring       1994
                     3071  20501   spring       1995
                     3071  20502   spring       1996
                     3071  20503   spring       1996
                     3071  20505   spring       1994
                     3071  20506   spring       1994
                     3071  20507   spring       1994
                     3071  20424   spring       1994
```

Example 3: Using Procedure Output as an Input File

Procedure features:

- PRINTTO statement:
  - Without options
  - Options:
    - LOG=
    - NEW
    - PRINT=

This example uses PROC PRINTTO to route procedure output to an external file and then uses that file as input to a DATA step.

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. See Output 35.6.

```
filename routed 'output-filename';
```

```
proc printto print=routed new;
run;
```

Produce the frequency counts. PROC FREQ computes frequency counts and a chi-square analysis of the variables X and Y in the data set TEST. This output is routed to the file that is referenced as ROUTED.

```
proc freq data=test;
   tables x*y / chisq;
run;
```

Close the file. You must use another PROC PRINTTO to close the file that is referenced by fileref ROUTED so that the following DATA step can read it. The step also routes subsequent procedure output to the default destination. PRINT= causes the step to affect only procedure output, not the SAS log.

```
proc printto print=print;
run;
```

Create the data set 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 Output 35.7.

```
proc print data=probtest;
   title 'Chi-Square Analysis for Table of X by Y';
run;
```

Output 35.6 PROC FREQ Output Routed to the External File Referenced as ROUTED

```
                          The FREQ Procedure

                           Table of x by y

          x         y

          Frequency|
          Percent  |
          Row Pct  |
          Col Pct  |       0|       1|       2|       3|       4|  Total
          ---------+--------+--------+--------+--------+--------+
                 0 |     29 |     33 |     12 |     25 |     27 |    126
                   |   2.90 |   3.30 |   1.20 |   2.50 |   2.70 |  12.60
                   |  23.02 |  26.19 |   9.52 |  19.84 |  21.43 |
                   |  15.18 |  16.18 |   6.25 |  11.74 |  13.50 |
          ---------+--------+--------+--------+--------+--------+
                 1 |     23 |     26 |     29 |     20 |     19 |    117
                   |   2.30 |   2.60 |   2.90 |   2.00 |   1.90 |  11.70
                   |  19.66 |  22.22 |  24.79 |  17.09 |  16.24 |
                   |  12.04 |  12.75 |  15.10 |   9.39 |   9.50 |
          ---------+--------+--------+--------+--------+--------+
                 2 |     28 |     26 |     32 |     30 |     25 |    141
                   |   2.80 |   2.60 |   3.20 |   3.00 |   2.50 |  14.10
                   |  19.86 |  18.44 |  22.70 |  21.28 |  17.73 |
                   |  14.66 |  12.75 |  16.67 |  14.08 |  12.50 |
          ---------+--------+--------+--------+--------+--------+
                 3 |     26 |     24 |     36 |     32 |     45 |    163
                   |   2.60 |   2.40 |   3.60 |   3.20 |   4.50 |  16.30
                   |  15.95 |  14.72 |  22.09 |  19.63 |  27.61 |
                   |  13.61 |  11.76 |  18.75 |  15.02 |  22.50 |
          ---------+--------+--------+--------+--------+--------+
                 4 |     25 |     31 |     28 |     36 |     29 |    149
                   |   2.50 |   3.10 |   2.80 |   3.60 |   2.90 |  14.90
                   |  16.78 |  20.81 |  18.79 |  24.16 |  19.46 |
                   |  13.09 |  15.20 |  14.58 |  16.90 |  14.50 |
          ---------+--------+--------+--------+--------+--------+
                 5 |     32 |     29 |     26 |     33 |     27 |    147
                   |   3.20 |   2.90 |   2.60 |   3.30 |   2.70 |  14.70
                   |  21.77 |  19.73 |  17.69 |  22.45 |  18.37 |
                   |  16.75 |  14.22 |  13.54 |  15.49 |  13.50 |
          ---------+--------+--------+--------+--------+--------+
                 6 |     28 |     35 |     29 |     37 |     28 |    157
                   |   2.80 |   3.50 |   2.90 |   3.70 |   2.80 |  15.70
                   |  17.83 |  22.29 |  18.47 |  23.57 |  17.83 |
                   |  14.66 |  17.16 |  15.10 |  17.37 |  14.00 |
          ---------+--------+--------+--------+--------+--------+
          Total         191      204      192      213      200     1000
                      19.10    20.40    19.20    21.30    20.00   100.00

                                                                            2

                          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
```

Output 35.7 PROC PRINT Output of Data Set PROBTEST, Routed to Default Destination

```
                Chi-Square Analysis for Table of X by Y                  3

                         Obs     chisq     prob

                          1     27.297    0.291
```

Example 4: Routing to a Printer

Procedure features:

PRINTTO statement:

Option:

PRINT= option

This example uses PROC PRINTTO to route procedure output directly to a printer.

Program

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

36

The PROTO Procedure

Information about the PROTO Procedure

See: For documentation of the PROTO procedure, go to **`http://support.sas.com/documentation/onlinedoc`**. Select Base SAS from the Product-Specific Documentation list.

CHAPTER

37

The 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.

Syntax: PRTDEF Procedure

PROC PRTDEF <*option(s)*>;

PROC PRTDEF Statement

PROC PRTDEF <*option(s)*>;

To do this	Use this option
Specify the input data set that contains the printer attributes	DATA=
Specify that the default operation is to delete the printer definitions from the registry	DELETE
Specify that the registry entries are being created for export to a different host	FOREIGN
Specify that a list of printers that are created or replaced will be written to the log	LIST
Specify that any printer name that already exists will be modified by using the information in the printer attributes data set	REPLACE
Specify whether the printer definitions are available to all users or just the users running PROC PRTDEF	USESASHELP

Options

DATA=*SAS-data-set*
specifies the SAS input data set that contains the printer attributes.

Requirements: Printer attributes variables that must be specified are DEST, DEVICE, MODEL, and NAME, except when the value of the variable OPCODE is DELETE, in which case only the NAME variable is required.

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 may still appear if it exists in the SASHELP catalog.

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 are created or replaced will be written to the log.

REPLACE
specifies that the default operation is to modify existing printer definitions. Any printer name that already exists will be modified by using the information in the printer attributes data set. Any printer name that does not exist will be added.

Interaction: If both REPLACE and DELETE are specified, then a DELETE will be performed.

USESASHELP
specifies that the printer definitions that 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 that are placed in the current SASUSER library, where they are available to the local user only.

Restriction: To use the USESASHELP option, you must have permission to write to the SASHELP catalog.

Operating Environment Information: 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 do one of the following:

- specify the printer definition as part of the PRINTERPATH system option
- from the Output Delivery System (ODS), issue the following code:

```
ODS PRINTER SAS PRINTER=myprinter;
```

  where *myprinter* is the name of your printer definition.

△

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.

Variable Name	Variable Description
Required	
DEST	Destination
DEVICE	Device
MODEL	Prototype
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

Variable Name	Variable Description
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.

Tip: While in interactive mode, you can invoke the registry with the REGEDIT command.

Tip: While in interactive mode, you can invoke the Print Setup dialog (DMPRTSETUP) and press New to view the list that is specified in the second window of the Printer Definition wizard.

Restriction: MODEL is limited to 127 characters.

NAME
: specifies the printer definition name that will be 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 will not be processed unless the REPLACE option has been specified or unless the value of the OPCODE variable is `Modify`.

Restriction: NAME must have the following features:

- It is limited to 127 characters.
- It must have at least one nonblank character.
- It cannot contain a backslash.

Note: Leading and trailing blanks will be 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.

Restriction: CHARSET is limited to 31 characters.

DESC
specifies the description of the printer.

Restriction: The description can have a maximum of 1023 characters.

Default: DESC defaults to the prototype that is used to create the printer.

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 will be reset to that 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.

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.

Restriction: OPTCODE is limited to 8 characters.

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.

Restriction: PAPERIN is limited to 31 characters.

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.

Restriction: PAPEROUT is limited to 31 characters.

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.

Restriction: PAPERSIZ is limited to 31 characters.

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.

Restriction: PAPERTYP is limited to 31 characters.

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.

Restriction: RES is limited to 31 characters.

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.

Restriction: STYLE is limited to 31 characters.

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 8 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.

Restriction: TYPEFACE is limited to 63 characters.

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.

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. The **%s** can be used as many times as needed.

Restriction: VIEWER is limited to 127 characters.

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

Procedure features:

PROC PRTDEF statement options:

DATA=
USESASHELP

This example shows you how to set up various printers.

Program

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, create the printer definitions, and make the definitions available to all users. The DATA= option specifies PRINTERS as the input data set that contains the printer attributes.

PROC PRTDEF creates the printer definitions for the SAS registry, and the USESASHELP option specifies that the printer definitions will be available to all users.

```
proc prtdef data=printers usesashelp;
run;
```

Example 2: Creating a Ghostview Printer in SASUSER to Preview PostScript Printer Output in SASUSER

Procedure features:

PROC PRTDEF statement options:

DATA=
LIST
REPLACE

This example creates a Ghostview printer definition in the SASUSER library for previewing PostScript output.

Program

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

Procedure features:

PROC PRTDEF statement option:

DATA=

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

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 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 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 will be 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. The DATA= option specifies TEK780 as the input data set.

```
proc prtdef data=tek780;
run;
```

Example 4: Adding, Modifying, and Deleting Printer Definitions

Procedure features:

PROC PRTDEF statement options:

DATA=
LIST

This example

- adds two printer definitions
- modifies a printer definition
- deletes two printer definitions.

Program

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, and 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 library=sasuser list;
run;
```

Example 5: Deleting a Single Printer Definition

Procedure features:

PROC PRTDEF statement option:

DELETE

This example shows you how to delete a printer from the registry.

Program

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;
```

See Also

Procedures
Chapter 38, “The PRTEXP Procedure,” on page 803

CHAPTER

38

The 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.

Syntax: PRTEXP Procedure

Note: 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)>;
  <**SELECT** printer_1 ...<printer_n>>;
  <**EXCLUDE** printer_1 ... <printer_n>>;

PROC PRTEXP Statement

PROC PRTEXP<*option(s)*>;

Options

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

The EXCLUDE statement will cause the output to contain information from all those printers that are not listed.

EXCLUDE printer_1 … <printer_n>;

Required Arguments

printer_1 printer_n
specifies the printer(s) that you do not want the output to contain information about.

SELECT Statement

The SELECT statement will cause the output to contain information from only those printers that are listed.

SELECT printer_1 … <printer_n>;

Required Arguments

printer_1 printer_n
specifies the printer(s) that you would like the output to contain information about.

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 791.

Examples: PRTEXP Procedure

Example 1: Writing Attributes to the SAS Log

Procedure Features:

PROC PRTEXP statement option:

SELECT statement
USESASHELP option

This example shows you how to write the attributes that are used to define a printer to the SAS log.

Program

Specify the printer that you want information about, specify that only the SASHELP portion of the registry be searched, and write the information to the SAS log. The SELECT statement specifies that you want the attribute information that is used to define the printer Postscript to be included in the output. The USESASHELP option specifies that only the SASHELP registry is to be searched for Postscript's printer definitions. The data that is needed to define each printer is written to the SAS log because the OUT= option was not used to specify a SAS data set.

```
proc prtexp usesashelp;
select postscript;
run;
```

Example 2: Writing Attributes to a SAS Data Set

Procedure Features:

PROC PRTEXP statement option:

OUT= option
SELECT statement

This example shows you how to create a SAS data set that contains the data that PROC PRTDEF would use to define the printers PCL4, PCL5, PCL5E, and PCLC.

Program

Specify the printers that you want information about and create the PRDVTER data set. The SELECT statement specifies the printers PCL4, PCL5, PCL5E, and PCLC. The OUT= option creates the SAS data set PRDVTER, which contains the same attributes that are used by PROC PRTDEF to define the printers PCL4, PCL5, PCL5E, and PCLC. SAS will search both the SASUSER and SASHELP registries, because USESASHELP was not specified.

```
proc prtexp out=PRDVTER;
select pcl4 pcl5 pcl5e pcl5c;
run;
```

See Also

Procedures

Chapter 37, "The PRTDEF Procedure," on page 789

CHAPTER

39

The PWENCODE Procedure

Overview: PWENCODE Procedure

The PWENCODE procedure enables you to encode passwords. Encoded passwords can be used in place of plain-text passwords in SAS programs that access relational database management systems (RDBMSs), SAS/SHARE servers, and SAS Integrated Object Model (IOM) servers (such as the SAS Metadata Server).

Syntax: PWENCODE Procedure

PROC PWENCODE IN='*password*' <OUT=*fileref*> <METHOD=*encoding-method*>;

PROC PWENCODE Statement

PROC PWENCODE IN='*password*' <OUT=*fileref*> <METHOD=*encoding-method*>;

Required Argument

IN='*password*'

specifies the password to encode. *password* can have no more than 512 characters. *password* can contain letters, numerals, spaces, and special characters. If *password* contains embedded single or double quotation marks, then use the standard SAS rules for quoting character constants (see "SAS Constants in Expressions" in *SAS Language Reference: Concepts* for details).

Featured in: Example 1 on page 809, Example 2 on page 809, and Example 3 on page 811

Options

OUT=*fileref*

specifies a fileref to which the output string is to be written. If the OUT= option is not specified, then the output string is written to the SAS log.

Featured in: Example 2 on page 809 and Example 3 on page 811

METHOD=*encoding-method*

specifies the encoding method to use. Currently, **sas001** is the only supported encoding method and is the default if the METHOD= option is omitted.

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 plain text.

Note: SAS does not currently support encoded read, write, or alter passwords for SAS data sets. △

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.

Examples: PWENCODE Procedure

Example 1: Encoding a Password

Procedure features: IN= argument

This example shows a simple case of encoding a password and writing the encoded password to the SAS log.

Program

Encode the password.

```
proc pwencode in='my password';
run;
```

Log

Output 39.1

```
6    proc pwencode in='my password';
7    run;

{sas001}bXkgcGFzc3dvcmQ=

NOTE: PROCEDURE PWENCODE used (Total process time):
      real time           0.31 seconds
      cpu time            0.08 seconds
```

Example 2: Using an Encoded Password in a SAS Program

Procedure features:
IN= argument
OUT= option

This example

- □ encodes a password and saves it to an external file
- □ reads the encoded password with a DATA step, stores it in a macro variable, and uses it in a SAS/ACCESS LIBNAME statement.

Program 1: Encoding the Password

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

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. For more information about the SYMBOLGEN SAS system option, see **SAS Macro Language: Reference**.

```
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. For details about the INFILE and INPUT statements, the $VARYING. informat, and the CALL SYMPUT routine, see **SAS Language Reference: Dictionary**.

```
data _null_;
   infile pwfile obs=1 length=l;
   input @;
   input @1 line $varying1024. l;
   call symput('dbpass',substr(line,1,l));
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

```
28   data _null_;
29      infile pwfile obs=1 length=l;
30      input @;
31      input @1 line $varying1024. l;
32      call symput('dbpass',substr(line,1,l));
33   run;

NOTE: The infile PWFILE is:
      File Name=external-filename,
      RECFM=V,LRECL=256

NOTE: 1 record was read from the infile PWFILE.
      The minimum record length was 20.
      The maximum record length was 20.
NOTE: DATA statement used (Total process time):
      real time           3.94 seconds
      cpu time            0.03 seconds


34   libname x odbc
SYMBOLGEN:  Macro variable DBPASS resolves to {sas001}bXlwYXNzMQ==
34 !                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

Procedure features:

IN= argument

OUT= option

Other features:

FILENAME statement with CLIPBRD access method

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

Declare a fileref with the CLIPBRD access method. For more information about the FILENAME statement with the CLIPBRD accedd method, see **SAS Language Reference: Dictionary**.

```
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;
```

CHAPTER 40

The 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

Output 40.1 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 40.1 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 Output 40.2, 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

- 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 on page 823.

Output 40.2 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
```

Syntax: RANK Procedure

Reminder: You can use the ATTRIB, FORMAT, LABEL, and WHERE statements. See Chapter 3, "Statements with the Same Function in Multiple Procedures," on page 57 for details. You can also use any global statements. See "Global Statements" on page 18 for a list.

PROC RANK <*option(s)*>;

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

VAR *data-set-variables(s)*;

RANKS *new-variables(s)*;

To do this	Use this statement
Calculate a separate set of ranks for each BY group	BY
Identify a variables that contain the ranks	RANKS
Specify the variables to rank	VAR

PROC RANK Statement

PROC RANK <*option(s)*>;

To do this	Use this option
Specify the input data set	DATA=
Create an output data set	OUT=
Specify the ranking method	
Compute fractional ranks	FRACTION or NPLUS1
Partition observations into groups	GROUPS=
Compute normal scores	NORMAL=
Compute percentages	PERCENT
Compute Savage scores	SAVAGE
Reverse the order of the rankings	DESCENDING
Specify how to rank tied values	TIES=

Note: You can specify only one ranking method in a single PROC RANK step. △

Options

DATA=*SAS-data-set*
specifies the input SAS data set.

Main discussion: “Input Data Sets” on page 19

Restriction: You cannot use PROC RANK with an engine that supports concurrent access if another user is updating the data set at the same time.

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.

Featured in: Example 1 on page 822 and Example 2 on page 823

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 also: 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

$$\mathrm{FLOOR}\left(rank * k / (n+1)\right)$$

where 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.

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.

Featured in: Example 3 on page 826

NORMAL=BLOM | TUKEY | VW

computes normal scores from the ranks. The resulting variables appear normally distributed. The formulas are

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	$y_i=\Phi^{-1}(r_i)/(n+1)$

where Φ^{-1} is the inverse cumulative normal (PROBIT) function, r_i is the rank of the ith 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).

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 normal score.

NPLUS1

computes fractional ranks by dividing each rank by the denominator n+1, where n is the number of observations having nonmissing values of the ranking variable.

Aliases: FN1, N1

Interaction: TIES=HIGH is the default with the NPLUS1 option.

See also: 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.

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.

Alias: P

Interaction: TIES=HIGH is the default with the PERCENT option.

Tip: You can use PERCENT to calculate cumulative percentages, but use GROUPS=100 to compute percentiles.

SAVAGE
computes Savage (or exponential) scores from the ranks by the following formula (Lehman 1998):

$$y_i = \left[\sum_{j=n-r_i+1} \left(\frac{1}{j} \right) \right] - 1$$

TIES=HIGH | LOW | MEAN
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).

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.

Featured in: Example 1 on page 822 and Example 2 on page 823

BY Statement

Produces a separate set of ranks for each BY group.

Main discussion: "BY" on page 58

Featured in: Example 2 on page 823 and Example 3 on page 826

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, 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*.

Options

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.

RANKS Statement

Creates new variables for the rank values.

Requirement: If you use the RANKS statement, you must also use the VAR statement.

Default: If you omit the RANKS statement, the rank values replace the original variable values in the output data set.

Featured in: Example 1 on page 822 and Example 2 on page 823

RANKS *new-variables(s)*;

Required Arguments

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 forth.

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.

Featured in: Example 1 on page 822, Example 2 on page 823, and Example 3 on page 826

VAR *data-set-variables(s)*;

Required Arguments

data-set-variable(s)
specifies one or more variables for which ranks are computed.

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 correspond 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.

Concepts: RANK Procedure

Computer Resources

PROC RANK stores all values in memory of the variables for which it computes ranks.

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, which you can compare 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).

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.

Examples: RANK Procedure

Example 1: Ranking Values of Multiple Variables

Procedure features:

PROC RANK statement options:

- DESCENDING
- TIES=

RANKS statement

VAR statement

Other features:

PRINT procedure

This example

- reverses the order of the ranks so that the highest value receives the rank of 1
- assigns tied values the best possible rank
- creates ranking variables and prints them with the original variables.

Program

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 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

```
                    Rankings of Participants' Scores                      1

                                                  Present    Taste
          Obs    Name          Present    Taste     Rank      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

Procedure features:

PROC RANK statement options:

- DESCENDING
- TIES=

BY statement

RANKS statement

VAR statement

Other features:
PRINT procedure

This example
- ranks observations separately within BY groups
- reverses the order of the ranks so that the highest value receives the rank of 1
- assigns tied values the best possible rank
- creates ranking variables and prints them with the original variables.

Program

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 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

In the second district, Hinkley and Thrash tied with 912 votes. 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

Procedure features:
PROC RANK statement option:
GROUPS=
BY statement
VAR statement

Other features:
PRINT procedure
SORT procedure

This example

- □ 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

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 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

The group values pair up 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: John Wiley & Sons, Inc.

Conover, W.J. (1998), *Practical Nonparametric Statistics, Third Edition*, New York: John Wiley & Sons, Inc.

Conover, W.J. and Iman, R.L. (1976), "On Some Alternative Procedures Using Ranks for the Analysis of Experimental Designs," *Communications in Statistics*, A5, 14, 1348–1368.

Conover, W.J. and Iman, R.L. (1981), "Rank Transformations as a Bridge between Parametric and Nonparametric Statistics," *The American Statistician*, 35, 124–129.

Iman, R.L. and Conover, W.J. (1979), "The Use of the Rank Transform in Regression," *Technometrics*, 21, 499–509.

Lehman, E.L. (1998), *Nonparametrics: Statistical Methods Based on Ranks*, New Jersey: Prentice Hall .

Quade, D. (1966), "On Analysis of Variance for the k-Sample Problem," *Annals of Mathematical Statistics*, 37, 1747–1758.

Tukey, John W. (1962), "The Future of Data Analysis," *Annals of Mathematical Statistics*, 33, 22.

CHAPTER
41

The 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

- 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.

Syntax: REGISTRY Procedure

PROC REGISTRY <*option(s)*>;

PROC REGISTRY Statement

PROC REGISTRY <*option(s)*>;

To do this	Use this option
Erase the contents of the SASUSER registry	CLEARSASUSER
Compare two registry files	COMPAREREG1 and COMPAREREG2
Compare the contents of a registry to a file	COMPARETO
Enable registry debugging	DEBUGON
Disable registry debugging	DEBUGOFF
Write the contents of a registry to the specified file	EXPORT=
Provide additional information in the SAS log about the results of the IMPORT= and the UNINSTALL options	FULLSTATUS
Import the specified file to a registry	IMPORT=
Write the contents of the registry to the SAS log. Used with the STARTAT= option to list specific keys.	LIST
Write the contents of the SASHELP portion of the registry to the SAS log	LISTHELP
Send the contents of a registry to the log	LISTREG
Write the contents of the SASUSER portion of the registry to the SAS log	LISTUSER
Start exporting or writing or comparing the contents of a registry at the specified key	STARTAT=
Delete from the specified registry all the keys and values that are in the specified file	UNINSTALL
Uppercase all incoming key names	UPCASE
Perform the specified operation on the SASHELP portion of the SAS registry	USESASHELP

Options

CLEARSASUSER
erases the content of the SASUSER portion of the SAS registry.

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.

Featured in: Example 4 on page 842

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.

Featured in: Example 4 on page 842

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 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.

Featured in: Example 3 on page 841

See also: For information about how to structure a file that contains registry information, see "Creating Registry Files with the REGISTRY Procedure" on page 836.

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.

Interaction: 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.

Interaction: To export a single key and all of its subkeys, specify the STARTAT= option.

Featured in: Example 2 on page 840

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.

Interaction: 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.

Interaction: To obtain additional information in the SAS log as you import a file, use FULLSTATUS.

Featured in: Example 1 on page 839

See also: For information about how to structure a file that contains registry information, see "Creating Registry Files with the REGISTRY Procedure" on page 836.

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.

Featured in: Example 2 on page 840

STARTAT=*'key-name'*
exports or writes the contents of a single key and all of its subkeys.

Interaction: USE STARTAT= with the EXPORT=, LIST, LISTHELP, LISTUSER, COMPAREREG1=, COMPAREREG2= and the LISTREG option.

Featured in: Example 4 on page 842

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

- use the Explorer Window
- use the FILENAME statement. (For information about the FILENAME statement, see the section on statements in *SAS Language Reference: Dictionary*.)

Interaction: 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.

Interaction: Use FULLSTATUS to obtain additional information in the SAS log as you uninstall a registry.

See also: For information about how to structure a file that contains registry information, see "Creating Registry Files with the REGISTRY Procedure" on page 836.

UPCASE
uppercases all incoming key names.

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. See “Sample Registry Entries” on page 837.

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. Hex values may 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 41.1 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

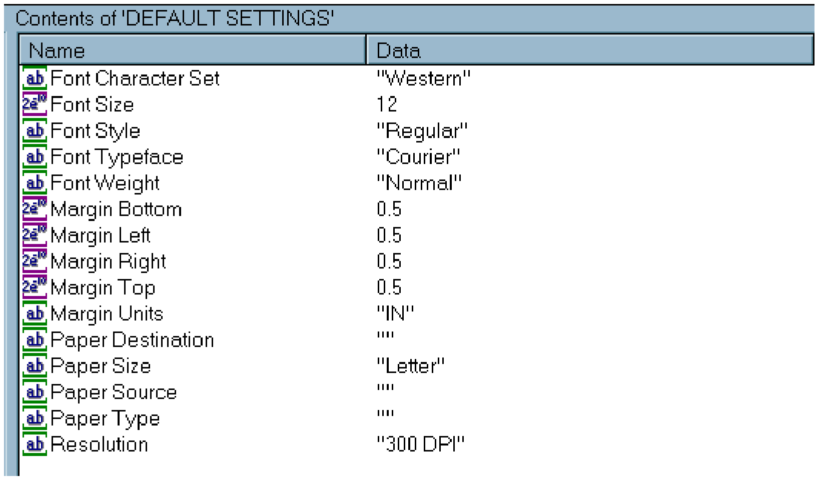
The following list contains a sample of valid registry values:

- □ 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 41.2 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:

Example Code 41.1 SAS code for sending a SASUSER registry entry to the log

```
proc registry
  listuser
  startat='sasuser-registry-key-name';
run;
```

Example Code 41.2 SAS code for sending a SASUSER registry entry to the log

```
proc registry
  listuser
  startat='HKEY_SYSTEM_ROOT\CORE\PRINTING\PRINTERS\PostScript\DEFAULT SETTINGS';
run;
```

For example, the list below begins at the CORE\PRINTING\PRINTERS\PostScript\DEFAULT SETTINGS key.

Output 41.1 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"
NOTE: PROCEDURE REGISTRY used (Total process time):
      real time           0.03 seconds
      cpu time            0.03 seconds
```

Examples: REGISTRY Procedure

Example 1: Importing a File to the Registry

Procedure features: IMPORT=

Other features: FILENAME statement

This example imports a file into the SASUSER portion of the SAS registry.

Source File

The following 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

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;
```

SAS Log

```
1   filename source 'external-file';
2   proc registry
3      import=source;
4   run;
Parsing REG file and loading the registry please wait....
Registry IMPORT is now complete.
```

Example 2: Listing and Exporting the Registry

Procedure features:

EXPORT=
LISTUSER

This example lists the SASUSER portion of the SAS registry and exports it to an external file.

Note: This is usually a very large file. To export a portion of the registry, use the STARTAT= option. △

Program

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;
```

SAS Log

```
1  proc registry listuser export='external-file';
2  run;
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

Procedure features: COMPARETO= option
Other features: FILENAME statement

This example compares the SASUSER portion of the SAS registry to an external file. Comparisons such as this 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.

Note: To compare the SASHELP portion of the registry with an external file, specify the USESASHELP option. △

Program

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;
```

SAS Log

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”.

```
1   filename testreg 'external-file';
2   proc registry
3     compareto=testreg;
4  run;
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

Procedure features

COMPAREREG1= and COMPAREREG2= options

STARTAT= option

This example uses the REGISTRY procedure options COMPAREREG1= and COMPAREREG2= to specify two registry files for comparison.

Program

Declare the PROCLIB library. The PROCLIB library contains a registry file.

```
libname proclib 'SAS-data-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;
```

SAS Log

```
8    proc registry comparereg1='sasuser.regstry'
9
10      startat='CORE\EXPLORER'
11      comparereg2='proclib.regstry';
12   run;
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;

NOTE: PROCEDURE REGISTRY used (Total process time):
      real time           0.07 seconds
      cpu time            0.02 seconds
```

See Also

SAS registry chapter in *SAS Language Reference: Concepts*

CHAPTER 42

The 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 windowing environment with a prompting facility that guides you as you build a report.
- in a windowing environment without the prompting facility.
- 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 running SAS in *SAS Language Reference: Concepts*).

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 *detail row*. A *summary report* consolidates data so that each row represents multiple observations. Each of these rows is also called a detail row.

Both detail and summary reports can contain *summary lines* as well as detail 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 861).

This overview illustrates the kinds of reports that PROC REPORT can produce. The statements that create the data sets and formats used in these reports are in Example 1 on page 948. The formats are stored in a permanent SAS data library. See "Examples: REPORT Procedure" on page 948 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 42.1 on page 848 illustrates the simplest kind 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 on page 948.

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-data-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 42.1 Simple Detail Report with a Detail Row for Each Observation

Detail row

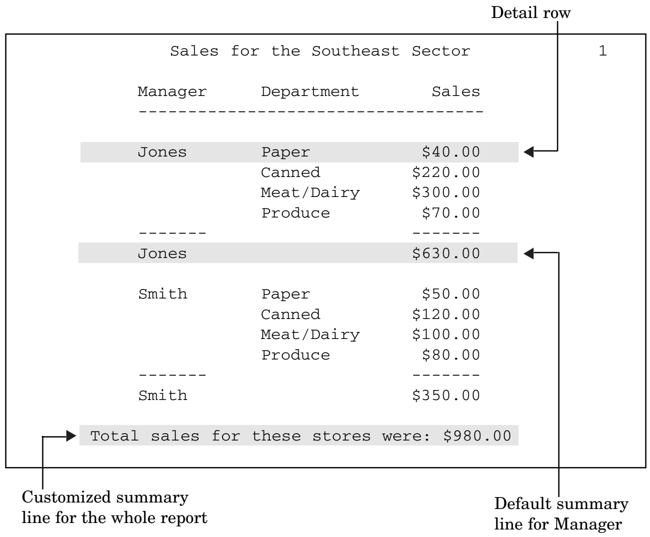
```
                        The SAS System                       1

      Sector       Manager  Department        Sales
      Southeast    Smith    Paper            $50.00
      Southeast    Smith    Meat/Dairy      $100.00
      Southeast    Smith    Canned          $120.00
      Southeast    Smith    Produce          $80.00
      Southeast    Jones    Paper            $40.00
      Southeast    Jones    Meat/Dairy      $300.00
      Southeast    Jones    Canned          $220.00
      Southeast    Jones    Produce          $70.00
```

The report in Figure 42.2 on page 849 uses the same observations as those in Figure 42.1 on page 848. 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 on page 951.

Figure 42.2 Ordered Detail Report with Default and Customized Summaries

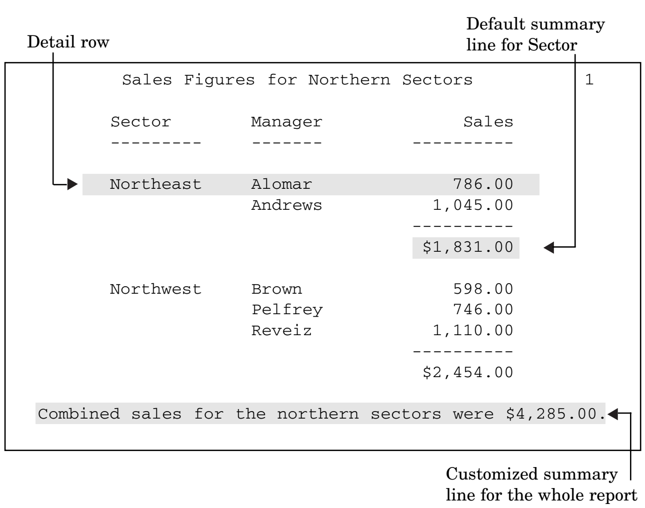


The summary report in Figure 42.3 on page 849 contains one row for each store in the northern sector. Each detail row represents four observations in the input data set, one observation for each department. Information about individual departments does not appear in this report. Instead, the value of 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 headers
- □ 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 on page 957.

Figure 42.3 Summary Report with Default and Customized Summaries



The summary report in Figure 42.4 on page 850 is similar to Figure 42.3 on page 849. 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 headers.

For an explanation of the program that produces this report, see Example 5 on page 960.

Figure 42.4 Summary Report with a Column for Each Value of a Variable

Computed variable

```
     Sales Figures for Perishables in Northern Sectors    1

                      ______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


  --------------------------------------------------
  |  Combined sales for meat and dairy : $1,545.00 |
  |  Combined sales for produce :          $390.00 |
  |                                                |
  |  Combined sales for all perishables: $1,935.00 |
  --------------------------------------------------
```

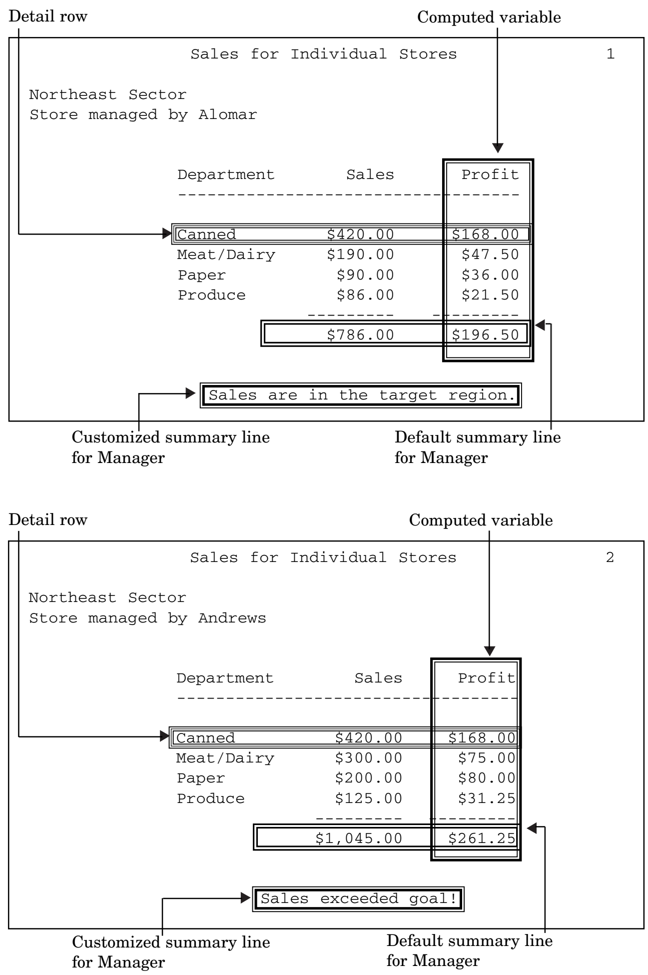
Customized summary lines
for the whole report

The customized report in Figure 42.5 on page 851 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 header 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 on page 971.

Figure 42.5 Customized Summary Report



The report in Figure 42.6 on page 852 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 on page 994. For information on ODS, see "Output Delivery System" on page 32.

Figure 42.6 HTML Output

Sales for the Southeast Sector

Manager	Department	Sales
Jones	Paper	40
	Canned	220
	Meat/Dairy	300
	Produce	70
Jones		630
Subtotal for Jones is $630.00.		
Smith	Paper	50
	Canned	120
	Meat/Dairy	100
	Produce	80
Smith		350
Subtotal for Smith is $350.00.		
Total for all departments is: $980.00.		

Concepts: REPORT Procedure

Laying Out a Report

Planning the Layout

Report writing is simplified if you approach it with a clear understanding of what you want the report to look like. The most important thing to determine is the layout of the report. To design the layout, ask yourself the following kinds 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 headers 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 windowing 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 windowing 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 header, 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.

Featured in: Example 1 on page 948

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.

Featured in: Example 2 on page 951

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 (see "COLUMN Statement" on page 893), then PROC REPORT displays the N statistic (the number of observations in the input data set that belong to that cell of the report).

If you are familiar with procedures that use class variables, then you will see that across variables are class variables that are used in the column dimension.

Featured in: Example 5 on page 960

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.

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 "COLUMN Statement" on page 893). In the windowing 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. △

Featured in: Example 4 on page 957

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.

See also: "BREAK Statement" on page 885 and "RBREAK Statement" on page 908

Featured in: Example 2 on page 951, Example 3 on page 954, Example 4 on page 957, and Example 5 on page 960

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 kind of variable, PROC REPORT summarizes them. △

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 windowing 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.

Featured in: Example 5 on page 960, Example 10 on page 975, and Example 13 on page 983

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 detail rows of the report according to the values of order and group variables, considered from left to right in the report. 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 instance, in Figure 42.7 on page 856, 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 kind of report with the COLUMN statement or, in the windowing environment, by placing report items above or below each other. This is called stacking items in the report because each item generates a header, and the headers are stacked one above the other.

Figure 42.7 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.

Table 42.1 on page 857 shows which report items can share a column.

Note: You cannot stack order variables with other report items. △

Table 42.1 Report Items That Can Share Columns

	Display	Analysis	Order	Group	Computed	Across	Statistic
Display						X*	X
Analysis						X	X
Order							
Group						X	
Computed variable						X	
Across	X*	X			X	X	X
Statistic	X	X				X	

*When a display variable and an across variable share a column, the report must also contain another variable that is not in the same column.

When a column is defined by stacked report items, PROC REPORT formats the values in the column by using the format that is specified for the lowest report item in the stack that does not have an ACROSS usage.

The following items can stand alone in a column:

- display variable
- analysis variable
- order variable
- group variable
- computed variable
- across variable
- N statistic.

Note: The values in a column that is occupied only by an across variable are frequency counts. △

Statistics That Are Available in PROC REPORT

Descriptive statistic keywords	
CSS	PCTSUM
CV	RANGE
MAX	STDDEV | 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	
PROBT	T

These statistics, the formulas that are used to calculate them, and their data requirements are discussed in "Keywords and Formulas" on page 1340.

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 861).

Note: When you use the COMPUTE statement, you do not have to use a corresponding BREAK or RBREAK statement. (See Example 2 on page 951, 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 on page 971, 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 890).

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 859). A PROC REPORT step can contain multiple compute blocks, but they cannot be nested.

The Contents of Compute Blocks

In the windowing 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	IF-THEN/ELSE
assignment	LENGTH
CALL	RETURN
DO (all forms)	SELECT
END	sum

- comments
- null statements
- macro variables and macro invocations
- all DATA step functions.

For information about SAS language elements see the appropriate section in *SAS Language Reference: Dictionary*.

Within a compute block, you can also use these PROC REPORT features:

- Compute blocks for a customized summary can contain one or more LINE statements, which place customized text and formatted values in the summary. (See "LINE Statement" on page 907.)
- 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 "CALL DEFINE Statement" on page 890.)
- Any compute block can contain the automatic variable _BREAK_ (see "The Automatic Variable _BREAK_" on page 862.

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 or not 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: 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 902 and NOZERO on page 903. △

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.

If the variable that you reference is this type...	Then refer to it by...	For 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.

Featured in: Example 3 on page 954, which references analysis variables by their aliases; Example 5 on page 960, which references variables by column number; and Example 10 on page 975, 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 detail 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 on compute block processing see "How PROC REPORT Builds a Report" on page 936. △

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 (in traditional SAS monospace output only)

4 blank line
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 Output 42.1. The statements that create the output follow. The user-defined formats that are used are created by a PROC FORMAT step on page 949.

```
libname proclib 'SAS-data-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;
```

Output 42.1 Three Different Meanings of Sales.sum

```
                    The SAS System                      1

            Sector     Manager      Sales
            Northeast  Alomar     $786.00     ❶
                       Andrews  $1,045.00
            ---------           ---------
            Northeast           $1,831.00     ❷

            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     ❸
            =========           =========
```

Here Sales.sum has three different meanings:

❶ 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.

❷ 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.

❸ In the report summary line, the value $6,313.00 is the sales for all stores in the city.

Note: Unless you use the NOALIAS option in the PROC REPORT statement, when you refer in a compute block to a statistic that has an alias, you 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 859). △

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 forth. For information about the attributes that you can set for a style, see *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 42.2 Location Values

Location Value	Part of Report Affected
CALLDEF	Cells identified by a CALL DEFINE statement
COLUMN	Column cells
HEADER | HDR	Column headers
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. Table 42.3 on page 864 shows valid and default values for *location* 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. Users can create their own style definitions with the TEMPLATE procedure (see *SAS Output Delivery System: User's Guide* for information about PROC TEMPLATE). The following table shows the default style elements for each statement.

Table 42.3 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
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 the REPORT location. Note that not all style attributes are valid in all destinations. See *SAS Output Delivery System: User's Guide* for more information on these style attributes, their valid values, and their applicable destinations.

BACKGROUND=	FONT_WIDTH=*
BACKGROUNDIMAGE=	FOREGROUND=*
BORDERCOLOR=	FRAME=
BORDERCOLORDARK=	HTMLCLASS=
BORDERCOLORLIGHT=	JUST=
BORDERWIDTH=	OUTPUTWIDTH=
CELLPADDING=	POSTHTML=
CELLSPACING=	POSTIMAGE=
FONT=*	POSTTEXT=
FONT_FACE=*	PREHTML=
FONT_SIZE=*	PREIMAGE=
FONT_STYLE=*	PRETEXT=
FONT_WEIGHT=*	RULES=

* When you use these attributes in this location, they affect only the text that is specified with the PRETEXT=, POSTTEXT=, PREHTML=, and POSTHTML= attributes. To alter the foreground color or the font for the text that appears in the table, you must set the corresponding attribute in a location that affects the cells rather than the table.

The following table shows valid values of *style-attribute-name* for the CALLDEF, COLUMN, HEADER, LINES, and SUMMARY locations. Note that not all style attributes are valid in all destinations. See *SAS Output Delivery System: User's Guide* for more information on these style attributes, their valid values, and their applicable destinations.

ASIS=	FONT_WIDTH=
BACKGROUND=	HREFTARGET=
BACKGROUNDIMAGE=	HTMLCLASS=
BORDERCOLOR=	JUST=
BORDERCOLORDARK=	NOBREAKSPACE=
BORDERCOLORLIGHT=	POSTHTML=
BORDERWIDTH=	POSTIMAGE=
CELLHEIGHT=	POSTTEXT=
CELLWIDTH=	PREHTML=
FLYOVER=	PREIMAGE=
FONT=	PRETEXT=

FONT_FACE=	PROTECTSPECIALCHARS=
FONT_SIZE=	TAGATTR=
FONT_STYLE=	URL=
FONT_WEIGHT=	VJUST=

Specifications in a statement other than the PROC REPORT statement 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 statement are inherited. For instance, 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 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=[background=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)=[font_size=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 such as Printer or RTF. For details on these destinations and on using the ODS statement, 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 867). 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 a windowing environment. △

Operating Environment Information: Printing is implemented differently in different operating environments. For information related to printing, consult *SAS Language Reference: Concepts*. Additional information may 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 35, "The PRINTTO Procedure," on page 771).

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 windowing 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 windowing environment, you can save the definition of the current report by selecting

File ► Save Report

A report definition may differ from the SAS program that creates the report (see the discussion of OUTREPT= on page 879).

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 windowing environment, load a report definition from the LOAD REPORT window by selecting

File ► Open Report

Syntax: REPORT Procedure

Tip: Supports the Output Delivery System. See "Output Delivery System" on page 32 for details.

ODS Table Name: Report

Reminder: You can use the ATTRIB, FORMAT, LABEL, and WHERE statements. See Chapter 3, "Statements with the Same Function in Multiple Procedures," on page 57 for details. You can also use any global statements. See "Global Statements" on page 18 for a list.

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*;

To do this	Use this statement
Produce a default summary at a change in the value of a group or order variable	BREAK
Create a separate report for each BY group	BY
Set the value of an attribute for a particular column in the current row	CALL DEFINE
Describe the arrangement of all columns and of headers that span more than one column	COLUMN

To do this	Use this statement
Specify one or more programming statements that PROC REPORT executes as it builds the report	COMPUTE and ENDCOMP
Describe how to use and display a report item	DEFINE
Treat observations as if they appear multiple times in the input data set	FREQ
Provide a subset of features of the PUT statement for writing customized summaries	LINE
Produce a default summary at the beginning or end of a report or at the beginning and end of each BY group	RBREAK
Specify weights for analysis variables in the statistical calculations	WEIGHT

PROC REPORT Statement

PROC REPORT <*option(s)*>;

<table>
<tr><th>To do this</th><th>Use this option</th></tr>
<tr><td>Specify the input data set</td><td>DATA=</td></tr>
<tr><td>Specify the output data set</td><td>OUT=</td></tr>
<tr><td>Override the SAS system option THREADS | NOTHREADS</td><td>THREADS | NOTHREADS</td></tr>
<tr><td>Select the windowing or the nonwindowing environment</td><td>WINDOWS|NOWINDOWS</td></tr>
<tr><td>Use a report that was created before compute blocks required aliases (before Release 6.11)</td><td>NOALIAS</td></tr>
<tr><td>Control the statistical analysis</td><td></td></tr>
<tr><td>Specify the divisor to use in the calculation of variances</td><td>VARDEF=</td></tr>
<tr><td>Specify the sample size to use for the P^2 quantile estimation method</td><td>QMARKERS=</td></tr>
<tr><td>Specify the quantile estimation method</td><td>QMETHOD=</td></tr>
<tr><td>Specify the mathematical definition to calculate quantiles</td><td>QNTLDEF=</td></tr>
<tr><td>Exclude observations with nonpositive weight values from the analysis.</td><td>EXCLNPWGT</td></tr>
</table>

To do this	Use this option
Control classification levels	
Create all possible combinations of the across variable values	COMPLETECOLS | NOCOMPLETECOLS
Create all possible combinations of the group variable values	COMPLETEROWS | NOCOMPLETEROWS
Control the layout of the report	
Use formatting characters to add line-drawing characters to the report	BOX*
Specify whether to center or left-justify the report and summary text	CENTER | NOCENTER
Specify the default number of characters for columns containing computed variables or numeric data set variables	COLWIDTH=*
Define the characters to use as line-drawing characters in the report	FORMCHAR=*
Specify the length of a line of the report	LS=*
Consider missing values as valid values for group, order, or across variables	MISSING
Specify the number of panels on each page of the report	PANELS=*
Specify the number of lines in a page of the report	PS=
Specify the number of blank characters between panels	PSPACE=*
Override options in the DEFINE statement that suppress the display of a column	SHOWALL
Specify the number of blank characters between columns	SPACING=*
Display one value from each column of the report, on consecutive lines if necessary, before displaying another value from the first column	WRAP
Customize column headers	
Underline all column headers and the spaces between them	HEADLINE*
Write a blank line beneath all column headers	HEADSKIP*
Suppress column headers	NOHEADER
Write *name*= in front of each value in the report, where *name*= is the column header for the value	NAMED
Specify the split character	SPLIT=

To do this	Use this option
Control ODS output	
Specify one or more style elements (for the Output Delivery System) to use for different parts of the report	STYLE=
Specify text for the HTML or PDF table of contents entry for the output	CONTENTS=
Store and retrieve report definitions, PROC REPORT statements, and your report profile	
Write to the SAS log the PROC REPORT code that creates the current report	LIST
Suppress the building of the report	NOEXEC
Store in the specified catalog the report definition that is defined by the PROC REPORT step that you submit	OUTREPT=
Identify the report profile to use	PROFILE=
Specify the report definition to use	REPORT=
Control the windowing environment	
Display command lines rather than menu bars in all REPORT windows	COMMAND
Identify the library and catalog containing user-defined help for the report	HELP=
Open the REPORT window and start the PROMPT facility	PROMPT

* Traditional SAS monospace output only.

Options

BOX

uses formatting characters to add line-drawing characters to the report. These characters

- surround each page of the report
- separate column headers 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 also: the discussion of FORMCHAR= on page 874

Featured in: Example 12 on page 980

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.

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

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.

Default: 9

Range: 1 to the linesize

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Interaction: 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 901.)

If no format is associated with the item, then the column width depends on variable type:

If the variable is a...	Then the column width is the...
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

Featured in: Example 2 on page 951

COMMAND

displays command lines rather than menu bars in all REPORT windows.

After you have started PROC REPORT in the windowing 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 on HTML and PDF output, see "Output Delivery System" on page 32.

Note: A hexadecimal value (such as **`'DF'x`**) that is specified within *link-text* 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 *link-text* in double quotation marks (" ") so that the macro function will resolve. △

Restriction: For HTML output, the CONTENTS= option has no effect on the HTML body file. It affects only the HTML contents file.

DATA=*SAS-data-set*
specifies the input data set.

Main discussion: "Input Data Sets" on page 19

EXCLNPWGT
excludes observations with nonpositive weight values (zero or negative) from the analysis. By default, PROC REPORT treats observations with negative weights like those with zero weights and counts them in the total number of observations.

Alias: EXCLNPWGTS

Requirement: You must use a WEIGHT statement.

See also: "WEIGHT Statement" on page 912

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.

Range: PROC REPORT uses 12 of the 20 formatting characters that SAS provides. Table 42.4 on page 875 shows the formatting characters that PROC REPORT uses. Figure 42.8 on page 876 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 that they are listed. For instance, 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)='*#'
```

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 an **x** after the closing quotation mark. For instance, 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
```

Table 42.4 Formatting Characters Used by PROC REPORT

Position	Default	Used to draw
1	|	the right and left borders and the vertical separators between columns
2	-	the 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	-	the top character in the left border
4	-	the top character in a line of characters that separates columns
5	-	the top character in the right border
6	|	the leftmost character in a row of horizontal separators
7	+	the intersection of a column of vertical characters and a row of horizontal characters
8	|	the rightmost character in a row of horizontal separators

9	-	the bottom character in the left border
10	-	the bottom character in a line of characters that separate columns
11	-	the bottom character in the right border
13	=	double overlining and double underlining in break lines

Figure 42.8 Formatting Characters in PROC REPORT Output

```
          Sales for Northern Sectors         1

        Sector     Manager       Sales
        ------------------------------      2

        Northeast  Alomar       786.00
                   Andrews    1,045.00
                             ----------
                              1,831.00     2
                             ----------

        Northwest  Brown        598.00
                   Pelfrey      746.00
                   Reveiz     1,110.00
                             ----------
                              2,454.00
                             ----------

                             ==========
                              4,285.00     13
                             ==========
```

HEADLINE

underlines all column headers 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= on page 874 .)

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 headers without underlining the spaces between them, by using two hyphens **('--')** as the last line of each column header instead of using HEADLINE.

Featured in: Example 2 on page 951 and Example 8 on page 968

HEADSKIP

writes a blank line beneath all column headers (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.

Featured in: Example 2 on page 951

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 has no effect in the nonwindowing environment or on ODS destinations other than traditional SAS monospace output.

LIST
writes to the SAS log the PROC REPORT code that creates the current report. This listing may differ in these ways from the statements that you submit:

- It shows some defaults that you may 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 windowing environment. In the windowing 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 first of these line size specifications that it finds:

- 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=.

Range: 64-256 (integer)

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Featured in: Example 6 on page 964 and Example 8 on page 968

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 also: For information about special missing values, see the section on missing values in *SAS Language Reference: Concepts*.

Featured in: Example 11 on page 977

NAMED

writes *name*= in front of each value in the report, where *name* is the column header 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.

Featured in: Example 7 on page 966

NOALIAS

lets you use a report that was created before compute blocks required aliases (before Release 6.11). If you use NOALIAS, then you cannot use aliases in compute blocks.

NOCENTER

See CENTER|NOCENTER on page 873.

NOCOMPLETECOLS

See COMPLETECOLS|NOCOMPLETECOLS on page 874.

NOCOMPLETEROWS

See COMPLETEROWS|NOCOMPLETEROWS on page 874.

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.

NOHEADER

suppresses column headers, including those that span multiple columns.

When you suppress the display of column headers in the windowing environment, you cannot select any report items.

NOTHREADS

See THREADS | NOTHREADS on page 883.

NOWINDOWS

Alias: NOWD

See WINDOWS|NOWINDOWS on page 884.

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 detail row of the report 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 forth) 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, this is not possible 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 forth).

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 headers 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. 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.

Interaction: You cannot use OUT= in a PROC REPORT step that uses a BY statement.

Featured in: Example 12 on page 980 and Example 13 on page 983

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 may 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

  FOOTNOTE

  FREQ

  TITLE

  WEIGHT

  WHERE

- It omits these PROC REPORT statement options:

  LIST

  NOALIAS

  OUT=

OUTREPT=

PROFILE=

REPORT=

WINDOWS|NOWINDOWS

- □ It omits SAS system options.
- □ It resolves automatic macro variables.

Note: The current version of SAS will correctly read REPORT entries that were created with earlier versions. However, earlier versions of SAS will not correctly read REPORT entries that are created with the current version. △

Featured in: Example 7 on page 966

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 kind 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.

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output. However, the COLUMNS= option in the ODS PRINTER or ODS PDF statement produces similar results. For details, see the chapter on ODS statements in *SAS Output Delivery System: User's Guide*.

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 also: For information about the space between panels and the line size, see the discussions of PSPACE= on page 881 and the discussion of LS= on page 877.

Featured in: Example 8 on page 968

PCTLDEF=

See QNTLDEF= on page 882.

PROFILE=*libref.catalog*

identifies the report profile to use. A profile

- □ specifies the location of menus that define alternative menu bars and pull-down 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 a windowing environment. To create a profile

1. Invoke PROC REPORT with the WINDOWS option.
2. Select

   Tools ▶ Report Profile

3. Fill in the fields to suit your needs.
4. 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 the limit.

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:

- 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 on report profiles, see "PROFILE" on page 926.

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.

Featured in: Example 6 on page 964 and Example 8 on page 968

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.

Featured in: Example 8 on page 968

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. This is the technique that PROC UNIVARIATE uses.

Note: This technique can be very memory intensive. △

P2

uses the P^2 method to approximate the quantile.

Default: OS

Restriction: When QMETHOD=P2, PROC REPORT does not compute 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 those 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.

Default: 5

Alias: PCTLDEF=

Main discussion: "Quantile and Related Statistics" on page 1345

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 also: OUTREPT= on page 879

Featured in: Example 7 on page 966

SHOWALL

overrides options in the DEFINE statement that suppress the display of a column.

See also: NOPRINT and NOZERO in "DEFINE Statement" on page 897

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.

Interaction: 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.

Interaction: When CENTER is in effect, PROC REPORT ignores spacing that precedes the leftmost variable in the report.

Featured in: Example 2 on page 951

SPLIT=*'character'*
specifies the split character. PROC REPORT breaks a column header when it reaches that character and continues the header on the next line. The split character itself is not part of the column header although each occurrence of the split character counts toward the 256-character maximum for a label.

Default: slash (/)

Interaction: The FLOW option in the DEFINE statement honors the split character.

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Featured in: Example 5 on page 960

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 863 for details.

Restriction: This option affects only the HTML, RTF, and Printer output.

Featured in: Example 15 on page 989 and Example 16 on page 994

THREADS | NOTHREADS
enables or disables parallel processing of the input data set. This option overrides the SAS system option THREADS | NOTHREADS. See *SAS Language Reference: Concepts* for more information about parallel processing.

Default: value of SAS system option THREADS | NOTHREADS.

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 use THREADS in the PROC REPORT statement to force PROC REPORT to use parallel processing in these situations.

VARDEF=*divisor*
specifies the divisor to use in the calculation of the variance and standard deviation. Table 42.5 on page 883 shows the possible values for *divisor* and associated divisors.

Table 42.5 Possible Values for VARDEF=

Value	Divisor	Formula for Divisor
DF	degrees of freedom	$n - 1$
N	number of observations	n

Value	Divisor	Formula for Divisor
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 $\sum (x_i - \overline{x})^2$. When you weight the analysis variables, CSS equals $\sum w_i (x_i - \overline{x}_w)^2$, where $\overline{x}_w$ 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=.

Tip: 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.

Tip: When you use the WEIGHT statement and VARDEF=WGT, the computed variance is asymptotically (for large *n*) an estimate of $\sigma^2/\overline{w}$, where $\overline{w}$ is the average weight. This yields an asymptotic estimate of the variance of an observation with average weight.

See also: "WEIGHT" on page 63

WINDOWS | NOWINDOWS

selects a windowing or nonwindowing environment.

When you use WINDOWS, SAS opens the REPORT window, 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 destination(s).

Alias: WD | NOWD

Restriction: When you use the WINDOWS option, you cannot send procedure output to the HTML, RTF, or 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 windowing environment; if DMS is OFF, then it uses the nonwindowing environment.

See also: For a discussion of the report profile see the discussion of PROFILE= on page 880.

Featured in: Example 1 on page 948

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 headers.

Featured in: Example 7 on page 966

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.

Featured in: Example 4 on page 957 and Example 5 on page 960.

BREAK *location break-variable*</ *option(s)*>;

To do this	Use this option
Specify the color of the break lines in the REPORT window	COLOR=
Double overline each value	DOL*
Double underline each value	DUL*
Overline each value	OL*
Start a new page after the last break line	PAGE
Write a blank line for the last break line	SKIP
Specify a style element for default summary lines, customized summary lines or both	STYLE=
Write a summary line in each group of break lines	SUMMARIZE
Suppress 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	SUPPRESS
Underline each value	UL*

* Traditional SAS monospace output only.

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.

Options

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 windowing environment only.

Note: Not all operating environments and devices support all colors, and on some operating systems and devices, one color may 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. △

DOL
(for double overlining) uses the thirteenth 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: equals 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 also: the discussion of FORMCHAR= on page 874.

DUL
(for double underlining) uses the thirteenth 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: equals 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 also: the discussion of FORMCHAR= on page 874.

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 also: the discussion of FORMCHAR= on page 874.

Featured in: Example 2 on page 951 and Example 9 on page 971

PAGE

starts a new page after the last break line.

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.

Featured in: Example 9 on page 971

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.

Featured in: Example 2 on page 951, Example 4 on page 957, Example 5 on page 960, and Example 8 on page 968

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 863 for details.

Restriction: This option affects only the HTML, RTF, and Printer output.

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 kind of report item in a summary line that is created by the BREAK statement:

If the report item is...	Then its value is...
the break variable	the 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	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*

If the report item is...	Then its value is...
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 "COMPUTE Statement" on page 895).

* 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. △

Featured in: Example 2 on page 951, Example 4 on page 957, and Example 9 on page 971

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 895).

Featured in: Example 4 on page 957

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 also: the discussion of FORMCHAR= on page 874.

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)
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 895 and "LINE Statement" on page 907. △

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.

Restriction: You cannot use the OUT= option when you use a BY 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.

Main discussion: "BY" on page 58

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 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*.

Options

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, 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.

CALL DEFINE Statement

Sets the value of an attribute for a particular column in the current row.

Restriction: Valid only in a compute block that is attached to a report item.

Featured in: Example 4 on page 957

CALL DEFINE (*column-id* | _ROW_ , '*attribute-name*', *value*);

The CALL DEFINE statement is often used to write report definitions that other people will use in a windowing 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. (See Table 42.6 on page 891 for descriptions of the available attributes.)

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 42.6 on page 891.

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 Table 42.6 on page 891.

Table 42.6 Attribute Descriptions

Attribute	Description	Values	Affects
BLINK	Controls blinking of current value	1 turns blinking on; 0 turns it off	windowing 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'	windowing environment
COMMAND	Specifies that a series of commands follows	a quoted string of SAS commands to submit to the command line	windowing environment
FORMAT	Specifies a format for the column	a SAS format or a user-defined format	windowing and nonwindowing environments
HIGHLIGHT	Controls highlighting of the current value	1 turns highlighting on; 0 turns it off	windowing environment
RVSVIDEO	Controls display of the current value	1 turns reverse video on; 0 turns it off	windowing environment
STYLE=	Specifies the style element for the Output Delivery System	See "Using the STYLE= Attribute" on page 892	HTML, RTF, and Printer output
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 Printer output

Attribute	Description	Values	Affects
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

* The total length of the URL that you specify (including any characters that come from the BASE= and PATH= options) cannot exceed the line size. Use the LS= option in the PROC REPORT statement to alter the line size for the PROC REPORT step.

\# For information on the BASE= and PATH= options, see the documentation for the ODS HTML statement in *SAS Output Delivery System: User's Guide*.

Note: The attributes BLINK, HIGHLIGHT, and RVSVIDEO do not work on all devices. △

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= attribute 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 instance, 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=[background=yellow font_size=7]");
```

See “Using Style Elements in PROC REPORT” on page 863 for 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", "" );
```

Featured in: Example 16 on page 994

COLUMN Statement

Describes the arrangement of all columns and of headers that span more than one column.

Restriction: You cannot use the COLUMN statement if you use REPORT= in the PROC REPORT statement.

Featured in: Example 1 on page 948, Example 3 on page 954, Example 5 on page 960, Example 6 on page 964, Example 10 on page 975, and Example 11 on page 977

COLUMN *column-specification(s)*;

Required Arguments

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 857 for a list of available statistics.

report-item(s)
identifies items that each form a column in the report.

Featured in: Example 1 on page 948 and Example 11 on page 977

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 header, and the headers are stacked one above the other. The header 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 headers should appear at the same level rather than stacked one above the other.

Featured in: Example 5 on page 960, Example 6 on page 964, and Example 10 on page 975

(*'header-1 '<... 'header-n '> report-item(s)*)
creates one or more headers that span multiple columns.

header
is a string of characters that spans one or more columns in the report. PROC REPORT prints each header on a separate line. You can use split characters in a header to split one header over multiple lines. See the discussion of SPLIT= on page 883.

In traditional (monospace) SAS output, if the first and last characters of a header are one of the following characters, then PROC REPORT uses that character to expand the header to fill the space over the column or columns:

```
:- = \_ .* +
```

Similarly, if the first character of a header is < and the last character is >, or vice-versa, then PROC REPORT expands the header to fill the space over the column by repeating the first character before the text of the header and the last character after it.

Note: A hexadecimal value (such as `'DF'x`) that is specified within *header* 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 *header* in double quotation marks (" ") so that the macro function will resolve. △

report-item(s)
specifies the columns to span.

Featured in: Example 10 on page 975

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 headers. 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.

Featured in: Example 3 on page 954

Note: You cannot always use an alias. When you refer in a compute block to a report item that has an alias, you must usually 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 859). △

COMPUTE Statement

Starts a *compute block*. A compute block contains 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.

Featured in: Example 2 on page 951, Example 3 on page 954, Example 4 on page 957, Example 5 on page 960, Example 9 on page 971, and Example 10 on page 975

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;

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 859.

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 936).
- □ 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 936).

- 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. △

Featured in: Example 3 on page 954 and Example 9 on page 971

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.

Featured in: Example 4 on page 957 and Example 5 on page 960

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. △

Options

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 863 for details.

Restriction: This option affects only the HTML, RTF, and Printer destinations.

Featured in: Example 16 on page 994

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>
except in Printer and RTF output, 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

Featured in: Example 9 on page 971

type-specification

specifies the type and, optionally, 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

Featured in: Example 10 on page 975

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.

Featured in: Example 10 on page 975

DEFINE Statement

Describes how to use and display a report item.

Tip: If you do not use a DEFINE statement, then PROC REPORT uses default characteristics.

Featured in: Example 2 on page 951, Example 3 on page 954, Example 4 on page 957, Example 5 on page 960, Example 6 on page 964, Example 9 on page 971, and Example 10 on page 975

DEFINE *report-item* / <*option(s)*>;

To do this	Use this option
Specify how to use a report item (see "Usage of Variables in a Report" on page 853)	
Define the item, which must be a data set variable, as an across variable	ACROSS
Define the item, which must be a data set variable, as an analysis variable	ANALYSIS
Define the item as a computed variable	COMPUTED
Define the item, which must be a data set variable, as a display variable	DISPLAY

To do this	Use this option
Define the item, which must be a data set variable, as a group variable	GROUP
Define the item, which must be a data set variable, as an order variable	ORDER
Customize the appearance of a report item	
Exclude all combinations of the item that are not found in the preloaded range of user-defined formats	EXCLUSIVE
Assign a SAS or user-defined format to the item	FORMAT=
Reference a HELP or CBT entry that contains Help information for the report item	ITEMHELP=
Consider missing values as valid values for the item	MISSING
Order the values of a group, order, or across variable according to the specified order	ORDER=
Specify that all formats are preloaded for the item.	PRELOADFMT
For traditional SAS monospace output, define the number of blank characters to leave between the column being defined and the column immediately to its left	SPACING=
Associate a statistic with an analysis variable	*statistic*
Specify a style element (for the Output Delivery System) for the report item	STYLE=
Specify a numeric variable whose values weight the value of the analysis variable	WEIGHT=
Define the width of the column in which PROC REPORT displays the report item	WIDTH=
Specify options for a report item	
Reverse the order in which PROC REPORT displays rows or values of a group, order, or across variable	DESCENDING
Wrap the value of a character variable in its column	FLOW
Specify that the item that you are defining is an ID variable	ID
Suppress the display of the report item	NOPRINT
Suppress the display of the report item if its values are all zero or missing	NOZERO
Insert a page break just before printing the first column containing values of the report item	PAGE
Control the placement of values and column headers	
Center the formatted values of the report item within the column width and center the column header over the values	CENTER
Left-justify the formatted values of the report item within the column width and left-justify the column headers over the values	LEFT

To do this	Use this option
Right-justify the formatted values of the report item within the column width and right-justify the column headers over the values	RIGHT
Specify the color in the REPORT window of the column header and of the values of the item that you define	COLOR=
Define the column header for the report item	*column-header*

Required Arguments

report-item
specifies the name or alias (established in the COLUMN statement) of the data set variable, computed variable, or statistic to define.

Note: Do not specify a usage option in the definition of a statistic. The name of the statistic tells PROC REPORT how to use it. △

Options

ACROSS
defines *report-item*, which must be a data set variable, as an across variable. (See “Across Variables” on page 854.)

Featured in: Example 5 on page 960

ANALYSIS
defines *report-item*, which must be a data set variable, as an analysis variable. (See “Analysis Variables” on page 855.)

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 may make your code easier for novice users to understand. △

Featured in: Example 2 on page 951, Example 3 on page 954, and Example 4 on page 957

CENTER
centers the formatted values of the report item within the column width and centers the column header 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 header 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

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 windowing environment only.

Note: Not all operating environments and devices support all colors, and in some operating environments and devices, one color may 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. △

column-header

defines the column header for the report item. Enclose each header in single or double quotation marks. When you specify multiple column headers, PROC REPORT uses a separate line for each one. The split character also splits a column header 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 header is < and the last character is >, or vice-versa, then PROC REPORT expands the header to fill the space over the column by repeating the first character before the text of the header and the last character after it.

Note: A hexadecimal value (such as **'DF'x**) that is specified within *column-header* 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 *column-header* in double quotation marks (" ") so that the macro function will resolve. △

Default:

Item	Header
variable without a label	variable name
variable with a label	variable label
statistic	statistic name

Tip: If you want to use names when labels exist, then submit the following SAS statement before invoking PROC REPORT:

```
options nolabel;
```

Tip: HEADLINE underlines all column headers and the spaces between them. In traditional (monospace) SAS output, you can underline column headers without underlining the spaces between them, by using the special characters **'--'** as the last line of each column header instead of using HEADLINE (see Example 4 on page 957).

See also: SPLIT= on page 883

Featured in: Example 3 on page 954, Example 4 on page 957, and Example 5 on page 960

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 windowing 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.

Featured in: Example 5 on page 960 and Example 10 on page 975

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 853.)

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.

Featured in: Example 10 on page 975

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 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 windowing 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.

Featured in: Example 2 on page 951 and Example 6 on page 964

GROUP

defines *report-item*, which must be a data set variable, as a group variable. (See "Group Variables" on page 854.)

Featured in: Example 4 on page 957, Example 6 on page 964, and Example 14 on page 986

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.

Featured in: Example 6 on page 964

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 a windowing 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 headers 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.

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.

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 on page 971).

Interaction: 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 859).

Interaction: SHOWALL in the PROC REPORT statement or the ROPTIONS window overrides all occurrences of NOPRINT.

Featured in: Example 3 on page 954, Example 9 on page 971, and Example 12 on page 980

NOZERO

suppresses the display of the report item if its values are all zero or missing.

Interaction: 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 859).

Interaction: 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 853.)

Featured in: Example 2 on page 951

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.

Featured in: Example 2 on page 951

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 may 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 you expect even if the default changes. △

PAGE

inserts a page break just before printing the first column containing values of the report item.

Interaction: PAGE is ignored if you use WRAP in the PROC REPORT statement or in the ROPTIONS window.

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.

Interaction: 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.

Interaction 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 headers 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.

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.

Interaction: When PROC REPORT's CENTER option is in effect, PROC REPORT ignores spacing that precedes the leftmost variable in the report.

Interaction: 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 kind of variable.

See "Statistics That Are Available in PROC REPORT" on page 857 for a list of available statistics.

Default: SUM

Featured in: Example 2 on page 951, Example 3 on page 954, and Example 4 on page 957

Note: PROC REPORT uses the name of the analysis variable as the default header for the column. You can customize the column header with the *column-header* option in the DEFINE statement. △

STYLE<(*location(s)*)>=<*style-element-name*><[*style-attribute-specification(s)*]>

specifies the style element to use for column headers and for text inside cells for this report item. See "Using Style Elements in PROC REPORT" on page 863 for details.

Restriction: This option affects only the HTML, RTF, and Printer destinations.

Featured in: Example 16 on page 994

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.

Restriction: to compute weighted quantiles, use QMETHOD=OS in the PROC REPORT statement.

Tip: When you use the WEIGHT= option, consider which value of the VARDEF= option in the PROC REPORT statement is appropriate.

Tip: Use the WEIGHT= option in separate variable definitions in order to specify different weights for the variables.

Note: Prior to Version 7 of SAS, the REPORT procedure did not exclude the observations with missing weights from the count of observations. △

WIDTH=*column-width*

defines the width of the column in which PROC REPORT displays *report-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 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.

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.

Featured in: Example 10 on page 975

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.

ENDCOMP;

See also: COMPUTE statement

Featured in: Example 2 on page 951

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 also: For an example that uses the FREQ statement, see “Example” on page 62

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 *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.

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.

Restriction: This statement is valid only in a compute block that is associated with a location in the report.

Restriction: 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.

Featured in: Example 2 on page 951, Example 3 on page 954, and Example 9 on page 971

LINE *specification(s)*;

Required Arguments

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 859.

item-format
is a SAS format or user-defined format. You must specify a format for each item.

Featured in: Example 2 on page 951

'*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. △

Featured in: Example 2 on page 951

number-of-repetitions'character-string* '
specifies a character string and the number of times to repeat it.

Featured in: Example 3 on page 954

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 on the HTML, RTF, or Printer output.

Featured in: Example 3 on page 954 and Example 5 on page 960

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 equals 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.

Featured in: Example 1 on page 948 and Example 10 on page 975

RBREAK *location* </ *option(s)*>;

To do this	Use this option
Specify the color of the break lines in the REPORT window	COLOR=
Double overline each value	DOL*
Double underline each value	DUL*
Overline each value	OL*
Start a new page after the last break line of a break located at the beginning of the report	PAGE
Write a blank line for the last break line of a break located at the beginning of the report	SKIP*
Specify a style element (for the Output Delivery System) for default summary lines, customized summary lines, or both	STYLE=
Include a summary line as one of the break lines	SUMMARIZE
Underline each value	UL*

* Traditional SAS monospace output only.

Required Arguments

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.

Options

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 windowing environment only.

Note: Not all operating environments and devices support all colors, and in some operating environments and devices, one color may 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. △

DOL
(for double overlining) uses the thirteenth 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: equals 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 also: the discussion of FORMCHAR= on page 874.

Featured in: Example 1 on page 948

DUL

(for double underlining) uses the thirteenth 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: equals 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 also: the discussion of FORMCHAR= on page 874.

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 also: the discussion of FORMCHAR= on page 874.

Featured in: Example 10 on page 975

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 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 863 for details.

Restriction: This option affects only the HTML, RTF, and Printer destinations.

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 kind of report item in a summary line created by the RBREAK statement:

If the report item is…	Then its value is…
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 DEFINE statement. 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 "COMPUTE Statement" on page 895).

Featured in: Example 1 on page 948 and Example 10 on page 975

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 also: the discussion of FORMCHAR= on page 874.

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 "COMPUTE Statement" on page 895 and "LINE Statement" on page 907. △

WEIGHT Statement

Specifies weights for analysis variables in the statistical calculations.

See also: For information about calculating weighted statistics see “Calculating Weighted Statistics” on page 64. For an example that uses the WEIGHT statement, see “Weighted Statistics Example” on page 65.

WEIGHT *variable*;

Required Arguments

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

Weight value…	PROC REPORT…
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.

Tip: When you use the WEIGHT statement, consider which value of the VARDEF= option is appropriate. See VARDEF= on page 883 and the calculation of weighted statistics in “Keywords and Formulas” on page 1340 for more information.

Note: Prior to Version 7 of SAS, the procedure did not exclude the observations with missing weights from the count of observations. △

Weight Information Is Not Saved

When you store a report definition, PROC REPORT does not store the WEIGHT statement.

REPORT Procedure Windows

The windowing 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 windowing environment.

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.

Path

Edit ▶ Summarize information

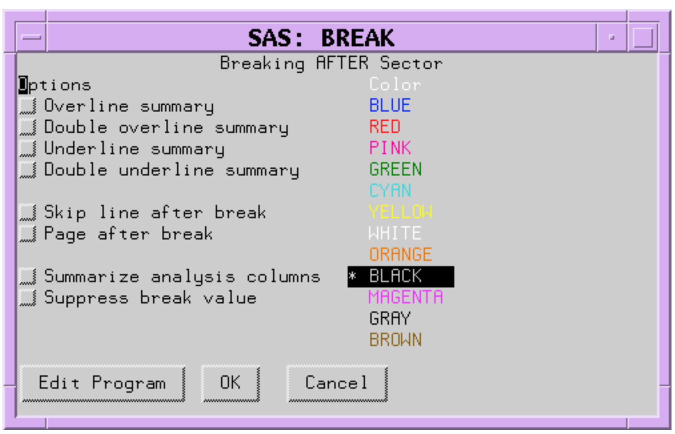
After you select **Summarize Information**, PROC REPORT offers you four choices for the location of the break:

- **Before Item**
- **After Item**
- **At the top**
- **At the bottom.**

After you select a location, the BREAK window opens.

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= on page 874. △

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 thirteenth 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: equals 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 thirteenth 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: equals 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 kind of report item in a summary line created by the BREAK window:

If the report item is...	**Then its value is...**
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 "COMPUTE Statement" on page 895).

*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 header and the values of the item that you are defining.

Default: 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 may 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.

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 859.

COMPUTED VAR

Adds a variable that is not in the input data set to the report.

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 opens.

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.

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 opens.

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.

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.

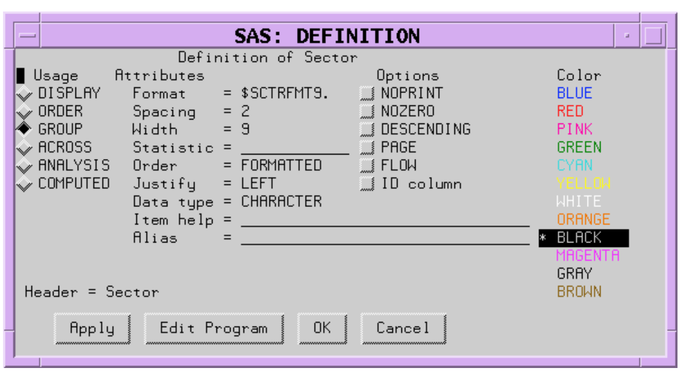
Path

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 852.

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 920) 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 windowing 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 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

Interaction: When PROC REPORT's CENTER option is in effect, PROC REPORT ignores spacing that precedes the leftmost variable in the report.

Interaction: 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.

Range: 1 to the value of the SAS system option LINESIZE=

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=.

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 kind of variable.

Default: SUM

Note: PROC REPORT uses the name of the analysis variable as the default header for the column. You can customize the column header with the **`Header`** field of the DEFINITION window. △

You can use the following values for *statistic*:

Descriptive statistic keywords	
CSS	PCTSUM
CV	RANGE
MAX	STDDEV | STD
MEAN	STDERR
MIN	SUM
N	SUMWGT
NMISS	USS
PCTN	VAR
Quantile statistic keywords	
MEDIAN | Q2 | P50	Q3 | P75
P1	P90
P5	P95
P10	P99
Q1 | P25	QRANGE
Hypothesis testing keyword	
PROBT	T

Explanations of the keywords, the formulas that are used to calculate them, and the data requirements are discussed in Appendix 1, "SAS Elementary Statistics Procedures," on page 1339.

Requirement: To compute standard error and the Student's *t*-test you must use the default value of VARDEF= which is DF.

See also: For definitions of these statistics, see "Keywords and Formulas" on page 1340.

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 may 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 you expect even if the default changes. △

Justify=

You can justify the placement of the column header 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 header 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 header 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 header 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 on page 954).

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 on page 971).

Interaction: 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 859).

Interaction: 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.

Interaction: 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 859).

Interaction: 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 header and the values of the item that you are defining.

Default: 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 may 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.

Path

View ► Display Page

Description

You can get to 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.

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.

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.

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.

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.

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= on page 880).

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.

Path

Specify the PROMPT option when you start PROC REPORT or select PROMPT from the ROPTIONS window. The PROMPTER window opens the next time that 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.

Path

Use WINDOWS or PROMPT in the PROC REPORT statement.

Description

You cannot write directly in any part of the REPORT window except column headers. 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. △

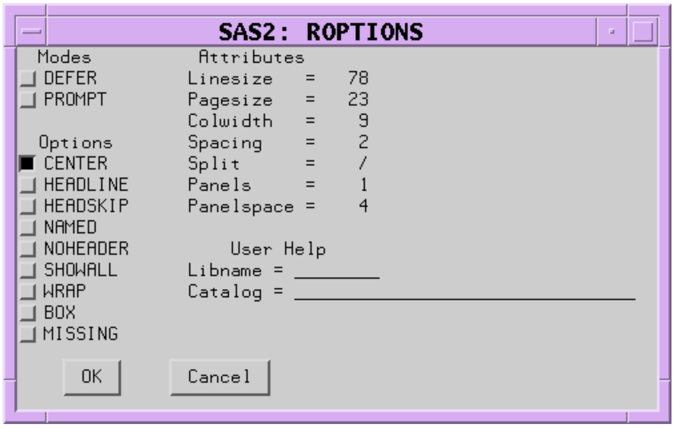
ROPTIONS

Displays choices that control the layout and display of the entire report and identifies the SAS data library and catalog containing CBT or HELP entries for items in the report.

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 that 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 headers 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= on page 874.)

Default: hyphen (-)

Tip: In traditional (monospace) SAS output, you can underline column headers without underlining the spaces between them, by using `'--'` as the last line of each column header instead of using HEADLINE.

HEADSKIP

writes a blank line beneath all column headers (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 header for the value.

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.

Interaction: When you use NAMED, PROC REPORT automatically uses NOHEADER.

NOHEADER

suppresses column headers, including those that span multiple columns.

Once you suppress the display of column headers in the windowing 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 headers.

BOX

uses formatting characters to add line-drawing characters to the report. These characters

- surround each page of the report
- separate column headers 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 also: For information about formatting characters, see the discussion of FORMCHAR= on page 874.

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 windowing 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.

Range: 1 to the linesize

Default: 9

Interaction: 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 919.) If no format is associated with the item, then the column width depends on variable type:

If the variable is a...	Then the column width is the...
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

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

Interaction: 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.

Interaction: 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 header when it reaches that character and continues the header on the next line. The split character itself is not part of the column header 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.

Note: If you are typing over a header (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 DEFINITION or a BREAK window, 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 kind 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 also: For information about specifying the space between panels see the discussion of PSPACE= on page 932. For information about setting the linesize, see the discussion of Linesize on page 931).

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.

Path

File ► Save Data Set

Description

To specify an output data set, enter the name of the SAS data 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.

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 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.

Path

Tools ► Report Statements

STATISTICS

Displays statistics that are available in PROC REPORT.

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 opens.

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.

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 the section on statements in *SAS Language Reference: Dictionary*.

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.

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 the chapter on statements in *SAS Language Reference: Dictionary*.

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.

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 937. The sequence of events is the same whether you use programming statements or the windowing 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, those for detail rows as well as those for summary lines in breaks. Statistics include those computed for analysis variables. PROC REPORT calculates statistics for summary lines whether or not they appear in the report. It stores all this information in a temporary file.
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 temporary file that was created 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 937 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 859. △

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 858 and "COMPUTE Statement" on page 895.

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 Output 42.2 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 Output 42.2. The user-defined formats that are used are created by a PROC FORMAT step on page 949.

```
libname proclib 'SAS-data-library';


options nodate pageno=1 linesize=64
        pagesize=60 fmtsearch=(proclib);
proc report data=grocery headline headskip;
   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 profit;
      if department='np1' or department='np2'
         then profit=0.4*sales.sum;
      else profit=0.25*sales.sum;
   endcomp;

   rbreak after / dol dul summarize;
   compute after;
      sector='TOTALS:';
   endcomp;

   where sector contains 'n';
   title 'Report for Northeast and Northwest Sectors';
run;
```

Output 42.2 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  $1,071.25         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. It stores these values in a temporary file.

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 Figure 42.9 on page 939 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 42.9 First Detail Row with Values Initialized

Sector	Department	Sales	Profit	N
		.	.	.

3 Figure 42.10 on page 939 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 temporary file that is created at the beginning of the report-building process.

Figure 42.10 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 Figure 42.11 on page 940.

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 42.11 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 temporary file created at the beginning of the report-building process. Figure 42.12 on page 940 illustrates the completed row.

Figure 42.12 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 Figure 42.13 on page 940).

Figure 42.13 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 Figure 42.14 on page 941.

Figure 42.14 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 Output 42.3 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 Output 42.3. The user-defined formats that are used are created by a PROC FORMAT step on page 949.

Note: Calculations of the percentages do not multiply their results by 100 because PROC REPORT prints them with the PERCENT. format. △

```
libname proclib 'SAS-data-library';


options nodate pageno=1 linesize=64
        pagesize=60 fmtsearch=(proclib);
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;
```

Output 42.3 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. It stores these values in a temporary file.

2 PROC REPORT initializes the temporary variable, Sctrtot, to missing (see Figure 42.15 on page 943).

Figure 42.15 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 temporary file that it created 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 Figure 42.16 on page 944).

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 42.16 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 headers. (The NOHEADER option in the PROC REPORT statement suppresses the default column headers, 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 temporary file that it created 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 Figure 42.17 on page 944).

Figure 42.17 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. Figure 42.18 on page 945 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 42.18 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 detail row of the report. It initializes all report variables to missing. Values for temporary variables do not change. Figure 42.19 on page 945 illustrates the first detail row at this point.

Figure 42.19 First Detail Row with Initialized Values

Report Variables					Temporary Variable
Sector	**Department**	**Sales.sum**	**Sctrpct**	**Sales.pctsum**	**Sctrtot**
		.	.	.	$1,831.00

9 Figure 42.20 on page 945 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 temporary file that it created at the beginning of the report-building process.

Figure 42.20 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 Figure 42.21 on page 946.

Figure 42.21 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 temporary file. The first detail row is now complete (see Figure 42.22 on page 946).

Figure 42.22 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 Figure 42.23 on page 947). The statements in the compute block for this break calculate the correct value (see the following step).

Figure 42.23 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. Figure 42.24 on page 947 shows the final summary line.

Figure 42.24 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

Procedure 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

This example uses a permanent data set and permanent formats to create a report that contains

- one row for every observation
- a default summary for the whole report.

Program

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

```
libname proclib 'SAS-data-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=64 pagesize=60;
```

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 data 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(s).

```
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               1
                           for 04JAN02

                 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

Procedure 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 on page 949

Formats: $MGRFMT. and $DEPTFMT. on page 949

This example

- 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 headers 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

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

```
libname proclib 'SAS-data-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. FMTSEARCH= specifies the library to include when searching for user-created formats.

```
options nodate pageno=1 linesize=64 pagesize=60
        fmtsearch=(proclib);
```

Specify the report options. The NOWD option runs PROC REPORT without the REPORT window and sends its output to the open output destination(s). COLWIDTH=10 sets the default column width to 10 characters. SPACING= puts five blank characters between columns. HEADLINE underlines all column headers 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              1

                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

Procedure features:

COLUMN statement:
- with aliases

COMPUTE statement arguments:
- AFTER

DEFINE statement options:
- ANALYSIS
- MAX
- MIN
- NOPRINT
- customizing column headers

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 on page 949

Formats: $MGRFMT. and $DEPTFMT. on page 949

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

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

```
libname proclib 'SAS-data-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. FMTSEARCH= specifies the library to include when searching for user-created formats.

```
options nodate pageno=1 linesize=64 pagesize=60
        fmtsearch=(proclib);
```

Specify the report options. The NOWD option runs PROC REPORT without the REPORT window and sends its output to the open output destination(s). HEADLINE underlines all column headers 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                1
                         for 04JAN02

                 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

Procedure 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 headers

LINE statement:

quoted text
variable values

Data set: GROCERY on page 949

Formats: $MGRFMT. and $DEPTFMT. on page 949

This example creates a summary report that

- 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 headers.

Program

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

```
libname proclib 'SAS-data-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. FMTSEARCH= specifies the library to include when searching for user-created formats.

```
options nodate pageno=1 linesize=64 pagesize=60
        fmtsearch=(proclib);
```

Specify the report options. The NOWD option runs PROC REPORT without the REPORT window and sends its output to the open output destination(s). 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           1

                  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

Procedure 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 on page 949

Formats: $SCTRFMT., $MGRFMT., and $DEPTFMT. on page 949

The report in this example

- □ 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 headers, 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

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

```
libname proclib 'SAS-data-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. FMTSEARCH= specifies the library to include when searching for user-created formats.

```
options nodate pageno=1 linesize=64 pagesize=60
        fmtsearch=(proclib);
```

Specify the report options. The NOWD option runs PROC REPORT without the REPORT window and sends its output to the open output destination(s). 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 header, but the header 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 the group variables. 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. Text in quotation marks in the DEFINE statements specifies column headings. These statements illustrate two ways to write a blank line in a column header. **`'Sector' ''`** writes a blank line because each quoted string is a line of the column heading. The two adjacent quotation marks write a blank line for the second line of the heading. **`'Manager* '`** writes a blank line because the split character (*) starts a new line of the heading. That line contains only a blank.

```
   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;
```

Select the observations to process. The WHERE statement selects for the report only the observations for departments **p1** and **p2** in stores in the northeast or northwest sector.

```
   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       1

                         _______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

   ---------------------------------------------------------
   |   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

Procedure features:

PROC REPORT statement options:

LS=
PS=

COLUMN statement:

specifying statistics for stacked variables

DEFINE statement options:

FORMAT=
GROUP
ID

Data set: GROCERY on page 949

Formats: $MGRFMT. on page 949

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

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

```
libname proclib 'SAS-data-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. FMTSEARCH= specifies the library to include when searching for user-created formats.

```
options nodate pageno=1 linesize=80 pagesize=60
        fmtsearch=(proclib);
```

Specify the report options. The NOWD option runs PROC REPORT without the REPORT window and sends its output to the open output destination(s). 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              1

                                 Sum          Min          Max
     Sector     Manager        Sales        Sales        Sales
     -----------------------------------------------------------

     Northeast  Alomar       $786.00       $86.00      $420.00
                Andrews    $1,045.00      $125.00      $420.00
     Northwest  Brown        $598.00       $45.00      $250.00
                Pelfrey      $746.00       $45.00      $420.00
                Reveiz     $1,110.00       $30.00      $600.00
     Southeast  Jones        $630.00       $40.00      $300.00
                Smith        $350.00       $50.00      $120.00
     Southwest  Adams        $695.00       $40.00      $350.00
                Taylor       $353.00       $50.00      $130.00
```

```
                 Sales Statistics for All Sectors              2

                               Range         Mean          Std
     Sector     Manager        Sales        Sales        Sales
     -----------------------------------------------------------

     Northeast  Alomar       $334.00      $196.50      $156.57
                Andrews      $295.00      $261.25      $127.83
     Northwest  Brown        $205.00      $149.50      $105.44
                Pelfrey      $375.00      $186.50      $170.39
                Reveiz       $570.00      $277.50      $278.61
     Southeast  Jones        $260.00      $157.50      $123.39
                Smith         $70.00       $87.50       $29.86
     Southwest  Adams        $310.00      $173.75      $141.86
                Taylor        $80.00       $88.25       $42.65
```

Example 7: Storing and Reusing a Report Definition

Procedure features:

PROC REPORT statement options:

NAMED
OUTREPT=
REPORT=
WRAP

Other features:

TITLE statement

WHERE statement

Data set: GROCERY on page 949

Formats: $SCTRFMT., $MGRFMT. and $DEPTFMT. on page 949

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 header.

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

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

```
libname proclib 'SAS-data-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. FMTSEARCH= specifies the library to include when searching for user-created formats.

```
options nodate pageno=1 linesize=80 pagesize=60
        fmtsearch=(proclib);
```

Specify the report options. The NOWD option runs PROC REPORT without the REPORT window and sends its output to the open output destination(s). 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

This is the output from the first PROC REPORT step, which creates the report definition.

```
              Sales Figures for Smith on 04JAN02            1

    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

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. FMTSEARCH= specifies the library to include when searching for user-created formats.

```
options nodate pageno=1 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 04JAN02       1

   Sector=Southwest  Manager=Taylor   Department=Paper
   Sector=Southwest  Manager=Taylor   Department=Meat/Dairy
   Sector=Southwest  Manager=Taylor   Department=Canned
   Sector=Southwest  Manager=Taylor   Department=Produce
   Sector=Southwest  Manager=Adams    Department=Paper
   Sector=Southwest  Manager=Adams    Department=Meat/Dairy
   Sector=Southwest  Manager=Adams    Department=Canned
   Sector=Southwest  Manager=Adams    Department=Produce
```

```
      Sales Figures for the Southwest Sector on 04JAN02       2

                    Sales=     $53.00
                    Sales=    $130.00
                    Sales=    $120.00
                    Sales=     $50.00
                    Sales=     $40.00
                    Sales=    $350.00
                    Sales=    $225.00
                    Sales=     $80.00
```

Example 8: Condensing a Report into Multiple Panels

Procedure features:

PROC REPORT statement options:

FORMCHAR=
HEADLINE
LS=
PANELS=
PS=
PSPACE=

BREAK statement options:

SKIP

Other features:

SAS system option FORMCHAR=

Data set: GROCERY on page 949

Formats: $MGRFMT. and $DEPTFMT. on page 949

The report in this example

- 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

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

```
libname proclib 'SAS-data-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. FMTSEARCH= specifies the library to include when searching for user-created formats.

```
options nodate pageno=1 linesize=80 pagesize=60
        fmtsearch=(proclib);
```

Specify the report options. The NOWD option runs PROC REPORT without the REPORT window and sends its output to the open output destination(s). 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                  1

 Manager  Department    Sales      Manager  Department    Sales
 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~      ~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 Adams    Paper        $40.00
          Canned      $225.00      Reveiz   Paper        $60.00
          Meat/Dairy  $350.00               Canned      $420.00
          Produce      $80.00               Meat/Dairy  $600.00
                                            Produce      $30.00
 Brown    Paper        $45.00
          Canned      $230.00      Taylor   Paper        $53.00
          Meat/Dairy  $250.00               Canned      $120.00
          Produce      $73.00               Meat/Dairy  $130.00
                                            Produce      $50.00
 Pelfrey  Paper        $45.00
          Canned      $420.00
          Meat/Dairy  $205.00
          Produce      $76.00
```

Example 9: Writing a Customized Summary on Each Page

Procedure 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 on page 949

Formats: $SCTRFMT., $MGRFMT., and $DEPTFMT. on page 949

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 header that identifies the sector and the store's manager.

The header 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

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

```
libname proclib 'SAS-data-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. FMTSEARCH= specifies the library to include when searching for user-created formats.

```
options nodate pageno=1 linesize=64 pagesize=30
        fmtsearch=(proclib);
```

Specify the report options. The NOWD option runs PROC REPORT without the REPORT window and sends its output to the open output destination(s). 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 header. 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                  1

Northeast Sector
Store managed by Alomar



              Department        Sales       Profit
              ------------------------------------

              Canned          $420.00      $168.00
              Meat/Dairy      $190.00       $47.50
              Paper            $90.00       $36.00
              Produce          $86.00       $21.50
                          -----------  -----------
                              $786.00      $196.50

                Sales are in the target region.
```

```
                   Sales for Individual Stores                  2

Northeast Sector
Store managed by Andrews



              Department        Sales       Profit
              ------------------------------------

              Canned          $420.00      $168.00
              Meat/Dairy      $300.00       $75.00
              Paper           $200.00       $80.00
              Produce         $125.00       $31.25
                          -----------  -----------
                            $1,045.00      $261.25

                      Sales exceeded goal!
```

Example 10: Calculating Percentages

Procedure features:

COLUMN statement arguments:
- PCTSUM
- SUM
- spanning headers

COMPUTE statement options:
- CHAR
- LENGTH=

DEFINE statement options:
- COMPUTED
- FLOW
- WIDTH=

RBREAK statement options:
- OL
- SUMMARIZE

Other features:

TITLE statement

Data set: GROCERY on page 949

Formats: $MGRFMT. and $DEPTFMT. on page 949

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 header. A single header also spans all the columns. This header 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 header.

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

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

```
libname proclib 'SAS-data-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. FMTSEARCH= specifies the library to include when searching for user-created formats.

```
options nodate pageno=1 linesize=64 pagesize=60
        fmtsearch=(proclib);
```

Specify the report options. The NOWD option runs PROC REPORT without the REPORT window and sends its output to the open output destination(s). HEADLINE underlines all column headings and the spaces between them at the top of each page of the report. The null TITLE statement suppresses the title of the report.

```
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

```
                                                              1

      Individual Store Sales as a Percent of All Sales

                          Total   Percent
Sector     Manager        Sales  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

Procedure features:

PROC REPORT statement options:

MISSING

COLUMN statement
with the N statistic

Other features:
TITLE statement

Formats: $MGRFMT. on page 949

This example illustrates the difference between the way 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

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

```
libname proclib 'SAS-data-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. FMTSEARCH= specifies the library to include when searching for user-created formats.

```
options nodate pageno=1 linesize=64 pagesize=60
        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 destination(s). 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         1

             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

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         2

            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

Procedure features:

PROC REPORT statement options:

BOX
OUT=

DEFINE statement options:

ANALYSIS
GROUP
NOPRINT
SUM

Other features:

Data set options:

WHERE=

Data set: GROCERY on page 949

Formats: $MGRFMT. on page 949

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

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

```
libname proclib 'SAS-data-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. FMTSEARCH= specifies the library to include when searching for user-created formats.

```
options nodate pageno=1 linesize=64 pagesize=60
        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 destination(s). 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

This is the output data set that PROC REPORT creates. It is used as the input set in the second PROC REPORT step.

```
                     The Data Set TEMP                       1

    Manager       Sales  ____________BREAK_____________
    3              1110
    8              1045
```

Program That Uses the Output Data Set

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                  1
                             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

Procedure 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 on page 949

Formats: $SCTRFMT. on page 949

The report in this example

- 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

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

```
libname proclib 'SAS-data-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. FMTSEARCH= specifies the library to include when searching for user-created formats.

```
options nodate pageno=1 linesize=64 pagesize=60
        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 destination(s). 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

This is the output data set that is created by PROC REPORT. It is used as input for PROC CHART.

```
                     The Data Set PROFIT                          1

Sector    Manager   Department       Sales      Profit  _BREAK__
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
sw        6         p1                 350        87.5
sw        6         np2                225          90
sw        6         p2                  80          20
ne        7         np1                 90          36
ne        7         p1                 190        47.5
ne        7         np2                420         168
ne        7         p2                  86        21.5
ne        8         np1                200          80
ne        8         p1                 300          75
ne        8         np2                420         168
ne        8         p2                 125       31.25
```

Program That Uses the Output Data Set

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. FMTSEARCH= specifies the library to include when searching for user-created formats.

```
options nodate pageno=1 linesize=80 pagesize=60
        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

```
                    Sum of Profit by Sector                            1

                    Sum of Profit by Sector

                        ___
                       /_ /|
          ___         |**| |
         /_ /|        |**| |
        |**| |        |**| |
        |**| |        |**| |
        |**| |        |**| |          ___           ___
       -|**| |--------|**| |---------/_ /|---------/_ /|-------
      / |**| |      / |**| |      / |**| |      / |**| |      /
     /  |**| |     /  |**| |     /  |**| |     /  |**| |     /
    /   |**| |    /   |**| |    /   |**| |    /   |**| |    /
   /    |**|/    /    |**|/    /    |**|/    /    |**|/    /
  /             /             /             /             /
 /     $627.25 /     $796.50 /     $309.50 /     $327.70 /
/-------------/-------------/-------------/-------------/

  Northeast     Northwest     Southeast     Southwest

                        Sector
```

Example 14: Using a Format to Create Groups

Procedure features:
DEFINE statement options:
GROUP

Other features: FORMAT procedure

Data set: GROCERY on page 949

Formats: $MGRFMT. on page 949

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 header is the formatted value of the variable.

Program

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

```
libname proclib 'SAS-data-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. FMTSEARCH= specifies the library to include when searching for user-created formats.

```
options nodate pageno=1 linesize=64 pagesize=60
        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 destination(s). 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                 1


           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

Procedure features: STYLE= option in the PROC REPORT statement

Other features:

ODS HTML statement

ODS PDF statement

ODS RTF statement

Data set: GROCERY on page 949

Formats: $MGRFMT. and $DEPTFMT. on page 949

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

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

```
libname proclib 'SAS-data-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. FMTSEARCH= specifies the library to include when searching for user-created formats. LINESIZE= and PAGESIZE= are not set for this example because they have no effect on HTML, RTF, and Printer output.

```
options nodate pageno=1 fmtsearch=(proclib);
```

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 REPORT 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 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 those that are specified here.

```
style(header)=[foreground=yellow
               font_style=italic font_size=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 those that are specified here.

```
style(column)=[foreground=moderate brown
               font_face=helvetica font_size=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 those that are specified here.

```
style(lines)=[foreground=white background=black
              font_style=italic font_weight=bold font_size=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 those that are specified here.

```
style(summary)=[foreground=cx3e3d73 background=cxaeadd9
                font_face=helvetica font_size=3 just=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 html close;
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

Procedure features:

STYLE= option in

- PROC REPORT statement
- CALL DEFINE statement
- COMPUTE statement
- DEFINE statement

Other features:

ODS HTML statement

ODS PDF statement

ODS RTF statement

Data set: GROCERY on page 949

Formats: $MGRFMT. on page 949 and $DEPTFMT. on page 949

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

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

```
libname proclib 'SAS-data-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. FMTSEARCH= specifies the library to include when searching for user-created formats. LINESIZE= and PAGESIZE= are not set for this example because they have no effect on HTML, RTF, and Printer output.

```
options nodate pageno=1 fmtsearch=(proclib);
```

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 REPORT 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 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 those 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 those that are specified here.

```
style(header)=[foreground=yellow
               font_style=italic font_size=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 those that are specified here.

```
style(column)=[foreground=moderate brown
               font_face=helvetica font_size=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 those that are specified here.

```
style(lines)=[foreground=white background=black
              font_style=italic font_weight=bold font_size=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 those that are specified here.

```
style(summary)=[foreground=cx3e3d73 background=cxaeadd9
                font_face=helvetica font_size=3 just=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 those that were established for the HEADER location in the PROC REPORT statement.

```
                    style(header)=[foreground=white
                                   background=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 those that were established for the COLUMN location in the PROC REPORT statement.

```
                    style(column)=[font_style=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=[font_style=roman font_size=3 font_weight=bold
          background=white foreground=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=[background=yellow
                       font_face=helvetica
                       font_weight=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 html close;
ods pdf close;
ods rtf close;
```

HTML Body File

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.		

CHAPTER

43

The 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 1016.

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. Refer to the SAS documentation for your operating environment for information about other sorting capabilities △

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. Output 43.1 shows the log that results from running this PROC SORT step. Output 43.2 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;
```

Output 43.1 SAS Log Generated by PROC SORT

```
NOTE: There were 6 observations read from the data set WORK.EMPLOYEE.
NOTE: The data set WORK.EMPLOYEE has 6 observations and 3 variables.
NOTE: PROCEDURE SORT used:
      real time           0.01 seconds
      cpu time            0.01 seconds
```

Output 43.2 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 on page 1019.

Output 43.3 Observations Sorted by the Values of Three Variables

```
                  Customers with Past-Due Accounts                  1
               Listed by Town, Amount, Account Number

                                                          Account
   Obs    Company                   Town              Debt     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
```

Syntax: SORT Procedure

Requirements: BY statement

Reminder: You can use the ATTRIB, FORMAT, LABEL, and WHERE statements. See Chapter 3, “Statements with the Same Function in Multiple Procedures,” on page 57 for details. You can also use any global statements. See “Global Statements” on page 18 for a list.

See: SORT Procedure in the documentation for your operating environment.

PROC SORT <*collating-sequence-option*> <*other option(s)*>;
  BY <DESCENDING> *variable-1* <…<DESCENDING> *variable-n*>;

PROC SORT Statement

PROC SORT <*collating-sequence-option*> <*other option(s)*>;

To do this	Use this option
Specify the collating sequence	
Specify ASCII	ASCII
Specify EBCDIC	EBCDIC

To do this	Use this option
Specify Danish	DANISH
Specify Finnish	FINNISH
Specify Norwegian	NORWEGIAN
Specify Swedish	SWEDISH
Specify a customized sequence	NATIONAL
Specify any of these collating sequences: ASCII, EBCDIC, DANISH, FINNISH, ITALIAN, NORWEGIAN, SPANISH, SWEDISH	SORTSEQ=
Specify the input data set	DATA=
Sort a SAS data set without changing the created and modified dates	DATECOPY
Create output data sets	
Specifies the output data set	OUT=
Specifies the output data set to which duplicate observations are written	DUPOUT=
Specify the output order	
Reverse the collation order for character variables	REVERSE
Maintain relative order within BY groups	EQUALS
Do not maintain relative order within BY groups	NOEQUALS
Eliminate duplicate observations	
Delete observations with duplicate BY values	NODUPKEY
Delete duplicate observations	NODUPRECS
Delete the input data set before the replacement output data set is populated	OVERWRITE
Specify the available memory	SORTSIZE=
Force redundant sorting	FORCE
Reduce temporary disk usage	TAGSORT
Override SAS system option THREADS	
Enable multi-threaded sorting	THREADS
Prevent multi-threaded sorting	NOTHREADS

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.

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. △

Restriction: You can specify only one *collating-sequence-option* in a PROC SORT step.

ASCII
sorts character variables using the ASCII collating sequence. You need this option only when you sort by ASCII on a system where EBCDIC is the native collating sequence.

See also: "Sorting Orders for Character Variables" on page 1014

DANISH
NORWEGIAN
sorts characters according to the Danish and Norwegian national standard.

The Danish and Norwegian collating sequence is shown in Figure 43.1 on page 1008.

EBCDIC
sorts character variables using the EBCDIC collating sequence. You need this option only when you sort by EBCDIC on a system where ASCII is the native collating sequence.

See also: "Sorting Orders for Character Variables" on page 1014

FINNISH
SWEDISH
sorts characters according to the Finnish and Swedish national standard. The Finnish and Swedish collating sequence is shown in Figure 43.1 on page 1008.

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 if a customized national sort sequence is available.

NORWEGIAN
See DANISH.

SORTSEQ=*collating-sequence*
specifies the collating sequence. The value of *collating-sequence* can be any one of the *collating-sequence-options* in the PROC SORT statement, or the value can be the name of a translation table, either a default translation table or one that you have created in the TRANTAB procedure. For an example of using PROC TRANTAB and PROC SORT with SORTSEQ=, see Using Different Translation Tables for Sorting in *SAS National Language Support (NLS): User's Guide*. The available translation tables are

Danish

Finnish

Italian

Norwegian

Spanish

Swedish

The following figure shows how the alphanumeric characters in each language will sort.

Figure 43.1 National Collating Sequences of Alphanumeric Characters

```
Danish:     0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZÆØÅabcdefghijklmnopqrstuvwxyzæøå
Finnish:    0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZÅÄÖabcdefghijklmnopqrstuvwxyzåäö
Italian:    0123456789AÀBCÇDEÉÈFGHIÌJKLMNOÒPQRSTUÙVWXYZaàbcçdeéèfghiìjklmnoòpqrstuùvwxyz
Norwegian:  0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZÆØÅabcdefghijklmnopqrstuvwxyzæøå
Spanish:    0123456789AÁaáBbCcDdEÉeéFfGgHhIÍiíJjKkLlMmNnÑñOÓoóPpQqRrSsTtUÚuúÜüVvWwXxYyZz
Swedish:    0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZÅÄÖabcdefghijklmnopqrstuvwxyzåäö
```

CAUTION:

If you use a host sort utility to sort your data, then specifying the SORTSEQ= option might corrupt the character BY variables. For more information, see the PROC SORT documentation for your operating environment. △

SWEDISH

See FINNISH.

Other Options

DATA=*SAS-data-set*

identifies the input SAS data set.

Main discussion: "Input Data Sets" on page 19

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 prior to 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 348.

DUPOUT= *SAS-data-set*

specifies the output data set to which duplicate observations are written.

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

Interaction: When you use NODUPRECS or NODUPKEY to remove observations in the output data set, the choice of EQUALS or NOEQUALS can affect which observations are removed.

Interaction: The EQUALS | NOEQUALS procedure option overrides the default sort stability behavior that is established with the SORTEQUALS | NOSORTEQUALS system option.

Interaction: The EQUALS option is supported by the multi-threaded sort. However, I/O performance may be reduced when using the EQUALS option with the multi-threaded sort because partitioned data sets will be processed as if they are non-partitioned data sets.

Interaction: 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. Therefore, using the NOEQUALS option can produce inconsistent results in your output data sets.

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.

Tip: PROC SORT checks for the sort information before it sorts a data set so that data is not re-sorted 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 Language Reference: Dictionary*.

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.

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 those 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 sort, then the observation that is written to the output data set is not always the first observation of the BY group. △

Note: See NODUPRECS for information about eliminating duplicate observations. △

Interaction: 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.

Tip: Use the EQUALS option with the NODUPKEY option for consistent results in your output data sets.

Featured in: Example 4 on page 1023

NODUPRECS

checks for and eliminates duplicate observations. If you specify this option, then PROC SORT compares all variable values for each observation to those for the previous observation that was written to the output data set. If an exact match is found, then the observation is not written to the output data set.

Note: See NODUPKEY for information about eliminating observations with duplicate BY values. △

Alias : NODUP

Interaction: When you are removing consecutive duplicate observations in the output data set with NODUPRECS, the choice of EQUALS or NOEQUALS can have an effect on which observations are removed.

Tip: Use the EQUALS option with the NODUPRECS option for consistent results in your output data sets.

Interaction: The action of NODUPRECS is directly related to the setting of the SORTDUP= system option. When SORTDUP= is set to LOGICAL, NODUPRECS removes duplicate observations based on the examination of the variables that remain after a DROP or KEEP operation on the input data set. Setting SORTDUP=LOGICAL increases the number of duplicate observations that are removed, because it eliminates variables before observation comparisons take place. Also, setting SORTDUP=LOGICAL can improve performance, because dropping variables before sorting reduces the amount of memory required to perform the sort. When SORTDUP= is set to PHYSICAL, NODUPRECS examines all variables in the data set, regardless of whether they have been kept or dropped. For more information about SORTDUP=, see the chapter on SAS system options in *SAS Language Reference: Dictionary*.

Tip: Because NODUPRECS checks only consecutive observations, some nonconsecutive duplicate observations might remain in the output data set. You can remove all duplicates with this option by sorting on all variables.

NOEQUALS
See EQUALS | NOEQUALS.

NOTHREADS
See THREADS|NOTHREADS.

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 OUT=, data could be lost if your system failed during execution of PROC SORT. △

Default: Without OUT=, PROC SORT overwrites the original data set.

Tip : You can use data set options with OUT=.

Featured in: Example 1 on page 1017

OVERWRITE
enables the input data set to be deleted before the replacement output data set is populated with observations.

Restriction: 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.

Restriction: 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.

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. △

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 1014, "ASCII Order" on page 1014, and the SAS documentation for your operating environment. △

Interaction: Using REVERSE with the DESCENDING option in the BY statement restores the sequence to the normal order.

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 a PROC SORT, but you cannot specify both.

See also: The DESCENDING 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 SORTSIZE=. For more information about SORTSIZE=, see the chapter on SAS system options in *SAS Language Reference: Dictionary*.

Operating Environment Information: Some system sort utilities may treat this option differently. Refer to the SAS documentation for your operating environment. △

Default: the value of the SAS system option SORTSIZE=

Tip: 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 that relate to memory and information regarding available memory that is gathered from the operating environment.

Operating Environment Information: For information about the SAS system options that relate to memory, 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.

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 may be much higher.

THREADS | NOTHREADS
enables or prevents the activation of multi-threaded sorting.

Default: the value of the SAS system option THREADS

Interaction: THREADS|NOTHREADS overrides the value of the SAS system option THREADS. For more information about THREADS, see the chapter on SAS system options in *SAS Language Reference: Dictionary*.

Interaction: The THREADS option is honored if the value of the SAS system option CPUCOUNT is greater than 1.

Interaction: The TAGSORT option is not supported by the multi-threaded sort.

Note: If THREADS is specified either as a SAS system option or in PROC SORT, and another program has the input SAS data set open for reading, writing, or updating using the SPDE engine, then the procedure might fail. In this case, PROC SORT stops processing and writes a message to the SAS log.

See also: "Multi-threaded Sorting" on page 1013

BY Statement

Specifies the sorting variables

Featured in: Example 1 on page 1017, Example 2 on page 1019, and Example 4 on page 1023

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

Required Arguments

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.

Option

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.

Featured in: Example 2 on page 1019

Concepts: SORT Procedure

Multi-threaded Sorting

The SAS system option THREADS activates 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 at the possible cost of additional CPU resources. For more information, see the section on "Support for Parallel Processing" in *SAS Language Reference: Concepts*.

The performance of the multi-threaded sort will be affected by the value of the SAS system option CPUCOUNT=. CPUCOUNT= suggests how many system CPUs are available for use by the multi-threaded sort.

The multi-threaded sort supports concurrent input from the partitions of a partitioned data set.

Note: These partitioned data sets should not be confused with partitioned data sets on z/OS. △

Operating Environment Information: For information about the support of partitioned data sets in your operating environment, see the SAS documentation for your operating environment. △

For more information about THREADS and CPUCOUNT=, see the chapter on SAS system options in *SAS Language Reference: Dictionary*.

Using PROC SORT with a DBMS

When you use a DBMS data source, the observation ordering that is produced by PROC SORT depends on whether the DBMS or SAS performs the sorting. If you use the BEST value of the SAS system option SORTPGM=, then either the DBMS or SAS will perform the sort. If the DBMS performs the sort, then the configuration and characteristics of the DBMS sorting program will affect the resulting data order. 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 will be the same, run after run. To achieve consistency in the ordering of observations within BY groups, first populate a SAS data set with the DBMS data, then use the EQUALS or SORTEQUALS option to perform a stable sort.

Sorting Orders for Numeric Variables

For numeric variables, the smallest-to-largest comparison sequence is

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

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.

EBCDIC Order

The z/OS operating environment uses the EBCDIC collating sequence.

The sorting order of the English-language EBCDIC sequence is

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

- □ UNIX and its derivatives
- □ OpenVMS
- □ Windows.

From the smallest to the largest character that you can display, the English-language ASCII sequence is

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. For complete details, see the TRANTAB procedure in *SAS National Language Support (NLS): User's Guide*. 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.

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 the way that it is currently sorted, then PROC SORT does not perform the sort and writes a message to the log to that effect. To override this behavior, use the FORCE option. If you try to sort a data set the way that 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. For more information about SORTEDBY=, see the chapter on SAS data set options in *SAS Language Reference: Dictionary*.

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 348.

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 323.

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 may 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 1016. For more information about integrity constraints, see the chapter on SAS data files in *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.

To do this	Use this statement
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 data 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 about COMPRESS=, see the chapter on SAS data set options in *SAS Language Reference: Dictionary*.

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

Procedure features:

PROC SORT statement option:

OUT=

BY statement

Other features:

PROC PRINT

This example

- sorts the observations by the values of two variables
- creates an output data set for the sorted observations
- prints the results.

Program

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 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 print. The VAR statement specifies the variables to print 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                    1
                   Listed Alphabetically within Town

                                                                Account
      Obs    Company                     Town              Debt      Number

        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

Procedure features:

This example BY statement option:

DESCENDING

Other features

PROC PRINT

Data set: ACCOUNT on page 1017

- sorts the observations by the values of three variables
- sorts one of the variables in descending order
- prints the results.

Program

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 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 print. The VAR statement specifies the variables to print 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                    1
               Listed by Town, Amount, Account Number

                                                               Account
       Obs    Company                    Town              Debt     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
```

Example 3: Maintaining the Relative Order of Observations in Each BY Group

Procedure features:

PROC SORT statement option:

EQUALS | NOEQUALS

Other features: PROC PRINT

This example

- 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

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 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 print. The VAR statement specifies the variables to print 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 print. The VAR statement specifies the variables to print 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                              1

                           Years    Insurance
                   Obs    Worked        ID

                    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                        2

                           Years    Insurance
                   Obs    Worked        ID

                    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

Procedure features:

PROC SORT statement option:

NODUPKEY

BY statement

Other features:

PROC PRINT

Data set: ACCOUNT on page 1017

Interaction: 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.

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

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 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.

Operating Environment Information: 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 print. The VAR statement specifies the variables to print 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                  1

                                                                 Account
       Obs    Town              Company                     Debt      Number

        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

44

The SQL Procedure

Overview: SQL Procedure

What Is the SQL Procedure?

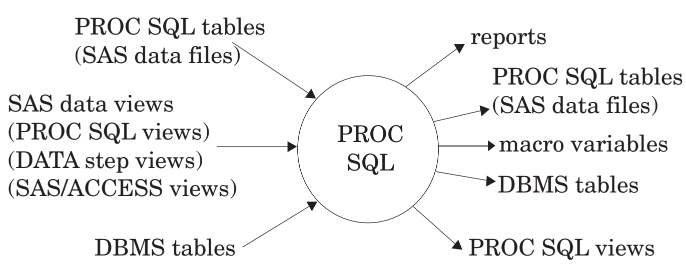
The SQL procedure implements Structured Query Language (SQL) for SAS. SQL is a standardized, widely used language that retrieves data from and updates data in tables and the views that are based on those tables.

The SAS SQL procedure enables you to

- retrieve and manipulate data that is stored in tables or views.
- create tables, views, and indexes on columns in tables.
- create SAS macro variables that contain values from rows in a query's result.
- add or modify the data values in a table's columns or insert and delete rows. You can also modify the table itself by adding, modifying, or dropping columns.
- send DBMS-specific SQL statements to a database management system (DBMS) and retrieve DBMS data.

The following figure summarizes the variety of source material that you can use with PROC SQL and what the procedure can produce.

Figure 44.1 PROC SQL Input and Output



What Are PROC SQL Tables?

A PROC SQL *table* is synonymous with a SAS data file and has a member type of DATA. You can use PROC SQL tables as input into DATA steps and procedures.

You create PROC SQL tables from SAS data files, from SAS data views, or from DBMS tables by using PROC SQL's Pass-Through Facility or the SAS/ACCESS LIBNAME statement. The Pass-Through Facility is described in "Connecting to a DBMS Using the SQL Procedure Pass-Through Facility" on page 1115. The SAS/ACCESS LIBNAME statement is described in "Connecting to a DBMS Using the LIBNAME Statement" on page 1115.

In PROC SQL terminology, a *row* in a table is the same as an *observation* in a SAS data file. A *column* is the same as a *variable*.

What Are Views?

A *SAS data view* defines a virtual data set that is named and stored for later use. A view contains no data but describes or defines data that is stored elsewhere. There are three types of SAS data views:

- PROC SQL views
- SAS/ACCESS views
- DATA step views.

You can refer to views in queries as if they were tables. The view derives its data from the tables or views that are listed in its FROM clause. The data that is accessed by a view is a subset or superset of the data that is in its underlying table(s) or view(s).

A PROC SQL view is a SAS data set of type VIEW that is created by PROC SQL. A PROC SQL view contains no data. It is a stored query expression that reads data values from its underlying files, which can include SAS data files, SAS/ACCESS views, DATA step views, other PROC SQL views, or DBMS data. When executed, a PROC SQL view's output can be a subset or superset of one or more underlying files.

SAS/ACCESS views and DATA step views are similar to PROC SQL views in that they are both stored programs of member type VIEW. SAS/ACCESS views describe data in DBMS tables from other software vendors. DATA step views are stored DATA step programs.

Note: Starting in SAS System 9, PROC SQL views, the Pass-Through Facility, and the SAS/ACCESS LIBNAME statement are the preferred ways to access relational DBMS data; SAS/ACCESS views are no longer recommended. You can convert existing SAS/ACCESS views to PROC SQL views by using the CV2VIEW procedure. See The CV2VIEW Procedure in *SAS/ACCESS for Relational Databases: Reference* for more information. △

You can update data through a PROC SQL or SAS/ACCESS view with certain restrictions. See "Updating PROC SQL and SAS/ACCESS Views" on page 1121.

You can use all types of views as input to DATA steps and procedures.

Note: In this chapter, the term *view* collectively refers to PROC SQL views, DATA step views, and SAS/ACCESS views, unless otherwise noted. △

Note: When the contents of an SQL view are processed (by a DATA step or a procedure), the referenced data set must be opened to retrieve information about the variables that is not stored in the view. If that data set has a libref associated with it that is not defined in the current SAS code, then an error will result. You can avoid this error by specifying a USING clause in the CREATE VIEW statement. See "CREATE VIEW Statement" on page 1049 for details. △

SQL Procedure Coding Conventions

Because PROC SQL implements Structured Query Language, it works somewhat differently from other base SAS procedures, as described here:

- When a PROC SQL statement is executed, PROC SQL continues to run until a QUIT statement, a DATA step, or another SAS procedure is executed. Therefore, you do not need to repeat the PROC SQL statement with each SQL statement. You need to repeat the PROC SQL statement only if you execute a QUIT statement, a DATA step, or another SAS procedure between SQL statements.
- SQL procedure statements are divided into clauses. For example, the most basic SELECT statement contains the SELECT and FROM clauses. Items within clauses are separated with commas in SQL, not with blanks as in other SAS code. For example, if you list three columns in the SELECT clause, then the columns are separated with commas.
- The SELECT statement, which is used to retrieve data, also automatically writes the output data to the Output window unless you specify the NOPRINT option in the PROC SQL statement. Therefore, you can display your output or send it to a list file without specifying the PRINT procedure.

□ The ORDER BY clause sorts data by columns. In addition, tables do not need to be presorted by a variable for use with PROC SQL. Therefore, you do not need to use the SORT procedure with your PROC SQL programs.

□ A PROC SQL statement runs when you submit it; you do not have to specify a RUN statement. If you follow a PROC SQL statement with a RUN statement, then SAS ignores the RUN statement and submits the statements as usual.

Syntax: SQL Procedure

Tip: Supports the Output Delivery System. See “Output Delivery System” on page 32 for details.

ODS Table Name: SQL_Results

Reminder: You can use any global statements. See Chapter 2, “Fundamental Concepts for Using Base SAS Procedures,” on page 15 for a list.

Reminder: You can use data set options any time a table name or view name is specified. See “Using SAS Data Set Options with PROC SQL” on page 1114 for details.

Note:

Regular type indicates the name of a component that is described in “SQL Procedure Component Dictionary” on page 1071.

view-name indicates a SAS data view of any type.

PROC SQL <*option(s)*>;

ALTER TABLE *table-name*
<**ADD** <CONSTRAINT> *constraint-clause*<, … *constraint-clause*>>
<**ADD** column-definition<, … column-definition>>
<**DROP CONSTRAINT** *constraint-name* <, … *constraint-name*>>
<**DROP** *column*<, … *column*>>
<**DROP FOREIGN KEY** *constraint-name*>
<**DROP PRIMARY KEY**>
<**MODIFY** column-definition<, … column-definition>>
;

CREATE <**UNIQUE**> **INDEX** *index-name*
ON *table-name* (*column* <, … *column*>);

CREATE TABLE *table-name*
(*column-specification*<, …*column-specification* | *constraint-specification*>)
;

CREATE TABLE *table-name* **LIKE** *table-name2*;

CREATE TABLE *table-name* **AS** query-expression
<**ORDER BY** *order-by-item*<, … *order-by-item*>>;

CREATE VIEW *proc-sql-view* **AS** query-expression
<**ORDER BY** *order-by-item*<, … *order-by-item*>>
<**USING** *libname-clause*<, … *libname-clause*>> ;

DELETE
FROM *table-name*|*proc-sql-view* |*sas/access-view* <**AS** *alias*>
<**WHERE** sql-expression>;

DESCRIBE TABLE*table-name* <, … *table-name*>;

DESCRIBE VIEW *proc-sql-view* <, … *proc-sql-view*>;

DESCRIBE TABLE CONSTRAINTS *table-name* <, … *table-name*>;

DROP INDEX *index-name* <, … *index-name*>
FROM *table-name*;

DROP TABLE *table-name* <, … *table-name*>;

DROP VIEW *view-name* <, … *view-name*>;

INSERT INTO *table-name* | *sas/access-view* | *proc-sql-view* <(*column*<, … *column*>)>
SET *column*=sql-expression
<, … *column*=sql-expression>
<**SET** *column*=sql-expression
<, … *column*=sql-expression>>;

INSERT INTO *table-name* | *sas/access-view* | *proc-sql-view* <(*column*<, … *column*>)>
VALUES (*value* <, … *value*>)
<… **VALUES** (*value* <, … *value*>)>;

INSERT INTO *table-name* | *sas/access-view* | *proc-sql-view*
<(*column*<, …*column*>)> query-expression;

RESET <*option(s)*>;

SELECT <DISTINCT> *object-item* <, …*object-item*>
<**INTO** *macro-variable-specification*
<, … *macro-variable-specification*>>
FROM *from-list*
<**WHERE** sql-expression>
<**GROUP BY** *group-by-item*
<, … *group-by-item*>>
<**HAVING** sql-expression>
<**ORDER BY** *order-by-item*
<, … *order-by-item*>>;

UPDATE *table-name* | *sas/access-view* | *proc-sql-view* <**AS** *alias*>
SET *column*=sql-expression
<, … *column*=sql-expression>
<**SET** *column*=sql-expression
<, … *column*=sql-expression>>
<**WHERE** sql-expression>;

VALIDATE query-expression;

To connect to a DBMS and send it a DBMS-specific nonquery SQL statement, use this form:

PROC SQL;
CONNECT TO *dbms-name* <AS *alias*>
<(*connect-statement-argument-1=value* <…
connect-statement-argument-n=value>)>
<(*database-connection-argument-1=value* <…
database-connection-argument-n=value>)>;
EXECUTE (*dbms-SQL-statement*)
BY *dbms-name* | *alias*;
<**DISCONNECT FROM** *dbms-name* | *alias*;>

<**QUIT**;>

To connect to a DBMS and query the DBMS data, use this form:

PROC SQL;
  CONNECT TO *dbms-name* <AS *alias*>
      <(*connect-statement-argument-1=value* <…
      connect-statement-argument-n=value>)>
      <(*database-connection-argument-1=value* <…
      database-connection-argument-n=value>)>;
  SELECT *column-list*
    FROM CONNECTION TO *dbms-name*|*alias*
        (*dbms-query*)
        optional PROC SQL clauses;
  <**DISCONNECT FROM** *dbms-name*|*alias*;>
<**QUIT**;>

To do this	Use this statement
Modify, add, or drop columns	ALTER TABLE
Establish a connection with a DBMS	CONNECT TO
Create an index on a column	CREATE INDEX
Create a PROC SQL table	CREATE TABLE
Create a PROC SQL view	CREATE VIEW
Delete rows	DELETE
Display a definition of a table or view	DESCRIBE
Terminate the connection with a DBMS	DISCONNECT FROM
Delete tables, views, or indexes	DROP
Send a DBMS-specific nonquery SQL statement to a DBMS	EXECUTE
Add rows	INSERT
Reset options that affect the procedure environment without restarting the procedure	RESET
Select and execute rows	SELECT
Query a DBMS	CONNECTION TO
Modify values	UPDATE
Verify the accuracy of your query	VALIDATE

PROC SQL Statement

PROC SQL <*option(s)*>;

To do this	Use this option
Control output	
Double-space the report	DOUBLE | NODOUBLE
Write a statement to the SAS log that expands the query	FEEDBACK | NOFEEDBACK
Flow characters within a column	FLOW | NOFLOW
Include a column of row numbers	NUMBER | NONUMBER
Specify whether PROC SQL prints the query's result	PRINT | NOPRINT
Specify whether PROC SQL should display sorting information	SORTMSG | NOSORTMSG
Specify a collating sequence	SORTSEQ=
Control execution	
Allow PROC SQL to use names other than SAS names	DQUOTE=
Specify whether PROC SQL should stop executing after an error	ERRORSTOP | NOERRORSTOP
Specify whether PROC SQL should execute statements	EXEC | NOEXEC
Restrict the number of input rows	INOBS=
Restrict the number of output rows	OUTOBS=
Restrict the number of loops	LOOPS=
Specify whether PROC SQL prompts you when a limit is reached with the INOBS=, OUTOBS=, or LOOPS= options	PROMPT | NOPROMPT
Specify whether PROC SQL writes timing information for each statement to the SAS log	STIMER | NOSTIMER
Override the SAS system option THREADS | NOTHREADS	THREADS | NOTHREADS
Specify how PROC SQL handles updates when there is an interruption	UNDO_POLICY=

Options

DOUBLE | NODOUBLE
double-spaces the report.

Default: NODOUBLE

Featured in: Example 5 on page 1134

DQUOTE=ANSI | SAS
specifies whether PROC SQL treats values within double quotation marks (" ") as variables or strings. With DQUOTE=ANSI, PROC SQL treats a quoted value as a

variable. This feature enables you to use the following as table names, column names, or aliases:

- □ reserved words such as AS, JOIN, GROUP, and so on
- □ DBMS names and other names that are not normally permissible in SAS.

The quoted value can contain any character.

With DQUOTE=SAS, values within double quotation marks are treated as strings.

Default: SAS

ERRORSTOP|NOERRORSTOP

specifies whether PROC SQL stops executing if it encounters an error. In a batch or noninteractive session, ERRORSTOP instructs PROC SQL to stop executing the statements but to continue checking the syntax after it has encountered an error.

NOERRORSTOP instructs PROC SQL to execute the statements and to continue checking the syntax after an error occurs.

Default: NOERRORSTOP in an interactive SAS session; ERRORSTOP in a batch or noninteractive session

Interaction: This option is useful only when the EXEC option is in effect.

Tip: ERRORSTOP has an effect only when SAS is running in the batch or noninteractive execution mode.

Tip: NOERRORSTOP is useful if you want a batch job to continue executing SQL procedure statements after an error is encountered.

EXEC|NOEXEC

specifies whether a statement should be executed after its syntax is checked for accuracy.

Default: EXEC

Tip: NOEXEC is useful if you want to check the syntax of your SQL statements without executing the statements.

See also: ERRORSTOP on page 1035

FEEDBACK|NOFEEDBACK

specifies whether PROC SQL displays, in the SAS log, PROC SQL statements after view references are expanded or certain other transformations of the statement are made.

This option has the following effects:

- □ Any asterisk (for example, `SELECT *`) is expanded into the list of qualified columns that it represents.
- □ Any PROC SQL view is expanded into the underlying query.
- □ Macro variables are resolved.
- □ Parentheses are shown around all expressions to further indicate their order of evaluation.
- □ Comments are removed.

Default: NOFEEDBACK

FLOW<=*n* <*m*>>|NOFLOW

specifies that character columns longer than *n* are flowed to multiple lines. PROC SQL sets the column width at *n* and specifies that character columns longer than *n* are flowed to multiple lines. When you specify FLOW=*n m*, PROC SQL floats the width of the columns between these limits to achieve a balanced layout. Specifying FLOW without arguments is equivalent to specifying FLOW=12 200.

Default: NOFLOW

INOBS=*n*
restricts the number of rows (observations) that PROC SQL retrieves from any single source.

Tip: This option is useful for debugging queries on large tables.

LOOPS=*n*
restricts PROC SQL to *n* iterations through its inner loop. You use the number of iterations reported in the SQLOOPS macro variable (after each SQL statement is executed) to discover the number of loops. Set a limit to prevent queries from consuming excessive computer resources. For example, joining three large tables without meeting the join-matching conditions could create a huge internal table that would be inefficient to execute.

See also: "Using Macro Variables Set by PROC SQL" on page 1119

NODOUBLE
See DOUBLE | NODOUBLE on page 1034.

NOERRORSTOP
See ERRORSTOP | NOERRORSTOP on page 1035.

NOEXEC
See EXEC | NOEXEC on page 1035.

NOFEEDBACK
See FEEDBACK | NOFEEDBACK on page 1035.

NOFLOW
See FLOW | NOFLOW on page 1035.

NONUMBER
See NUMBER | NONUMBER on page 1036.

NOPRINT
See PRINT | NOPRINT on page 1036.

NOPROMPT
See PROMPT | NOPROMPT on page 1037.

NOSORTMSG
See SORTMSG | NOSORTMSG on page 1037.

NOSTIMER
See STIMER | NOSTIMER on page 1037.

NOTHREADS
See THREADS | NOTHREADS.

NUMBER | NONUMBER
specifies whether the SELECT statement includes a column called ROW, which is the row (or observation) number of the data as the rows are retrieved.

Default: NONUMBER

Featured in: Example 4 on page 1131

OUTOBS=*n*
restricts the number of rows (observations) in the output. For example, if you specify OUTOBS=10 and insert values into a table using a query-expression, then the SQL procedure inserts a maximum of 10 rows. Likewise, OUTOBS=10 limits the output to 10 rows.

PRINT | NOPRINT
specifies whether the output from a SELECT statement is printed.

Default: PRINT

Tip: NOPRINT is useful when you are selecting values from a table into macro variables and do not want anything to be displayed.

Interaction: NOPRINT affects the value of the SQLOBS automatic macro variable. See "Using Macro Variables Set by PROC SQL" on page 1119 for details.

PROMPT | NOPROMPT
modifies the effect of the INOBS=, OUTOBS=, and LOOPS= options. If you specify the PROMPT option and reach the limit specified by INOBS=, OUTOBS=, or LOOPS=, then PROC SQL prompts you to stop or continue. The prompting repeats if the same limit is reached again.

Default: NOPROMPT

SORTMSG | NOSORTMSG
Certain operations, such as ORDER BY, may sort tables internally using PROC SORT. Specifying SORTMSG requests information from PROC SORT about the sort and displays the information in the log.

Default: NOSORTMSG

SORTSEQ=*sort-table*
specifies the collating sequence to use when a query contains an ORDER BY clause. Use this option only if you want a collating sequence other than your system's or installation's default collating sequence.

See also: SORTSEQ= option in *SAS National Language Support (NLS): User's Guide*.

STIMER | NOSTIMER
specifies whether PROC SQL writes timing information to the SAS log for each statement, rather than as a cumulative value for the entire procedure. For this option to work, you must also specify the SAS system option STIMER. Some operating environments require that you specify this system option when you invoke SAS. If you use the system option alone, then you receive timing information for the entire SQL procedure, not on a statement-by-statement basis.

Default: NOSTIMER

THREADS | NOTHREADS
overrides the SAS system option THREADS | NOTHREADS for a particular invocation of PROC SQL. THREADS | NOTHREADS can also be specified in a RESET statement for use in particular queries. When THREADS is specified, PROC SQL uses parallel processing in order to increase the performance of sorting operations that involve large amounts of data. For more information about parallel processing, see *SAS Language Reference: Concepts*.

Default: value of SAS system option THREADS | NOTHREADS.

Note: When THREADS | NOTHREADS has been specified in a PROC SQL statement or a RESET statement, there is no way to reset the option to its default (that is, the value of the SAS system option THREADS | NOTHREADS) for that invocation of PROC SQL. △

UNDO_POLICY=NONE | OPTIONAL | REQUIRED
specifies how PROC SQL handles updated data if errors occur while you are updating data. You can use UNDO_POLICY= to control whether your changes will be permanent:

NONE
keeps any updates or inserts.

OPTIONAL
: reverses any updates or inserts that it can reverse reliably.

REQUIRED
: reverses all inserts or updates that have been done to the point of the error. In some cases, the UNDO operation cannot be done reliably. For example, when a program uses a SAS/ACCESS view, it may not be able to reverse the effects of the INSERT and UPDATE statements without reversing the effects of other changes at the same time. In that case, PROC SQL issues an error message and does not execute the statement. Also, when a SAS data set is accessed through a SAS/SHARE server and is opened with the data set option CNTLLEV=RECORD, you cannot reliably reverse your changes.

  This option may enable other users to update newly inserted rows. If an error occurs during the insert, then PROC SQL can delete a record that another user updated. In that case, the statement is not executed, and an error message is issued.

Default: REQUIRED

Note: Options can be added, removed, or changed between PROC SQL statements with the RESET statement. △

ALTER TABLE Statement

Adds columns to, drops columns from, and changes column attributes in an existing table. Adds, modifies, and drops integrity constraints from an existing table.

Restriction: You cannot use any type of view in an ALTER TABLE statement.

Restriction: You cannot use ALTER TABLE on a table that is accessed by an engine that does not support UPDATE processing.

Restriction: You must use at least one ADD, DROP, or MODIFY clause in the ALTER TABLE statement.

Featured in: Example 3 on page 1129

ALTER TABLE *table-name*

<**ADD** CONSTRAINT *constraint-name constraint-clause*<, … *constraint-name constraint-clause*>>

<**ADD** *constraint-specification*<, … *constraint-specification*>>

<**ADD** column-definition<, … column-definition>>

<**DROP CONSTRAINT** *constraint-name* <, … *constraint-name*>>

<**DROP** *column*<, … *column*>>

<**DROP FOREIGN KEY** *constraint-name*>

<**DROP PRIMARY KEY**>

<**MODIFY** column-definition<, … column-definition>>

;

Arguments

<**ADD** CONSTRAINT *constraint-name constraint-specification*<, … *constraint-name constraint-specification*>>
adds the integrity constraint that is specified in *constraint-specification* and assigns *constraint-name* to it.

<**ADD** *constraint-specification*<, … *constraint-specification*>>
adds the integrity constraint that is specified in *constraint-specification* and assigns a default name to it. The default constraint name has the form that is shown in the following table:

Default Name	Constraint Type
NM*xxxx*	Not null
UN*xxxx*	Unique
CK*xxxx*	Check
PK*xxxx*	Primary key
FK*xxxx*	Foreign key

In these default names, *xxxx* is a counter that begins at 0001.

<**ADD** column-definition<, … column-definition>>
adds the column(s) that are specified in each column-definition.

column
names a column in *table-name*.

column-definition
See "column-definition" on page 1075.

constraint
is one of the following integrity constraints:

CHECK (*WHERE-clause*)
specifies that all rows in *table-name* satisfy the *WHERE-clause*.

DISTINCT (*column*<, … *column*>)
specifies that the values of each *column* must be unique. This constraint is identical to UNIQUE.

FOREIGN KEY (*column*<, … *column*>)
REFERENCES *table-name*
<ON DELETE *referential-action* > <ON UPDATE *referential-action*>
specifies a foreign key, that is, a set of *columns* whose values are linked to the values of the primary key variable in another table (the *table-name* that is specified for REFERENCES). The *referential-actions* are performed when the values of a primary key column that is referenced by the foreign key are updated or deleted.

Restriction: 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 the exact 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.

NOT NULL (*column*)
specifies that *column* does not contain a null or missing value, including special missing values.

PRIMARY KEY (*column*<, ... *column*>)
specifies one or more primary key columns, that is, columns that do not contain missing values and whose values are unique.

Restriction: When you are defining overlapping primary key and foreign key constraints, which means that variables in a data file are part of both a primary key definition and a foreign key definition, if you use the exact same variables, then the variables must be defined in a different order.

UNIQUE (*column*<, ... *column*>)
specifies that the values of each *column* must be unique. This constraint is identical to DISTINCT.

constraint-name
specifies a name for the constraint that is being specified. The name must be a valid SAS name.

Note: The names PRIMARY, FOREIGN, MESSAGE, UNIQUE, DISTINCT, CHECK, and NOT cannot be used as values for *constraint-name*. △

constraint-specification
consists of

constraint <MESSAGE='*message-string*' <MSGTYPE=*message-type*>>

<**DROP** *column*<, ... *column*>>
deletes each *column* from the table.

<**DROP CONSTRAINT** *constraint-name* <, ...*constraint-name*>>
deletes the integrity constraint that is referenced by each *constraint-name*. To find the name of an integrity constraint, use the DESCRIBE TABLE CONSTRAINTS clause (see “DESCRIBE Statement” on page 1052).

<**DROP FOREIGN KEY** *constraint-name*>
Removes the foreign key constraint that is referenced by *constraint-name*.

Note: The DROP FOREIGN KEY clause is a DB2 extension. △

<**DROP PRIMARY KEY**>
Removes the primary key constraint from *table-name*.

Note: The DROP PRIMARY KEY clause is a DB2 extension. △

message-string
specifies the text of an error message that is written to the log when the integrity constraint is not met. The maximum length of *message-string* is 250 characters.

message-type
specifies how the error message is displayed in the SAS log when an integrity constraint is not met.

NEWLINE
: the text that is specified for MESSAGE= is displayed as well as the default error message for that integrity constraint.

USER
: only the text that is specified for MESSAGE= is displayed.

<**MODIFY** column-definition<, … column-definition>>
: changes one or more attributes of the column that is specified in each column-definition.

referential-action
: specifies the type of action to be performed on all matching foreign key values.

CASCADE
: allows primary key data values to be updated, and updates matching values in the foreign key to the same values. This referential action is currently supported for updates only.

RESTRICT
: prevents the update or deletion of primary key data values if a matching value exists in the foreign key. This referential action is the default.

SET NULL
: allows primary key data values to be updated, and sets all matching foreign key values to NULL.

table-name

- □ in the ALTER TABLE statement, refers to the name of the table that is to be altered.
- □ in the REFERENCES clause, refers to the name of table that contains the primary key that is referenced by the foreign key.

table-name can be a one-level name, a two-level *libref.table* name, or a physical pathname that is enclosed in single quotation marks.

WHERE-clause
: specifies a SAS WHERE clause. Do not include the WHERE keyword in the WHERE clause.

Specifying Initial Values of New Columns

When the ALTER TABLE statement adds a column to the table, it initializes the column's values to missing in all rows of the table. Use the UPDATE statement to add values to the new column(s).

Changing Column Attributes

If a column is already in the table, then you can change the following column attributes by using the MODIFY clause: length, informat, format, and label. The values in a table are either truncated or padded with blanks (if character data) as necessary to meet the specified length attribute.

You cannot change a character column to numeric and vice versa. To change a column's data type, drop the column and then add it (and its data) again, or use the DATA step.

Note: You cannot change the length of a numeric column with the ALTER TABLE statement. Use the DATA step instead. △

Renaming Columns

To change a column's name, you must use the SAS data set option RENAME=. You cannot change this attribute with the ALTER TABLE statement. RENAME= is described in the section on SAS data set options in *SAS Language Reference: Dictionary*.

Indexes on Altered Columns

When you alter the attributes of a column and an index has been defined for that column, the values in the altered column continue to have the index defined for them. If you drop a column with the ALTER TABLE statement, then all the indexes (simple and composite) in which the column participates are also dropped. See "CREATE INDEX Statement" on page 1043 for more information about creating and using indexes.

Integrity Constraints

Use ALTER TABLE to modify integrity constraints for existing tables. Use the CREATE TABLE statement to attach integrity constraints to new tables. For more information on integrity constraints, see the section on SAS files in *SAS Language Reference: Concepts*.

CONNECT Statement

Establishes a connection with a DBMS that is supported by SAS/ACCESS software.

Requirement: SAS/ACCESS software is required. For more information about this statement, refer to your SAS/ACCESS documentation.

See also: "Connecting to a DBMS Using the SQL Procedure Pass-Through Facility" on page 1115

CONNECT TO *dbms-name* <AS *alias*>
<(*connect-statement-argument-1=value* <...
connect-statement-argument-n=value>)>
<(*database-connection-argument-1=value* <...
database-connection-argument-n=value>)>;

Arguments

alias
specifies an alias that has 1 to 32 characters. The keyword AS must precede *alias*. Some DBMSs allow more than one connection. The optional AS clause enables you to name the connections so that you can refer to them later.

connect-statement-argument=value
specifies values for arguments that indicate whether you can make multiple connections, shared or unique connections, and so on, to the database. These arguments are optional, but if they are included, then they must be enclosed in parentheses. See *SAS/ACCESS for Relational Databases: Reference* for more information about these arguments.

database-connection-argument=value
specifies values for the DBMS-specific arguments that are needed by PROC SQL in order to connect to the DBMS. These arguments are optional for most databases, but if they are included, then they must be enclosed in parentheses. For more information, see the SAS/ACCESS documentation for your DBMS.

dbms-name
identifies the DBMS that you want to connect to (for example, ORACLE or DB2).

CREATE INDEX Statement

Creates indexes on columns in tables.

Restriction: You cannot use CREATE INDEX on a table that is accessed with an engine that does not support UPDATE processing.

CREATE <**UNIQUE**> **INDEX** *index-name*
ON *table-name* (*column* <, … *column*>);

Arguments

column
specifies a column in *table-name*.

index-name
names the index that you are creating. If you are creating an index on one column only, then *index-name* must be the same as *column*. If you are creating an index on more than one column, then *index-name* cannot be the same as any column in the table.

table-name
specifies a PROC SQL table.

Indexes in PROC SQL

An *index* stores both the values of a table's columns and a system of directions that enable access to rows in that table by index value. Defining an index on a column or set of columns enables SAS, under certain circumstances, to locate rows in a table more quickly and efficiently. Indexes enable PROC SQL to execute the following classes of queries more efficiently:

- comparisons against a column that is indexed
- an IN subquery where the column in the inner subquery is indexed
- correlated subqueries, where the column being compared with the correlated reference is indexed
- join-queries, where the join-expression is an equals comparison and all the columns in the join-expression are indexed in one of the tables being joined.

SAS maintains indexes for all changes to the table, whether the changes originate from PROC SQL or from some other source. Therefore, if you alter a column's definition or update its values, then the same index continues to be defined for it. However, if an indexed column in a table is dropped, then the index on it is also dropped.

You can create simple or composite indexes. A *simple index* is created on one column in a table. A simple index must have the same name as that column. A *composite index* is one index name that is defined for two or more columns. The columns can be specified in any order, and they can have different data types. A composite index name cannot match the name of any column in the table. If you drop a composite index, then the index is dropped for all the columns named in that composite index.

UNIQUE Keyword

The UNIQUE keyword causes SAS to reject any change to a table that would cause more than one row to have the same index value. Unique indexes guarantee that data in one column, or in a composite group of columns, remain unique for every row in a table. For this reason, a unique index cannot be defined for a column that includes NULL or missing values.

Managing Indexes

You can use the CONTENTS statement in the DATASETS procedure to display a table's index names and the columns for which they are defined. You can also use the DICTIONARY tables INDEXES, TABLES, and COLUMNS to list information about indexes. For more information, see "Using the DICTIONARY Tables" on page 1116.

See the section on SAS files in *SAS Language Reference: Dictionary* for a further description of when to use indexes and how they affect SAS statements that handle BY-group processing.

CREATE TABLE Statement

Creates PROC SQL tables.

Featured in: Example 1 on page 1125 and Example 2 on page 1127

❶ **CREATE TABLE** *table-name*
(*column-specification*<, …*column-specification* | *constraint-specification*>)
;

❷ **CREATE TABLE** *table-name* **LIKE** *table-name2*;

❸ **CREATE TABLE** *table-name* **AS** query-expression
<**ORDER BY** *order-by-item*<, … *order-by-item*>>;

Arguments

column-constraint
is one of the following:

CHECK (*WHERE-clause*)
specifies that all rows in *table-name* satisfy the *WHERE-clause*.

DISTINCT
specifies that the values of the column must be unique. This constraint is identical to UNIQUE.

NOT NULL
specifies that the column does not contain a null or missing value, including special missing values.

PRIMARY KEY
specifies that the column is a primary key column, that is, a column that does not contain missing values and whose values are unique.

Restriction: 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 the exact same variables, then the variables must be defined in a different order.

REFERENCES *table-name*
<**ON DELETE** *referential-action* > <**ON UPDATE** *referential-action*>
specifies that the column is a foreign key, that is, a column whose values are linked to the values of the primary key variable in another table (the *table-name* that is specified for REFERENCES). The *referential-actions* are performed when the values of a primary key column that is referenced by the foreign key are updated or deleted.

Restriction: When you are defining overlapping primary key and foreign key constraints, which means that variables in a data file are part of both a primary key definition and a foreign key definition,

- if you use the exact 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.

UNIQUE
specifies that the values of the column must be unique. This constraint is identical to DISTINCT.

Note: If you specify *column-constraint*, then SAS automatically assigns a name to the constraint. The constraint name has the form

Default name	Constraint type
CK*xxxx*	Check
FK*xxxx*	Foreign key
NM*xxxx*	Not Null
PK*xxxx*	Primary key
UN*xxxx*	Unique

where *xxxx* is a counter that begins at 0001. △

column-definition
See "column-definition" on page 1075.

column-specification
consists of

column-definition <*column-constraint*>

constraint
is one of the following:

CHECK (*WHERE-clause*)
specifies that all rows in *table-name* satisfy the *WHERE-clause*.

DISTINCT (*column*<, … *column*>)
specifies that the values of each *column* must be unique. This constraint is identical to UNIQUE.

FOREIGN KEY (*column*<, … *column*>)
REFERENCES *table-name*
<ON DELETE *referential-action* > <ON UPDATE *referential-action*>
specifies a foreign key, that is, a set of *columns* whose values are linked to the values of the primary key variable in another table (the *table-name* that is specified for REFERENCES). The *referential-actions* are performed when the values of a primary key column that is referenced by the foreign key are updated or deleted.

Restriction: When you are defining overlapping primary key and foreign key constraints, which means that variables in a data file are part of both a primary key definition and a foreign key definition,

- if you use the exact 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.

NOT NULL (*column*)
specifies that *column* does not contain a null or missing value, including special missing values.

PRIMARY KEY (*column*<, … *column*>)
specifies one or more primary key columns, that is, columns that do not contain missing values and whose values are unique.

Restriction: When defining overlapping primary key and foreign ke constraints, which means that variables in a data file are part of both a primary key and a foreign key definition, if you use the exact same variables, then the variables must be defined in a different order.

UNIQUE (*column*<, …*column*>)
specifies that the values of each *column* must be unique. This constraint is identical to DISTINCT.

constraint-name
specifies a name for the constraint that is being specified. The name must be a valid SAS name.

Note: The names PRIMARY, FOREIGN, MESSAGE, UNIQUE, DISTINCT, CHECK, and NOT cannot be used as values for *constraint-name*. △

constraint-specification
consists of

CONSTRAINT *constraint-name constraint* <MESSAGE='*message-string*'
<MSGTYPE=*message-type*>>

message-string
specifies the text of an error message that is written to the log when the integrity constraint is not met. The maximum length of *message-string* is 250 characters.

message-type
specifies how the error message is displayed in the SAS log when an integrity constraint is not met.

NEWLINE
the text that is specified for MESSAGE= is displayed as well as the default error message for that integrity constraint.

USER
only the text that is specified for MESSAGE= is displayed.

ORDER BY *order-by-item*
sorts the rows in *table-name* by the values of each *order-by-item*. See ORDER BY Clause on page 1067.

query-expression
creates *table-name* from the results of a query. See "query-expression" on page 1093.

referential-action
specifies the type of action to be performed on all matching foreign key values.

CASCADE
allows primary key data values to be updated, and updates matching values in the foreign key to the same values. This referential action is currently supported for updates only.

RESTRICT
occurs only if there are matching foreign key values. This referential action is the default.

SET NULL
sets all matching foreign key values to NULL.

table-name

- in the CREATE TABLE statement, refers to the name of the table that is to be created. You can use data set options by placing them in parentheses immediately after *table-name*. See "Using SAS Data Set Options with PROC SQL" on page 1114 for details.
- in the REFERENCES clause, refers to the name of table that contains the primary key that is referenced by the foreign key.

table-name2
creates *table-name* with the same column names and column attributes as *table-name2*, but with no rows.

WHERE-clause
specifies a SAS WHERE clause. Do not include the WHERE keyword in the WHERE clause.

Creating a Table without Rows

❶ The first form of the CREATE TABLE statement creates tables that automatically map SQL data types to those that are supported by SAS. Use this form when you want to create a new table with columns that are not present in existing tables. It is also useful if you are running SQL statements from an SQL application in another SQL-based database.

❷ The second form uses a LIKE clause to create a table that has the same column names and column attributes as another table. To drop any columns in the new table, you can specify the DROP= data set option in the CREATE TABLE statement. The specified columns are dropped when the table is created. Indexes are not copied to the new table.

Both of these forms create a table without rows. You can use an INSERT statement to add rows. Use an ALTER TABLE statement to modify column attributes or to add or drop columns.

Creating a Table from a Query Expression

❸ The third form of the CREATE TABLE statement stores the results of any query-expression in a table and does not display the output. It is a convenient way to create temporary tables that are subsets or supersets of other tables.

When you use this form, a table is physically created as the statement is executed. The newly created table does not reflect subsequent changes in the underlying tables (in the query-expression). If you want to continually access the most current data, then create a view from the query expression instead of a table. See “CREATE VIEW Statement” on page 1049.

CAUTION:

Recursive table references can cause data integrity problems. While it is possible to recursively reference the target table of a CREATE TABLE AS statement, doing so can cause data integrity problems and incorrect results. Constructions such as the following should be avoided:

```
proc sql;
   create table a as
      select var1, var2
      from a;
```

△

Integrity Constraints

You can attach integrity constraints when you create a new table. To modify integrity constraints, use the ALTER TABLE statement. For more information on integrity constraints, see the section on SAS files in *SAS Language Reference: Concepts*.

CREATE VIEW Statement

Creates a PROC SQL view from a query-expression.

See also: “What Are Views?” on page 1029

Featured in: Example 8 on page 1143

CREATE VIEW *proc-sql-view* <(*column-name-list*)> **AS** query-expression

<**ORDER BY** *order-by-item*<, … *order-by-item*>>

<**USING** *libname-clause*<, … *libname-clause*>> ;

Arguments

column-name-list
is a comma-separated list of column names for the view, to be used in place of the column names or aliases that are specified in the SELECT clause. The names in this list are assigned to columns in the order in which they are specified in the SELECT clause. If the number of column names in this list does not equal the number of columns in the SELECT clause, then a warning is written to the SAS log.

query-expression
See “query-expression” on page 1093.

libname-clause
is one of the following:

LIBNAME *libref <engine> 'SAS-data-library' <option(s)> <engine-host-option(s)>*

LIBNAME *libref SAS/ACCESS-engine-name*
<SAS/ACCESS-engine-connection-option(s)>
<SAS/ACCESS-engine-LIBNAME-option(s)>

See *SAS Language Reference: Dictionary* for information about the base SAS LIBNAME statement. See *SAS/ACCESS for Relational Databases: Reference* for information about the LIBNAME statement for relational databases.

order-by-item
See ORDER BY Clause on page 1067.

proc-sql-view
specifies the name for the PROC SQL view that you are creating. See “What Are Views?” on page 1029 for a definition of a PROC SQL view.

Sorting Data Retrieved by Views

PROC SQL enables you to specify the ORDER BY clause in the CREATE VIEW statement. When a view with an ORDER BY clause is accessed, and the ORDER BY clause directly affects the order of the results, its data is sorted and displayed as specified by the ORDER BY clause. However, if the ORDER BY clause does not directly affect the order of the results (for instance, if the view is specified as part of a join), then PROC SQL ignores the ORDER BY clause in order to enhance performance.

Note: If you specify the NUMBER option in the PROC SQL statement when you create your view, then the ROW column appears in the output. However, you cannot order by the ROW column in subsequent queries. See the description of NUMBER|NONUMBER on page 1036. △

Librefs and Stored Views

You can refer to a table name alone (without the libref) in the FROM clause of a CREATE VIEW statement if the table and view reside in the same SAS data library, as in this example:

```
create view proclib.view1 as
   select *
      from invoice
      where invqty>10;
```

In this view, VIEW1 and INVOICE are stored permanently in the SAS data library referenced by PROCLIB. Specifying a libref for INVOICE is optional.

Updating Views

You can update a view's underlying data with some restrictions. See "Updating PROC SQL and SAS/ACCESS Views" on page 1121.

Embedded LIBNAME Statements

The USING clause enables you to store DBMS connection information in a view by *embedding* the SAS/ACCESS LIBNAME statement inside the view. When PROC SQL executes the view, the stored query assigns the libref and establishes the DBMS connection using the information in the LIBNAME statement. The scope of the libref is local to the view, and will not conflict with any identically named librefs in the SAS session. When the query finishes, the connection to the DBMS is terminated and the libref is deassigned.

The USING clause must be the last clause in the CREATE VIEW statement. Multiple LIBNAME statements can be specified, separated by commas. In the following example, a connection is made and the libref ACCREC is assigned to an ORACLE database.

```
create view proclib.view1 as
   select *
      from accrec.invoices as invoices
      using libname accrec oracle
         user=username pass=password
         path='dbms-path';
```

For more information on the SAS/ACCESS LIBNAME statement, see the SAS/ACCESS documentation for your DBMS.

Note: Starting in SAS System 9, PROC SQL views, the Pass-Through Facility, and the SAS/ACCESS LIBNAME statement are the preferred ways to access relational DBMS data; SAS/ACCESS views are no longer recommended. You can convert existing SAS/ACCESS views to PROC SQL views by using the CV2VIEW procedure. See "The CV2VIEW Procedure" in *SAS/ACCESS for Relational Databases: Reference* for more information. △

You can also embed a SAS LIBNAME statement in a view with the USING clause. This enables you to store SAS libref information in the view. Just as in the embedded SAS/ACCESS LIBNAME statement, the scope of the libref is local to the view, and it will not conflict with an identically named libref in the SAS session.

```
create view work.tableview as
   select * from proclib.invoices
      using libname proclib 'sas-data-library';
```

DELETE Statement

Removes one or more rows from a table or view that is specified in the FROM clause.

Restriction: You cannot use DELETE FROM on a table that is accessed by an engine that does not support UPDATE processing.

Featured in: Example 5 on page 1134

DELETE
FROM *table-name* | *sas/access-view* | *proc-sql-view* <**AS** *alias*>
<**WHERE** sql-expression>;

Arguments

alias
assigns an alias to *table-name*, *sas/access-view*, or *proc-sql-view*.

sas/access-view
specifies a SAS/ACCESS view that you are deleting rows from.

proc-sql-view
specifies a PROC SQL view that you are deleting rows from. *proc-sql-view* can be a one-level name, a two-level *libref.view* name, or a physical pathname that is enclosed in single quotation marks.

sql-expression
See "sql-expression" on page 1099.

table-name
specifies the table that you are deleting rows from. *table-name* can be a one-level name, a two-level *libref.table* name, or a physical pathname that is enclosed in single quotation marks.

CAUTION:
Recursive table references can cause data integrity problems. While it is possible to recursively reference the target table of a DELETE statement, doing so can cause data integrity problems and incorrect results. Constructions such as the following should be avoided:

```
proc sql;
   delete from a
      where var1 > (select min(var2) from a);
```

△

Deleting Rows through Views

You can delete one or more rows from a view's underlying table, with some restrictions. See "Updating PROC SQL and SAS/ACCESS Views" on page 1121.

CAUTION:
If you omit a WHERE clause, then the DELETE statement deletes all the rows from the specified table or the table that is described by a view. △

DESCRIBE Statement

Displays a PROC SQL definition in the SAS log.

Restriction: PROC SQL views are the only type of view allowed in a DESCRIBE VIEW statement.

Featured in: Example 6 on page 1136

DESCRIBE TABLE *table-name* <, … *table-name*>;

DESCRIBE VIEW *proc-sql-view* <, … *proc-sql-view*>;

DESCRIBE TABLE CONSTRAINTS *table-name* <, … *table-name*>;

Arguments

table-name
specifies a PROC SQL table. *table-name* can be a one-level name, a two-level *libref.table* name, or a physical pathname that is enclosed in single quotation marks.

proc-sql-view
specifies a PROC SQL view. *proc-sql-view* can be a one-level name, a two-level *libref.view* name, or a physical pathname that is enclosed in single quotation marks.

Details

- The DESCRIBE TABLE statement writes a CREATE TABLE statement to the SAS log for the table specified in the DESCRIBE TABLE statement, regardless of how the table was originally created (for example, with a DATA step). If applicable, SAS data set options are included with the table definition. If indexes are defined on columns in the table, then CREATE INDEX statements for those indexes are also written to the SAS log.

  When you are transferring a table to a DBMS that is supported by SAS/ACCESS software, it is helpful to know how it is defined. To find out more information about a table, use the FEEDBACK option or the CONTENTS statement in the DATASETS procedure.
- The DESCRIBE VIEW statement writes a view definition to the SAS log. If you use a PROC SQL view in the DESCRIBE VIEW statement that is based on or derived from another view, then you might want to use the FEEDBACK option in the PROC SQL statement. This option displays in the SAS log how the underlying view is defined and expands any expressions that are used in this view definition. The CONTENTS statement in DATASETS procedure can also be used with a view to find out more information.
- The DESCRIBE TABLE CONSTRAINTS statement lists the integrity constraints that are defined for the specified table(s).

DISCONNECT Statement

Ends the connection with a DBMS that is supported by a SAS/ACCESS interface.

Requirement: SAS/ACCESS software is required. For more information on this statement, refer to your SAS/ACCESS documentation.

See also: "Connecting to a DBMS Using the SQL Procedure Pass-Through Facility" on page 1115

DISCONNECT FROM *dbms-name* | *alias*;

Arguments

alias
specifies the alias that is defined in the CONNECT statement.

dbms-name
specifies the DBMS from which you want to end the connection (for example, DB2 or ORACLE). The name you specify should match the name that is specified in the CONNECT statement.

Details

- An implicit COMMIT is performed before the DISCONNECT statement ends the DBMS connection. If a DISCONNECT statement is not submitted, then implicit DISCONNECT and COMMIT actions are performed and the connection to the DBMS is broken when PROC SQL terminates.
- PROC SQL continues executing until you submit a QUIT statement, another SAS procedure, or a DATA step.

DROP Statement

Deletes tables, views, or indexes.

Restriction: You cannot use DROP TABLE or DROP INDEX on a table that is accessed by an engine that does not support UPDATE processing.

DROP TABLE *table-name* <, ... *table-name*>;

DROP VIEW *view-name* <, ... *view-name*>;

DROP INDEX *index-name* <, ... *index-name*>
  FROM *table-name*;

Arguments

index-name
specifies an index that exists on *table-name*.

table-name
specifies a PROC SQL table. *table-name* can be a one-level name, a two-level *libref.table* name, or a physical pathname that is enclosed in single quotation marks.

view-name
specifies a SAS data view of any type: PROC SQL view, SAS/ACCESS view, or DATA step view. *view-name* can be a one-level name, a two-level *libref.view* name, or a physical pathname that is enclosed in single quotation marks.

Details

- □ If you drop a table that is referenced in a view definition and try to execute the view, then an error message is written to the SAS log that states that the table does not exist. Therefore, remove references in queries and views to any table(s) and view(s) that you drop.
- □ If you drop a table with indexed columns, then all the indexes are automatically dropped. If you drop a composite index, then the index is dropped for all the columns that are named in that index.
- □ You can use the DROP statement to drop a table or view in an external database that is accessed with the Pass-Through Facility or SAS/ACCESS LIBNAME statement, but not for an external database table or view that is described by a SAS/ACCESS view.

EXECUTE Statement

Sends a DBMS-specific SQL statement to a DBMS that is supported by a SAS/ACCESS interface.

Requirement: SAS/ACCESS software is required. For more information on this statement, refer to your SAS/ACCESS documentation.

See also: “Connecting to a DBMS Using the SQL Procedure Pass-Through Facility” on page 1115 and the SQL documentation for your DBMS.

EXECUTE (*dbms-SQL-statement*)
  BY *dbms-name* | *alias*;

Arguments

alias
specifies an optional alias that is defined in the CONNECT statement. Note that *alias* must be preceded by the keyword BY.

dbms-name
identifies the DBMS to which you want to direct the DBMS statement (for example, ORACLE or DB2).

dbms-SQL-statement
is any DBMS-specific SQL statement, except the SELECT statement, that can be executed by the DBMS-specific dynamic SQL.

Details

- □ If your DBMS supports multiple connections, then you can use the alias that is defined in the CONNECT statement. This alias directs the EXECUTE statements to a specific DBMS connection.
- □ Any return code or message that is generated by the DBMS is available in the macro variables SQLXRC and SQLXMSG after the statement completes.

INSERT Statement

Adds rows to a new or existing table or view.

Restriction: You cannot use INSERT INTO on a table that is accessed with an engine that does not support UPDATE processing.

Featured in: Example 1 on page 1125

---

❶ **INSERT INTO** *table-name* | *sas/access-view* | *proc-sql-view* <(*column*<, … *column*>)>
  SET *column*=sql-expression
    <, … *column*=sql-expression>
  <**SET** *column*=sql-expression
    <, … *column*=sql-expression>>;

❷ **INSERT INTO** *table-name* | *sas/access-view* | *proc-sql-view* <(*column*<, … *column*>)>
  VALUES (*value* <, … *value*>)
    <… **VALUES** (*value* <, … *value*>)>;

❸ **INSERT INTO** *table-name* | *sas/access-view* | *proc-sql-view*
  <(*column*<, …*column*>)> query-expression;

Arguments

column
specifies the column into which you are inserting rows.

proc-sql-view
specifies a PROC SQL view into which you are inserting rows. *proc-sql-view* can be a one-level name, a two-level *libref.view* name, or a physical pathname that is enclosed in single quotation marks.

query-expression
See “query-expression” on page 1093.

sas/access-view
specifies a SAS/ACCESS view into which you are inserting rows.

sql-expression
See “sql-expression” on page 1099.

table-name
specifies a PROC SQL table into which you are inserting rows. *table-name* can be a one-level name, a two-level *libref.table* name, or a physical pathname that is enclosed in single quotation marks.

value
is a data value.

CAUTION:
Recursive table references can cause data integrity problems. While it is possible to recursively reference the target table of an INSERT statement, doing so can cause data integrity problems and incorrect results. Constructions such as the following should be avoided:

```
proc sql;
   insert into a
      select var1, var2
      from a
      where var1 > 0;
```

△

Methods for Inserting Values

❶ The first form of the INSERT statement uses the SET clause, which specifies or alters the values of a column. You can use more than one SET clause per INSERT statement, and each SET clause can set the values in more than one column. Multiple SET clauses are not separated by commas. If you specify an optional list of columns, then you can set a value only for a column that is specified in the list of columns to be inserted.

❷ The second form of the INSERT statement uses the VALUES clause. This clause can be used to insert lists of values into a table. You can either give a value for each column in the table or give values just for the columns specified in the list of column names. One row is inserted for each VALUES clause. Multiple VALUES clauses are not separated by commas. The order of the values in the VALUES clause matches the order of the column names in the INSERT column list or, if no list was specified, the order of the columns in the table.

❸ The third form of the INSERT statement inserts the results of a query-expression into a table. The order of the values in the query-expression matches the order of the column names in the INSERT column list or, if no list was specified, the order of the columns in the table.

Note: If the INSERT statement includes an optional list of column names, then only those columns are given values by the statement. Columns that are in the table but not listed are given missing values. △

Inserting Rows through Views

You can insert one or more rows into a table through a view, with some restrictions. See “Updating PROC SQL and SAS/ACCESS Views” on page 1121.

Adding Values to an Indexed Column

If an index is defined on a column and you insert a new row into the table, then that value is added to the index. You can display information about indexes with

- the CONTENTS statement in the DATASETS procedure. See “CONTENTS Statement” on page 323.
- the DICTIONARY.INDEXES table. See “Using the DICTIONARY Tables” on page 1116 for more information.

For more information on creating and using indexes, see “CREATE INDEX Statement” on page 1043.

RESET Statement

Resets PROC SQL options without restarting the procedure.

Featured in: Example 5 on page 1134

RESET <*option(s)*>;

The RESET statement enables you to add, drop, or change the options in PROC SQL without restarting the procedure. See "PROC SQL Statement" on page 1033 for a description of the options.

SELECT Statement

Selects columns and rows of data from tables and views.

Restriction: The clauses in the SELECT statement must appear in the order shown.

See also: "table-expression" on page 1113, "query-expression" on page 1093

SELECT <DISTINCT> *object-item* <, …*object-item*>

<**INTO** *macro-variable-specification*
<, … *macro-variable-specification*>>
FROM *from-list*
<**WHERE** sql-expression>
<**GROUP BY** *group-by-item*
<, … *group-by-item*>>
<**HAVING** sql-expression>
<**ORDER BY** *order-by-item*
<, … *order-by-item*>>;

SELECT Clause

Lists the columns that will appear in the output.

See Also: "column-definition" on page 1075

Featured in: Example 1 on page 1125 and Example 2 on page 1127

SELECT <DISTINCT> *object-item* <, … *object-item*>

Arguments

alias
assigns a temporary, alternate name to the column.

DISTINCT
eliminates duplicate rows.

Featured in: Example 13 on page 1154

object-item
is one of the following:

*
represents all columns in the tables or views that are listed in the FROM clause.

case-expression <**AS** *alias*>
derives a column from a CASE expression. See “CASE expression” on page 1073.

column-name <<**AS**> *alias*>
<column-modifier <… column-modifier>>
names a single column. See “column-name” on page 1078 and “column-modifier” on page 1076.

sql-expression <**AS** *alias*>
<column-modifier <… column-modifier>>
derives a column from an sql-expression. See “sql-expression” on page 1099 and “column-modifier” on page 1076.

table-name.*
specifies all columns in the PROC SQL table that is specified in *table-name*.

table-alias.*
specifies all columns in the PROC SQL table that has the alias that is specified in *table-alias*.

view-name.*
specifies all columns in the SAS data view that is specified in *view-name*.

view-alias.*
specifies all columns in the SAS data view that has the alias that is specified in *view-alias*.

Asterisk (*) Notation

The asterisk (*) represents all columns of the table(s) listed in the FROM clause. When an asterisk is not prefixed with a table name, all the columns from all tables in the FROM clause are included; when it is prefixed (for example, *table-name.** or *table-alias.**), all the columns from that table only are included.

Column Aliases

A column alias is a temporary, alternate name for a column. Aliases are specified in the SELECT clause to name or rename columns so that the result table is clearer or easier to read. Aliases are often used to name a column that is the result of an arithmetic expression or summary function. An alias is one word only. If you need a longer column name, then use the LABEL= column-modifier, as described in “column-modifier” on page 1076. The keyword AS is not required with a column alias.

Column aliases are optional, and each column name in the SELECT clause can have an alias. After you assign an alias to a column, you can use the alias to refer to that column in other clauses.

If you use a column alias when creating a PROC SQL view, then the alias becomes the permanent name of the column for each execution of the view.

INTO Clause

Stores the value of one or more columns for use later in another PROC SQL query or SAS statement.

Restriction: An INTO clause cannot be used in a CREATE TABLE statement.

See also: “Using Macro Variables Set by PROC SQL” on page 1119

INTO *macro-variable-specification*
    <, … *macro-variable-specification*>

Arguments

macro-variable
specifies a SAS macro variable that stores the values of the rows that are returned.

macro-variable-specification
is one of the following:

:*macro-variable* <SEPARATED BY '*character(s)*' <NOTRIM>>
stores the values that are returned into a single macro variable.

:*macro-variable-1* – :*macro-variable-n* <NOTRIM>
stores the values that are returned into a range of macro variables.

NOTRIM
protects the leading and trailing blanks from being deleted from values that are stored in a range of macro variables or multiple values that are stored in a single macro variable.

SEPARATED BY '*character*'
specifies a character that separates the values of the rows.

Details

- Use the INTO clause only in the outer query of a SELECT statement and not in a subquery.
- When storing a single value into a macro variable, PROC SQL preserves leading or trailing blanks. However, when storing values into a range of macro variables, or when using the SEPARATED BY option to store multiple values in one macro variable, PROC SQL trims leading or trailing blanks unless you use the NOTRIM option.
- You can put multiple rows of the output into macro variables. You can check the PROC SQL macro variable SQLOBS to see the number of rows that are produced by a query-expression. See “Using Macro Variables Set by PROC SQL” on page 1119 for more information on SQLOBS.

Examples

These examples use the PROCLIB.HOUSES table:

```
                    The SAS System                    1

                 Style       SqFeet
                 ------------------
                 CONDO          900
                 CONDO         1000
                 RANCH         1200
                 RANCH         1400
                 SPLIT         1600
                 SPLIT         1800
                 TWOSTORY      2100
                 TWOSTORY      3000
                 TWOSTORY      1940
                 TWOSTORY      1860
```

With the *macro-variable-specification*, you can do the following:

□ You can create macro variables based on the first row of the result.

```
proc sql noprint;
   select style, sqfeet
      into :style, :sqfeet
      from proclib.houses;

%put &style &sqfeet;
```

The results are written to the SAS log:

```
1   proc sql noprint;
2      select style, sqfeet
3         into :style, :sqfeet
4         from proclib.houses;
5
6   %put &style &sqfeet;
CONDO         900
```

□ You can create one new macro variable per row in the result of the SELECT statement. This example shows how you can request more values for one column than for another. The hyphen (-) is used in the INTO clause to imply a range of macro variables. You can use either of the keywords THROUGH or THRU instead of a hyphen.

The following PROC SQL step puts the values from the first four rows of the PROCLIB.HOUSES table into macro variables:

```
proc sql noprint;
select distinct Style, SqFeet
   into :style1 - :style3, :sqfeet1 - :sqfeet4
   from proclib.houses;

%put &style1 &sqfeet1;
%put &style2 &sqfeet2;
%put &style3 &sqfeet3;
%put &sqfeet4;
```

The %PUT statements write the results to the SAS log:

```
1   proc sql noprint;
2   select distinct style, sqfeet
3      into :style1 - :style3, :sqfeet1 - :sqfeet4
4      from proclib.houses;
5
6   %put &style1 &sqfeet1;
CONDO 900
7   %put &style2 &sqfeet2;
CONDO 1000
8   %put &style3 &sqfeet3;
RANCH 1200
9   %put &sqfeet4;
1400
```

□ You can concatenate the values of one column into one macro variable. This form is useful for building up a list of variables or constants.

```
proc sql noprint;
   select distinct style
      into :s1 separated by ','
      from proclib.houses;

%put &s1;
```

The results are written to the SAS log:

```
3    proc sql noprint;
4       select distinct style
5          into :s1 separated by ','
6          from proclib.houses;
7
8   %put &s1

CONDO,RANCH,SPLIT,TWOSTORY
```

□ You can use leading zeros in order to create a range of macro variable names, as shown in the following example:

```
proc sql noprint;
   select SqFeet
      into :sqfeet01 - :sqfeet10
   from proclib.houses;

%put &sqfeet01 &sqfeet02 &sqfeet03 &sqfeet04 &sqfeet05;
%put &sqfeet06 &sqfeet07 &sqfeet08 &sqfeet09 &sqfeet10;
```

The results are written to the SAS log:

```
11   proc sql noprint;
   12      select sqfeet
   13         into :sqfeet01 - :sqfeet10
   14      from proclib.houses;

15   %put &sqfeet01 &sqfeet02 &sqfeet03 &sqfeet04 &sqfeet05;
900 1000 1200 1400 1600
16   %put &sqfeet06 &sqfeet07 &sqfeet08 &sqfeet09 &sqfeet10;
1800 2100 3000 1940 1860
```

- You can prevent leading and trailing blanks from being trimmed from values that are stored in macro variables. By default, when storing values in a range of macro variables or when storing multiple values in one macro variable (with the SEPARATED BY option), PROC SQL trims the leading and trailing blanks from the values before creating the macro variables. If you do not want the blanks to be trimmed, then add the NOTRIM option, as shown in the following example:

```
proc sql noprint;
   select style, sqfeet
      into :style1 - :style4 notrim,
           :sqfeet separated by ',' notrim
      from proclib.houses;

%put *&style1* *&sqfeet*;
%put *&style2* *&sqfeet*;
%put *&style3* *&sqfeet*;
%put *&style4* *&sqfeet*;
```

The results are written to the SAS log, as shown in the following output:

```
3   proc sql noprint;
4      select style, sqfeet
5         into :style1 - :style4 notrim,
6              :sqfeet separated by ',' notrim
7         from proclib.houses;
8
9   %put *&style1* *&sqfeet*;
*CONDO   * *     900,    1000,    1200,    1400,    1600,    1800,    2100,
 3000,    1940,    1860*
10  %put *&style2* *&sqfeet*;
*CONDO   * *     900,    1000,    1200,    1400,    1600,    1800,    2100,
 3000,    1940,    1860**
11  %put *&style3* *&sqfeet*;
*RANCH   * *     900,    1000,    1200,    1400,    1600,    1800,    2100,
 3000,    1940,    1860**
12  %put *&style4* *&sqfeet*;
*RANCH   * *     900,    1000,    1200,    1400,    1600,    1800,    2100,
 3000,    1940,    1860**
```

FROM Clause

Specifies source tables or views.

Featured in: Example 1 on page 1125, Example 4 on page 1131, Example 9 on page 1145, and Example 10 on page 1148

FROM *from-list*

Arguments

alias
specifies a temporary, alternate name for a table, view, or in-line view that is specified in the FROM clause.

column
names the column that appears in the output. The column names that you specify are matched by position to the columns in the output.

from-list
is one of the following:

table-name <<**AS**> *alias*>
names a single PROC SQL table. *table-name* can be a one-level name, a two-level *libref.table* name, or a physical pathname that is enclosed in single quotation marks.

view-name <<**AS**> *alias*>
names a single SAS data view. *view-name* can be a one-level name, a two-level *libref.view* name, or a physical pathname that is enclosed in single quotation marks.

joined-table
specifies a join. See "joined-table" on page 1082.

(query-expression) <**AS** *alias*>
<(*column* <, ... *column*>)>
specifies an in-line view. See "query-expression" on page 1093.

CONNECTION TO
specifies a DBMS table. See "CONNECTION TO" on page 1079.

Note: With *table-name* and *view-name*, you can use data set options by placing them in parentheses immediately after *table-name* or *view-name*. See "Using SAS Data Set Options with PROC SQL" on page 1114 for details. △

Table Aliases

A table alias is a temporary, alternate name for a table that is specified in the FROM clause. Table aliases are prefixed to column names to distinguish between columns that are common to multiple tables. Column names in reflexive joins (joining a table with itself) must be prefixed with a table alias in order to distinguish which copy of the table the column comes from. Column names in other kinds of joins must be prefixed with table aliases or table names unless the column names are unique to those tables.

The optional keyword AS is often used to distinguish a table alias from other table names.

In-Line Views

The FROM clause can itself contain a query-expression that takes an optional table alias. This kind of nested query-expression is called an *in-line view*. An in-line view is any query-expression that would be valid in a CREATE VIEW statement. PROC SQL can support many levels of nesting, but it is limited to 32 tables in any one query. The 32-table limit includes underlying tables that may contribute to views that are specified in the FROM clause.

An in-line view saves you a programming step. Rather than creating a view and referring to it in another query, you can specify the view *in-line* in the FROM clause.

Characteristics of in-line views include the following:

- An in-line view is not assigned a permanent name, although it can take an alias.
- An in-line view can be referred to only in the query in which it is defined. It cannot be referenced in another query.
- You cannot use an ORDER BY clause in an in-line view.

- The names of columns in an in-line view can be assigned in the object-item list of that view or with a parenthesized list of names following the alias. This syntax can be useful for renaming columns. See Example 10 on page 1148 for an example.
- In order to visually separate an in-line view from the rest of the query, you can enclose the in-line view in any number of pairs of parentheses. Note that if you specify an alias for the in-line view, the alias specification must appear outside the outermost pair of parentheses for that in-line view.

WHERE Clause

Subsets the output based on specified conditions.

Featured in: Example 4 on page 1131 and Example 9 on page 1145

WHERE sql-expression

Argument

sql-expression
See "sql-expression" on page 1099.

Details

- When a condition is met (that is, the condition resolves to true), those rows are displayed in the result table; otherwise, no rows are displayed.
- You cannot use summary functions that specify only one column. For example:

```
where max(measure1) > 50;
```

However, this WHERE clause will work:

```
where max(measure1,measure2) > 50;
```

GROUP BY Clause

Specifies how to group the data for summarizing.

Featured in: Example 8 on page 1143 and Example 12 on page 1152

GROUP BY *group-by-item* <, …, *group-by-item*>

Arguments

group-by-item
is one of the following:

integer
is a positive integer that equates to a column's position.

column-name
is the name of a column or a column alias. See "column-name" on page 1078.

sql-expression
See "sql-expression" on page 1099.

Details

- You can specify more than one *group-by-item* to get more detailed reports. Both the grouping of multiple items and the BY statement of a PROC step are evaluated in similar ways. If more than one *group-by-item* is specified, then the first one determines the major grouping.
- Integers can be substituted for column names (that is, SELECT object-items) in the GROUP BY clause. For example, if the *group-by-item* is 2, then the results are grouped by the values in the second column of the SELECT clause list. Using integers can shorten your coding and enable you to group by the value of an unnamed expression in the SELECT list. Note that if you use a floating-point value (for example, 2.3), then PROC SQL ignores the decimal portion.
- The data does not have to be sorted in the order of the group-by values because PROC SQL handles sorting automatically. You can use the ORDER BY clause to specify the order in which rows are displayed in the result table.
- If you specify a GROUP BY clause in a query that does not contain a summary function, then your clause is transformed into an ORDER BY clause and a message to that effect is written to the SAS log.
- You can group the output by the values that are returned by an expression. For example, if X is a numeric variable, then the output of the following is grouped by the integer portion of values of X:

  ```
  select x, sum(y)
  from table1
  group by int(x);
  ```

  Similarly, if Y is a character variable, then the output of the following is grouped by the second character of values of Y:

  ```
  select sum(x), y
  from table1
  group by substring(y from 2 for 1);
  ```

  Note that an expression that contains only numeric literals (and functions of numeric literals) or only character literals (and functions of character literals) is ignored.

  An expression in a GROUP BY clause cannot be a summary function. For example, the following GROUP BY clause is not valid:

  ```
  group by sum(x)
  ```

HAVING Clause

Subsets grouped data based on specified conditions.

Featured in: Example 8 on page 1143 and Example 12 on page 1152

HAVING sql-expression

Argument

sql-expression
See "sql-expression" on page 1099.

Subsetting Grouped Data

The HAVING clause is used with at least one summary function and an optional GROUP BY clause to summarize groups of data in a table. A HAVING clause is any valid SQL expression that is evaluated as either true or false for each group in a query. Alternatively, if the query involves remerged data, then the HAVING expression is evaluated for each row that participates in each group. The query must include one or more summary functions.

Typically, the GROUP BY clause is used with the HAVING expression and defines the group(s) to be evaluated. If you omit the GROUP BY clause, then the summary function and the HAVING clause treat the table as one group.

The following PROC SQL step uses the PROCLIB.PAYROLL table (shown in Example 2 on page 1127) and groups the rows by Gender to determine the oldest employee of each gender. In SAS, dates are stored as integers. The lower the birth date as an integer, the greater the age. The expression **birth=min(birth)** is evaluated for each row in the table. When the minimum birth date is found, the expression becomes true and the row is included in the output.

```
proc sql;
   title 'Oldest Employee of Each Gender';
   select *
      from proclib.payroll
      group by gender
      having birth=min(birth);
```

Note: This query involves remerged data because the values returned by a summary function are compared to values of a column that is not in the GROUP BY clause. See "Remerging Data" on page 1110 for more information about summary functions and remerging data. △

ORDER BY Clause

Specifies the order in which rows are displayed in a result table.

See also: "query-expression" on page 1093

Featured in: Example 11 on page 1150

ORDER BY *order-by-item* <**ASC**|**DESC**><, … *order-by-item* <**ASC**|**DESC**>>;

Arguments

order-by-item
is one of the following:

integer
equates to a column's position.

column-name
is the name of a column or a column alias. See "column-name" on page 1078.

sql-expression
See "sql-expression" on page 1099.

ASC
orders the data in ascending order. This is the default order; if neither ASC nor DESC is specified, the data is ordered in ascending order.

DESC
orders the data in descending order.

Details

- The ORDER BY clause sorts the result of a query expression according to the order specified in that query. When this clause is used, the default ordering sequence is ascending, from the lowest value to the highest. You can use the SORTSEQ= option to change the collating sequence for your output. See "PROC SQL Statement" on page 1033.
- If an ORDER BY clause is omitted, then a particular order to the output rows, such as the order in which the rows are encountered in the queried table, cannot be guaranteed. Without an ORDER BY clause, the order of the output rows is determined by the internal processing of PROC SQL, the default collating sequence of SAS, and your operating environment. Therefore, if you want your result table to appear in a particular order, then use the ORDER BY clause.
- If more than one *order-by-item* is specified (separated by commas), then the first one determines the major sort order.
- Integers can be substituted for column names (that is, SELECT object-items) in the ORDER BY clause. For example, if the *order-by-item* is 2 (an integer), then the results are ordered by the values of the second column. If a query-expression includes a set operator (for example, UNION), then use integers to specify the order. Doing so avoids ambiguous references to columns in the table expressions. Note that if you use a floating-point value (for example, 2.3) instead of an integer, then PROC SQL ignores the decimal portion.
- In the ORDER BY clause, you can specify any column of a table or view that is specified in the FROM clause of a query-expression, regardless of whether that column has been included in the query's SELECT clause. For example, this query produces a report ordered by the descending values of the population change for each country from 1990 to 1995:

```
proc sql;
   select country
```

```
      from census
      order by pop95-pop90 desc;


   NOTE: The query as specified involves
         ordering by an item that
         doesn't appear in its SELECT clause.
```

- You can order the output by the values that are returned by an expression. For example, if X is a numeric variable, then the output of the following is ordered by the integer portion of values of X:

  ```
  select x, y
  from table1
  order by int(x);
  ```

  Similarly, if Y is a character variable, then the output of the following is ordered by the second character of values of Y:

  ```
  select x, y
  from table1
  order by substring(y from 2 for 1);
  ```

  Note that an expression that contains only numeric literals (and functions of numeric literals) or only character literals (and functions of character literals) is ignored.

UPDATE Statement

Modifies a column's values in existing rows of a table or view.

Restriction: You cannot use UPDATE on a table that is accessed by an engine that does not support UPDATE processing.

Featured in: Example 3 on page 1129

UPDATE *table-name* | *sas/access-view* | *proc-sql-view* <**AS** *alias*>
  SET *column*=sql-expression
    <, ... *column*=sql-expression>
  <**SET** *column*=sql-expression
    <, ... *column*=sql-expression>>
  <**WHERE** sql-expression>;

Arguments

alias
assigns an alias to *table-name*, *sas/access-view*, or *proc-sql-view*.

column
specifies a column in *table-name*, *sas/access-view*, or *proc-sql-view*.

sas/access-view
specifies a SAS/ACCESS view.

sql-expression
See "sql-expression" on page 1099.

table-name
specifies a PROC SQL table. *table-name* can be a one-level name, a two-level *libref.table* name, or a physical pathname that is enclosed in single quotation marks.

proc-sql-view
specifies a PROC SQL view. *proc-sql-view* can be a one-level name, a two-level *libref.view* name, or a physical pathname that is enclosed in single quotation marks.

Updating Tables through Views

You can update one or more rows of a table through a view, with some restrictions. See "Updating PROC SQL and SAS/ACCESS Views" on page 1121.

Details

- Any column that is not modified retains its original values, except in certain queries using the CASE expression. See "CASE expression" on page 1073 for a description of CASE expressions.
- To add, drop, or modify a column's definition or attributes, use the ALTER TABLE statement, described in "ALTER TABLE Statement" on page 1038.
- In the SET clause, a column reference on the left side of the equal sign can also appear as part of the expression on the right side of the equal sign. For example, you could use this expression to give employees a $1,000 holiday bonus:

```
set salary=salary + 1000
```

- If you omit the WHERE clause, then all the rows are updated. When you use a WHERE clause, only the rows that meet the WHERE condition are updated.
- When you update a column and an index has been defined for that column, the values in the updated column continue to have the index defined for them.

VALIDATE Statement

Checks the accuracy of a query-expression's syntax and semantics without executing the expression.

VALIDATE query-expression;

Argument

query-expression
See "query-expression" on page 1093.

Details

- □ The VALIDATE statement writes a message in the SAS log that states that the query is valid. If there are errors, then VALIDATE writes error messages to the SAS log.
- □ The VALIDATE statement can also be included in applications that use the macro facility. When used in such an application, VALIDATE returns a value that indicates the query-expression's validity. The value is returned through the macro variable SQLRC (a short form for SQL return code). For example, if a SELECT statement is valid, then the macro variable SQLRC returns a value of 0. See "Using Macro Variables Set by PROC SQL" on page 1119 for more information.

SQL Procedure Component Dictionary

This section describes the components that are used in SQL procedure statements. *Components* are the items in PROC SQL syntax that appear in roman type.

Most components are contained in clauses within the statements. For example, the basic SELECT statement is composed of the SELECT and FROM clauses, where each clause contains one or more components. Components can also contain other components.

For easy reference, components appear in alphabetical order, and some terms are referred to before they are defined. Use the index or the "See Also" references to refer to other statement or component descriptions that may be helpful.

BETWEEN condition

Selects rows where column values are within a range of values.

sql-expression <NOT> **BETWEEN** sql-expression
    AND sql-expression

Argument

sql-expression
is described in "sql-expression" on page 1099.

Details

- □ The sql-expressions must be of compatible data types. They must be either all numeric or all character types.
- □ Because a BETWEEN condition evaluates the boundary values as a range, it is not necessary to specify the smaller quantity first.
- □ You can use the NOT logical operator to exclude a range of numbers, for example, to eliminate customer numbers between 1 and 15 (inclusive) so that you can retrieve data on more recently acquired customers.

- PROC SQL supports the same comparison operators that the DATA step supports. For example:

```
x between 1 and 3
x between 3 and 1
1<=x<=3
x>=1 and x<=3
```

BTRIM function

Removes blanks or specified characters from the beginning, the end, or both the beginning and end of a character string.

BTRIM (<<*btrim-specification*> <'*btrim-character*' FROM>> sql-expression)

Arguments

btrim-specification
is one of the following:

LEADING
removes the blanks or specified characters from the beginning of the character string.

TRAILING
removes the blanks or specified characters from the end of the character string.

BOTH
removes the blanks or specified characters from both the beginning and the end of the character string.

Default: BOTH

btrim-character
is a single character that is to be removed from the character string. The default character is a blank.

sql-expression
must resolve to a character string or character variable and is described in "sql-expression" on page 1099.

Details

The BTRIM function operates on character strings. BTRIM removes one or more instances of a single character (the value of *btrim-character*) from the beginning, the end, or both the beginning and end of a string, depending whether LEADING, TRAILING, or BOTH is specified. If *btrim-specification* is not specified, then BOTH is used. If *btrim-character* is omitted, then blanks are removed.

Note: SAS adds trailing blanks to character values that are shorter than the length of the variable. Suppose you have a character variable Z, with length 10, and a value **xxabcxx**. SAS stores the value with three blanks after the last x (for a total length of 10). If you attempt to remove all the x characters with

```
btrim(both 'x' from z)
```

then the result is **abcxx** because PROC SQL sees the trailing characters as blanks, not the x character. In order to remove all the x characters, use

```
btrim(both 'x' from btrim(z))
```

The inner BTRIM function removes the trailing blanks before passing the value to the outer BTRIM function. △

CALCULATED

Refers to columns already calculated in the SELECT clause.

CALCULATED *column-alias*

Argument

column-alias
is the name that is assigned to the column in the SELECT clause.

Referencing a CALCULATED Column

CALCULATED enables you to use the results of an expression in the same SELECT clause or in the WHERE clause. It is valid only when used to refer to columns that are calculated in the immediate query expression.

CASE expression

Selects result values that satisfy specified conditions.

Featured in: Example 3 on page 1129 and Example 13 on page 1154

CASE <*case-operand*>
  WHEN *when-condition* **THEN** *result-expression*
  <…**WHEN** *when-condition* **THEN** *result-expression*>
  <**ELSE** *result-expression*>
  END

Arguments

case-operand

is a valid sql-expression that resolves to a table column whose values are compared to all the *when-conditions*. See “sql-expression” on page 1099.

when-condition

- When *case-operand* is specified, *when-condition* is a shortened sql-expression that assumes *case-operand* as one of its operands and that resolves to true or false.
- When *case-operand* is not specified, *when-condition* is an sql-expression that resolves to true or false.

result-expression

is an sql-expression that resolves to a value.

Details

The CASE expression selects values if certain conditions are met. A CASE expression returns a single value that is conditionally evaluated for each row of a table (or view). Use the WHEN-THEN clauses when you want to execute a CASE expression for some but not all of the rows in the table that is being queried or created. An optional ELSE expression gives an alternative action if no THEN expression is executed.

When you omit *case-operand*, *when-condition* is evaluated as a Boolean (true or false) value. If *when-condition* returns a nonzero, nonmissing result, then the WHEN clause is true. If *case-operand* is specified, then it is compared with *when-condition* for equality. If *case-operand* equals *when-condition*, then the WHEN clause is true.

If the *when-condition* is true for the row that is being executed, then the *result-expression* that follows THEN is executed. If *when-condition* is false, then PROC SQL evaluates the next *when-condition* until they are all evaluated. If every *when-condition* is false, then PROC SQL executes the ELSE expression, and its result becomes the CASE expression's result. If no ELSE expression is present and every *when-condition* is false, then the result of the CASE expression is a missing value.

You can use a CASE expression as an item in the SELECT clause and as either operand in an sql-expression.

Example

The following two PROC SQL steps show two equivalent CASE expressions that create a character column with the strings in the THEN clause. The CASE expression in the second PROC SQL step is a shorthand method that is useful when all the comparisons are with the same column.

```
proc sql;
   select Name, case
                when Continent = 'North America' then 'Continental U.S.'
                when Continent = 'Oceania' then 'Pacific Islands'
                else 'None'
                end as Region
      from states;

proc sql;
   select Name, case Continent
                when 'North America' then 'Continental U.S.'
                when 'Oceania' then 'Pacific Islands'
```

```
            else 'None'
            end as Region
      from states;
```

Note: When you use the shorthand method, the conditions must all be equality tests. That is, they cannot use comparison operators or other types of operators. △

COALESCE Function

Returns the first nonmissing value from a list of columns.

Featured in: Example 7 on page 1138

COALESCE (column-name <, … column-name>)

Arguments

column-name
is described in "column-name" on page 1078.

Details

COALESCE accepts one or more column names of the same data type. The COALESCE function checks the value of each column in the order in which they are listed and returns the first nonmissing value. If only one column is listed, the COALESCE function returns the value of that column. If all the values of all arguments are missing, the COALESCE function returns a missing value.

In some SQL DBMSs, the COALESCE function is called the IFNULL function. See "PROC SQL and the ANSI Standard" on page 1122 for more information.

Note: If your query contains a large number of COALESCE function calls, it might be more efficient to use a natural join instead. See "Natural Joins" on page 1088. △

column-definition

Defines PROC SQL's data types and dates.

See also: "column-modifier" on page 1076

Featured in: Example 1 on page 1125

column data-type <column-modifier <… column-modifier>>

Arguments

column
is a column name.

column-modifier
is described in "column-modifier" on page 1076.

data-type
is one of the following data types:

CHARACTER | VARCHAR <(*width*)>
indicates a character column with a column width of *width*. The default column width is eight characters.

INTEGER | SMALLINT
indicates an integer column.

DECIMAL | NUMERIC | FLOAT <(*width*<, *ndec*>)>
indicates a floating-point column with a column width of *width* and *ndec* decimal places.

REAL | DOUBLE PRECISION
indicates a floating-point column.

DATE
indicates a date column.

Details

- SAS supports many but not all of the data types that SQL-based databases support.
- For all the numeric data types (INTEGER, SMALLINT, DECIMAL, NUMERIC, FLOAT, REAL, DOUBLE PRECISION, and DATE), the SQL procedure defaults to the SAS data type NUMERIC. The *width* and *ndec* arguments are ignored; PROC SQL creates all numeric columns with the maximum precision allowed by SAS. If you want to create numeric columns that use less storage space, then use the LENGTH statement in the DATA step. The various numeric data type names, along with the *width* and *ndec* arguments, are included for compatibility with other SQL software.
- For the character data types (CHARACTER and VARCHAR), the SQL procedure defaults to the SAS data type CHARACTER. The *width* argument is honored.
- The CHARACTER, INTEGER, and DECIMAL data types can be abbreviated to CHAR, INT, and DEC, respectively.
- A column that is declared with DATE is a SAS numeric variable with a date informat or format. You can use any of the column-modifiers to set the appropriate attributes for the column that is being defined. See *SAS Language Reference: Dictionary* for more information on dates.

column-modifier

Sets column attributes.

See also: "column-definition" on page 1075 and SELECT Clause on page 1058

Featured in: Example 1 on page 1125 and Example 2 on page 1127

column-modifier

Arguments

column-modifier
is one of the following:

INFORMAT=*informatw.d*
specifies a SAS informat to be used when SAS accesses data from a table or view. You can change one permanent informat to another by using the ALTER statement. PROC SQL stores informats in its table definitions so that other SAS procedures and the DATA step can use this information when they reference tables created by PROC SQL.

See *SAS Language Reference: Dictionary* for more information about informats.

FORMAT=*formatw.d*
specifies a SAS format for determining how character and numeric values in a column are displayed by the query-expression. If the FORMAT= modifier is used in the ALTER, CREATE TABLE, or CREATE VIEW statements, then it specifies the permanent format to be used when SAS displays data from that table or view. You can change one permanent format to another by using the ALTER statement.

See *SAS Language Reference: Dictionary* for more information about formats.

LABEL=*'label'*
specifies a column label. If the LABEL= modifier is used in the ALTER, CREATE TABLE, or CREATE VIEW statements, then it specifies the permanent label to be used when displaying that column. You can change one permanent label to another by using the ALTER statement.

A label can begin with the following characters: a through z, A through Z, 0 through 9, an underscore (_), or a blank space. If you begin a label with any other character, such as pound sign (#), then that character is used as a split character and it splits the label onto the next line wherever it appears. For example:

```
select dropout label=
'#Percentage of#Students Who#Dropped Out'
   from educ(obs=5);
```

If a special character must appear as the first character in the output, then precede it with a space or a forward slash (/).

You can omit the LABEL= part of the column-modifier and still specify a label. Be sure to enclose the label in quotation marks, as in this example:

```
select empname "Names of Employees"
   from sql.employees;
```

If an apostrophe must appear in the label, then type it twice so that SAS reads the apostrophe as a literal. Alternatively, you can use single and double quotation marks alternately (for example, “Date Rec’d”).

LENGTH=*length*
specifies the length of the column. This column modifier is valid only in the context of a SELECT statement.

TRANSCODE=YES|NO
for character columns, specifies whether values can be transcoded. Use TRANSCODE=NO to suppress transcoding. Note that when you create a table by using the CREATE TABLE AS statement, the transcoding attribute for a given character column in the created table is the same as it is in the source table unless

you change it with the TRANSCODE= column modifier. For more information about transcoding, see *SAS National Language Support (NLS): User's Guide*.

Default: YES

Restriction: Suppression of transcoding is not supported for the V6TAPE engine.

Interaction: If the TRANSCODE= attribute is set to NO for any character variable in a table, then PROC CONTENTS prints a transcode column that contains the TRANSCODE= value for each variable in the data set. If all variables in the table are set to the default TRANSCODE= value (YES), then no transcode column is printed.

Details

If you refer to a labeled column in the ORDER BY or GROUP BY clause, then you must use either the column name (not its label), the column's alias, or its ordering integer (for example, `ORDER BY 2`). See the section on SAS statements in *SAS Language Reference: Dictionary* for more information about labels.

column-name

Specifies the column to select.

See also: "column-modifier" on page 1076 and SELECT Clause on page 1058

column-name

column-name
is one of the following:

column
is the name of a column.

table-name.column
is the name of a column in the table *table-name*.

table-alias.column
is the name of a column in the table that is referenced by *table-alias*.

view-name.column
is the name of a column in the view *view-name*.

view-alias.column
is the name of a column in the view that is referenced by *view-alias*.

Details

A column can be referred to by its name alone if it is the only column by that name in all the tables or views listed in the current query-expression. If the same column name exists in more than one table or view in the query-expression, then you must *qualify* each use of the column name by prefixing a reference to the table that contains it. Consider the following examples:

```
SALARY       /* name of the column */
EMP.SALARY   /* EMP is the table or view name */
E.SALARY     /* E is an alias for the table
                or view that contains the
                SALARY column */
```

CONNECTION TO

Retrieves and uses DBMS data in a PROC SQL query or view.

Tip: You can use CONNECTION TO in the SELECT statement's FROM clause as part of the from-list.

See also: "Connecting to a DBMS Using the SQL Procedure Pass-Through Facility" on page 1115 and your SAS/ACCESS documentation.

CONNECTION TO *dbms-name (dbms-query)*

CONNECTION TO *alias (dbms-query)*

Arguments

alias
specifies an alias, if one was defined in the CONNECT statement.

dbms-name
identifies the DBMS that you are using.

dbms-query
specifies the query to send to a DBMS. The query uses the DBMS's dynamic SQL. You can use any SQL syntax that the DBMS understands, even if that is not valid for PROC SQL. However, your DBMS query cannot contain a semicolon because that represents the end of a statement to SAS.

The number of tables that you can join with *dbms-query* is determined by the DBMS. Each CONNECTION TO component counts as one table toward the 32-table PROC SQL limit for joins.

See *SAS/ACCESS for Relational Databases: Reference* for more information about DBMS queries.

CONTAINS condition

Tests whether a string is part of a column's value.

Alias: ?

Restriction: The CONTAINS condition is used only with character operands.

Featured in: Example 7 on page 1138

sql-expression <NOT> **CONTAINS** sql-expression

Argument

sql-expression
is described in “sql-expression” on page 1099.

EXISTS condition

Tests if a subquery returns one or more rows.

See also: “Query Expressions (Subqueries)” on page 1102

<NOT> **EXISTS** (query-expression)

Argument

query-expression
is described in “query-expression” on page 1093.

Details

The EXISTS condition is an operator whose right operand is a subquery. The result of an EXISTS condition is true if the subquery resolves to at least one row. The result of a NOT EXISTS condition is true if the subquery evaluates to zero rows. For example, the following query subsets PROCLIB.PAYROLL (which is shown in Example 2 on page 1127) based on the criteria in the subquery. If the value for STAFF.IDNUM is on the same row as the value `CT` in PROCLIB.STAFF (which is shown in Example 4 on page 1131), then the matching IDNUM in PROCLIB.PAYROLL is included in the output. Thus, the query returns all the employees from PROCLIB.PAYROLL who live in `CT`.

```
proc sql;
  select *
    from proclib.payroll p
    where exists (select *
                    from proclib.staff s
                    where p.idnumber=s.idnum
                          and state='CT');
```

IN condition

Tests set membership.

Featured in: Example 4 on page 1131

sql-expression <NOT> **IN** (query-expression | *constant* <, ... *constant*>)

Arguments

constant
is a number or a quoted character string (or other special notation) that indicates a fixed value. Constants are also called *literals*.

query-expression
is described in "query-expression" on page 1093.

sql-expression
is described in "sql-expression" on page 1099.

Details

An IN condition tests if the column value that is returned by the sql-expression on the left is a member of the set (of constants or values returned by the query-expression) on the right. The IN condition is true if the value of the left-hand operand is in the set of values that are defined by the right-hand operand.

IS condition

Tests for a missing value.

Featured in: Example 5 on page 1134

sql-expression **IS** <NOT> **NULL** | **MISSING**

Argument

sql-expression
is described in "sql-expression" on page 1099.

Details

IS NULL and IS MISSING are predicates that test for a missing value. IS NULL and IS MISSING are used in the WHERE, ON, and HAVING expressions. Each predicate resolves to true if the sql-expression's result is missing and false if it is not missing.

SAS stores a numeric missing value as a period (.) and a character missing value as a blank space. Unlike missing values in some versions of SQL, missing values in SAS always appear first in the collating sequence. Therefore, in Boolean and comparison operations, the following expressions resolve to true in a predicate:

```
 3>null
-3>null
 0>null
```

The SAS way of evaluating missing values differs from that of the ANSI Standard for SQL. According to the Standard, these expressions are NULL. See “sql-expression” on page 1099 for more information on predicates and operators. See “PROC SQL and the ANSI Standard” on page 1122 for more information on the ANSI Standard.

---

joined-table

Joins a table with itself or with other tables or views.

Restrictions: Joins are limited to 32 tables.

See also: FROM Clause on page 1063 and “query-expression” on page 1093

Featured in: Example 4 on page 1131, Example 7 on page 1138, Example 9 on page 1145, Example 13 on page 1154, and Example 14 on page 1158

---

❶ *table-name* <<**AS**> *alias*>, *table-name* <<**AS**> *alias*>
<, … *table-name* <<**AS**> *alias*>>

❷ <(>*table-name* <**INNER**> **JOIN** *table-name*
ON sql-expression<)>

❸ <(>*table-name* **LEFT JOIN** | **RIGHT JOIN** | **FULL JOIN**
table-name **ON** sql-expression<)>

❹ <(>*table-name* **CROSS JOIN** *table-name*<)>

❺ <(>*table-name* **UNION JOIN** *table-name*<)>

❻ <(>*table-name* **NATURAL**
<**INNER** | **FULL** <**OUTER**> | **LEFT** <**OUTER** > | **RIGHT** <**OUTER** >>
JOIN *table-name*<)>

Arguments

alias
specifies an alias for *table-name*. The AS keyword is optional.

sql-expression
is described in “sql-expression” on page 1099.

table-name
can be one of the following:

- □ the name of a PROC SQL table.
- □ the name of a SAS data view or PROC SQL view.
- □ a query-expression. A query-expression in the FROM clause is usually referred to as an *in-line view*. See “FROM Clause” on page 1063 for more information about in-line views.
- □ a connection to a DBMS in the form of the CONNECTION TO component. See “CONNECTION TO” on page 1079 for more information.

table-name can be a one-level name, a two-level *libref.table* name, or a physical pathname that is enclosed in single quotation marks.

Note: If you include parentheses, then be sure to include them in pairs. Parentheses are not valid around comma joins (type ❶). △

Types of Joins

❶❷ Inner join. See "Inner Joins" on page 1084.

❸ Outer join. See "Outer Joins" on page 1086.

❹ Cross join. See "Cross Joins" on page 1087.

❺ Union join. See "Union Joins" on page 1088.

❻ Natural join. See "Natural Joins" on page 1088.

Joining Tables

When multiple tables, views, or query-expressions are listed in the FROM clause, they are processed to form one table. The resulting table contains data from each contributing table. These queries are referred to as *joins*.

Conceptually, when two tables are specified, each row of table A is matched with all the rows of table B to produce an internal or intermediate table. The number of rows in the intermediate table (*Cartesian product*) is equal to the product of the number of rows in each of the source tables. The intermediate table becomes the input to the rest of the query in which some of its rows may be eliminated by the WHERE clause or summarized by a summary function.

A common type of join is an *equijoin*, in which the values from a column in the first table must equal the values of a column in the second table.

Table Limit

PROC SQL can process a maximum of 32 tables for a join. If you are using views in a join, then the number of tables on which the views are based count toward the 32-table limit. Each CONNECTION TO component in the Pass-Through Facility counts as one table.

Specifying the Rows to Be Returned

The WHERE clause or ON clause contains the conditions (sql-expression) under which the rows in the Cartesian product are kept or eliminated in the result table. WHERE is used to select rows from inner joins. ON is used to select rows from inner or outer joins.

The expression is evaluated for each row from each table in the intermediate table described earlier in "Joining Tables" on page 1083. The row is considered to be matching if the result of the expression is true (a nonzero, nonmissing value) for that row.

Note: You can follow the ON clause with a WHERE clause to further subset the query result. See Example 7 on page 1138 for an example. △

Table Aliases

Table aliases are used in joins to distinguish the columns of one table from those in the other table(s). A table name or alias must be prefixed to a column name when you are joining tables that have matching column names. See FROM Clause on page 1063 for more information on table aliases.

Joining a Table with Itself

A single table can be joined with itself to produce more information. These joins are sometimes called *reflexive joins*. In these joins, the same table is listed twice in the FROM clause. Each instance of the table must have a table alias or you will not be able to distinguish between references to columns in either instance of the table. See Example 13 on page 1154 and Example 14 on page 1158 for examples.

Inner Joins

An *inner join* returns a result table for all the rows in a table that have one or more matching rows in the other table(s), as specified by the sql-expression. Inner joins can be performed on up to 32 tables in the same query-expression.

You can perform an inner join by using a list of table-names separated by commas or by using the INNER, JOIN, and ON keywords.

The LEFTTAB and RIGHTTAB tables are used to illustrate this type of join:

```
                 Left Table - LEFTTAB

            Continent  Export    Country
            -----------------------------
            NA         wheat     Canada
            EUR        corn      France
            EUR        rice      Italy
            AFR        oil       Egypt
```

```
                Right Table - RIGHTTAB

            Continent  Export    Country
            -----------------------------
            NA         sugar     USA
            EUR        corn      Spain
            EUR        beets     Belgium
            ASIA       rice      Vietnam
```

The following example joins the LEFTTAB and RIGHTTAB tables to get the *Cartesian product* of the two tables. The Cartesian product is the result of combining every row from one table with every row from another table. You get the Cartesian product when you join two tables and do not subset them with a WHERE clause or ON clause.

```
proc sql;
   title 'The Cartesian Product of';
   title2 'LEFTTAB and RIGHTTAB';
   select *
      from lefttab, righttab;
```

The Cartesian Product of
LEFTTAB and RIGHTTAB

Continent	Export	Country	Continent	Export	Country
NA	wheat	Canada	NA	sugar	USA
NA	wheat	Canada	EUR	corn	Spain
NA	wheat	Canada	EUR	beets	Belgium
NA	wheat	Canada	ASIA	rice	Vietnam
EUR	corn	France	NA	sugar	USA
EUR	corn	France	EUR	corn	Spain
EUR	corn	France	EUR	beets	Belgium
EUR	corn	France	ASIA	rice	Vietnam
EUR	rice	Italy	NA	sugar	USA
EUR	rice	Italy	EUR	corn	Spain
EUR	rice	Italy	EUR	beets	Belgium
EUR	rice	Italy	ASIA	rice	Vietnam
AFR	oil	Egypt	NA	sugar	USA
AFR	oil	Egypt	EUR	corn	Spain
AFR	oil	Egypt	EUR	beets	Belgium
AFR	oil	Egypt	ASIA	rice	Vietnam

The LEFTTAB and RIGHTTAB tables can be joined by listing the table names in the FROM clause. The following query represents an equijoin because the values of Continent from each table are matched. The column names are prefixed with the table aliases so that the correct columns can be selected.

```
proc sql;
   title 'Inner Join';
   select *
      from lefttab as l, righttab as r
      where l.continent=r.continent;
```

Inner Join

Continent	Export	Country	Continent	Export	Country
NA	wheat	Canada	NA	sugar	USA
EUR	corn	France	EUR	corn	Spain
EUR	corn	France	EUR	beets	Belgium
EUR	rice	Italy	EUR	corn	Spain
EUR	rice	Italy	EUR	beets	Belgium

The following PROC SQL step is equivalent to the previous one and shows how to write an equijoin using the INNER JOIN and ON keywords.

```
proc sql;
   title 'Inner Join';
   select *
      from lefttab as l inner join
           righttab as r
      on l.continent=r.continent;
```

See Example 4 on page 1131, Example 13 on page 1154, and Example 14 on page 1158 for more examples.

Outer Joins

Outer joins are inner joins that have been augmented with rows that did not match with any row from the other table in the join. The three types of outer joins are left, right, and full.

A left outer join, specified with the keywords LEFT JOIN and ON, has all the rows from the Cartesian product of the two tables for which the sql-expression is true, plus rows from the first (LEFTTAB) table that do not match any row in the second (RIGHTTAB) table.

```
proc sql;
   title 'Left Outer Join';
   select *
      from lefttab as l left join
           righttab as r
      on l.continent=r.continent;
```

Left Outer Join

Continent	Export	Country	Continent	Export	Country
AFR	oil	Egypt			
EUR	rice	Italy	EUR	beets	Belgium
EUR	corn	France	EUR	beets	Belgium
EUR	rice	Italy	EUR	corn	Spain
EUR	corn	France	EUR	corn	Spain
NA	wheat	Canada	NA	sugar	USA

A right outer join, specified with the keywords RIGHT JOIN and ON, has all the rows from the Cartesian product of the two tables for which the sql-expression is true, plus rows from the second (RIGHTTAB) table that do not match any row in the first (LEFTTAB) table.

```
proc sql;
   title 'Right Outer Join';
   select *
      from lefttab as l right join
           righttab as r
      on l.continent=r.continent;
```

Right Outer Join

Continent	Export	Country	Continent	Export	Country
			ASIA	rice	Vietnam
EUR	rice	Italy	EUR	beets	Belgium
EUR	rice	Italy	EUR	corn	Spain
EUR	corn	France	EUR	beets	Belgium
EUR	corn	France	EUR	corn	Spain
NA	wheat	Canada	NA	sugar	USA

A full outer join, specified with the keywords FULL JOIN and ON, has all the rows from the Cartesian product of the two tables for which the sql-expression is true, plus rows from each table that do not match any row in the other table.

```
proc sql;
   title 'Full Outer Join';
   select *
      from lefttab as l full join
           righttab as r
      on l.continent=r.continent;
```

Full Outer Join

Continent	Export	Country	Continent	Export	Country
AFR	oil	Egypt			
			ASIA	rice	Vietnam
EUR	rice	Italy	EUR	beets	Belgium
EUR	rice	Italy	EUR	corn	Spain
EUR	corn	France	EUR	beets	Belgium
EUR	corn	France	EUR	corn	Spain
NA	wheat	Canada	NA	sugar	USA

See Example 7 on page 1138 for another example.

Cross Joins

A cross join returns as its result table the product of the two tables.

Using the LEFTTAB and RIGHTTAB example tables, the following program demonstrates the cross join:

```
proc sql;
   title 'Cross Join';
   select *
      from lefttab as l cross join
         righttab as r;
```

Cross Join

Continent	Export	Country	Continent	Export	Country
NA	wheat	Canada	NA	sugar	USA
NA	wheat	Canada	EUR	corn	Spain
NA	wheat	Canada	EUR	beets	Belgium
NA	wheat	Canada	ASIA	rice	Vietnam
EUR	corn	France	NA	sugar	USA
EUR	corn	France	EUR	corn	Spain
EUR	corn	France	EUR	beets	Belgium
EUR	corn	France	ASIA	rice	Vietnam
EUR	rice	Italy	NA	sugar	USA
EUR	rice	Italy	EUR	corn	Spain
EUR	rice	Italy	EUR	beets	Belgium
EUR	rice	Italy	ASIA	rice	Vietnam
AFR	oil	Egypt	NA	sugar	USA
AFR	oil	Egypt	EUR	corn	Spain
AFR	oil	Egypt	EUR	beets	Belgium
AFR	oil	Egypt	ASIA	rice	Vietnam

The cross join is not functionally different from a Cartesian product join. You would get the same result by submitting the following program:

```
proc sql;
   select *
      from lefttab, righttab;
```

Do not use an ON clause with a cross join. An ON clause will cause a cross join to fail. However, you can use a WHERE clause to subset the output.

Union Joins

A union join returns a union of the columns of both tables. The union join places in the results all rows with their respective column values from each input table. Columns that do not exist in one table will have null (missing) values for those rows in the result table. The following example demonstrates a union join.

```
proc sql;
   title 'Union Join';
   select *
      from lefttab union join righttab;
```

Union Join

Continent	Export	Country	Continent	Export	Country
			NA	sugar	USA
			EUR	corn	Spain
			EUR	beets	Belgium
			ASIA	rice	Vietnam
NA	wheat	Canada			
EUR	corn	France			
EUR	rice	Italy			
AFR	oil	Egypt			

Using a union join is similar to concatenating tables with the OUTER UNION set operator. See "query-expression" on page 1093 for more information.

Do not use an ON clause with a union join. An ON clause will cause a union join to fail.

Natural Joins

A natural join selects rows from two tables that have equal values in columns that share the same name and the same type. An error results if two columns have the same name but different types. If *join-specification* is omitted when specifying a natural join, then INNER is implied. If no like columns are found, then a cross join is performed.

The following examples use these two tables:

table1

x	y	z
1	2	3
2	1	8
6	5	4
2	5	6

```
                         table2

                x         b         z
          ----------------------------
                1         5         3
                3         5         4
                2         7         8
                6         0         4
```

The following program demonstrates a natural inner join.

```
proc sql;
   title 'Natural Inner Join';
   select *
   from table1 natural join table2;
```

```
                    Natural Inner Join

           x         z         b         y
     --------------------------------------
           1         3         5         2
           2         8         7         1
           6         4         0         5
```

The following program demonstrates a natural left outer join.

```
proc sql;
   title 'Natural Left Outer Join';
   select *
      from table1 natural left join table2;
```

```
                 Natural Left Outer Join

           x         z         b         y
     --------------------------------------
           1         3         5         2
           2         6         .         5
           2         8         7         1
           6         4         0         5
```

Do not use an ON clause with a natural join. An ON clause will cause a natural join to fail. When using a natural join, an ON clause is implied, matching all like columns.

Joining More Than Two Tables

Inner joins are usually performed on two or three tables, but they can be performed on up to 32 tables in PROC SQL. A join on three tables is described here to explain how and why the relationships work among the tables.

In a three-way join, the sql-expression consists of two conditions: one relates the first table to the second table and the other relates the second table to the third table. It is possible to break this example into stages, performing a two-way join into a temporary table and then joining that table with the third one for the same result. However, PROC SQL can do it all in one step as shown in the next example.

The example shows the joining of three tables: COMM, PRICE, and AMOUNT. To calculate the total revenue from exports for each country, you need to multiply the amount exported (AMOUNT table) by the price of each unit (PRICE table), and you must know the commodity that each country exports (COMM table).

```
                    COMM Table

          Continent  Export    Country
          -----------------------------
          NA         wheat     Canada
          EUR        corn      France
          EUR        rice      Italy
          AFR        oil       Egypt
```

```
                    PRICE Table

                Export       Price
                ------------------
                rice          3.56
                corn          3.45
                oil             18
                wheat         2.98
```

```
                    AMOUNT Table

                Country   Quantity
                ------------------
                Canada       16000
                France        2400
                Italy          500
                Egypt        10000
```

```
proc sql;
   title  'Total Export Revenue';
   select c.Country, p.Export, p.Price,
          a.Quantity,a.quantity*p.price
          as Total
      from comm c, price p, amount a
      where c.export=p.export
            and c.country=a.country;
```

```
                 Total Export Revenue

  Country   Export       Price  Quantity     Total
  ------------------------------------------------
  Italy     rice          3.56       500      1780
  France    corn          3.45      2400      8280
  Egypt     oil             18     10000    180000
  Canada    wheat         2.98     16000     47680
```

See Example 9 on page 1145 for another example.

Comparison of Joins and Subqueries

You can often use a subquery or a join to get the same result. However, it is often more efficient to use a join if the outer query and the subquery do not return duplicate rows. For example, the following queries produce the same result. The second query is more efficient:

```
proc sql;
   select IDNumber, Birth
      from proclib.payroll
      where IDNumber in (select idnum
                      from proclib.staff
                      where lname like 'B%');

proc sql;
   select  p.IDNumber, p.Birth
     from proclib.payroll p, proclib.staff s
     where p.idnumber=s.idnum
           and s.lname like 'B%';
```

Note: PROCLIB.PAYROLL is shown in Example 2 on page 1127. △

LIKE condition

Tests for a matching pattern.

sql-expression <NOT> **LIKE** sql-expression <ESCAPE *character-expression*>

Arguments

sql-expression
is described in “sql-expression” on page 1099.

character-expression
is an sql-expression that evaluates to a single character. The operands of *character-expression* must be character or string literals; they cannot be column names.

Note: If you use an ESCAPE clause, then the pattern-matching specification must be a quoted string or quoted concatenated string; it cannot contain column names. △

Details

The LIKE condition selects rows by comparing character strings with a pattern-matching specification. It resolves to true and displays the matched string(s) if the left operand matches the pattern specified by the right operand.

The ESCAPE clause is used to search for literal instances of the percent (%) and underscore (_) characters, which are usually used for pattern matching.

Patterns for Searching

Patterns are composed of three classes of characters:

underscore (_)
matches any single character.

percent sign (%)
matches any sequence of zero or more characters.

any other character
matches that character.

These patterns can appear before, after, or on both sides of characters that you want to match. The LIKE condition is case-sensitive.

The following list uses these values: **Smith**, **Smooth**, **Smothers**, **Smart**, and **Smuggle**.

'Sm%'
matches **Smith**, **Smooth**, **Smothers**, **Smart**, **Smuggle**.

'%th'
matches **Smith**, **Smooth**.

'S__gg%'
matches **Smuggle**.

'S_o'
matches a three-letter word, so it has no matches here.

'S_o%'
matches **Smooth**, **Smothers**.

'S%th'
matches **Smith**, **Smooth**.

'Z'
matches the single, uppercase character **Z** only, so it has no matches here.

Searching for Literal % and _

Because the % and _ characters have special meaning in the context of the LIKE condition, you must use the ESCAPE clause to search for these character literals in the input character string.

These example use the values **app**, **a_%**, **a__**, **bbaa1**, and **ba_1**.

- The condition **like 'a_%'** matches **app**, **a_%**, and **a__**, because the underscore (_) in the search pattern matches any single character (including the underscore), and the percent (%) in the search pattern matches zero or more characters, including '%' and '_'.
- The condition **like 'a_^%' escape '^'** matches only **a_%**, because the escape character (^) specifies that the pattern search for a literal '%'.
- The condition **like 'a_%' escape '_'** matches none of the values, because the escape character (_) specifies that the pattern search for an 'a' followed by a literal '%', which does not apply to any of these values.

Searching for Mixed-Case Strings

To search for mixed-case strings, use the UPCASE function to make all the names uppercase before entering the LIKE condition:

```
upcase(name) like 'SM%';
```

Note: When you are using the % character, be aware of the effect of trailing blanks. You may have to use the TRIM function to remove trailing blanks in order to match values. △

LOWER function

Converts the case of a character string to lowercase.

See also: “UPPER function” on page 1114

LOWER (sql-expression)

Argument

sql-expression
must resolve to a character string and is described in “sql-expression” on page 1099.

Details

The LOWER function operates on character strings. LOWER changes the case of its argument to all lowercase.

Note: The LOWER function is provided for compatibility with the ANSI SQL standard. You can also use the SAS function LOWCASE. △

query-expression

Retrieves data from tables.

See also: “table-expression” on page 1113, “Query Expressions (Subqueries)” on page 1102, and “In-Line Views” on page 1064

table-expression <*set-operator* table-expression> <...*set-operator* table-expression>

Arguments

table-expression
is described in “table-expression” on page 1113.

set-operator
is one of the following:

INTERSECT <CORRESPONDING> <ALL>

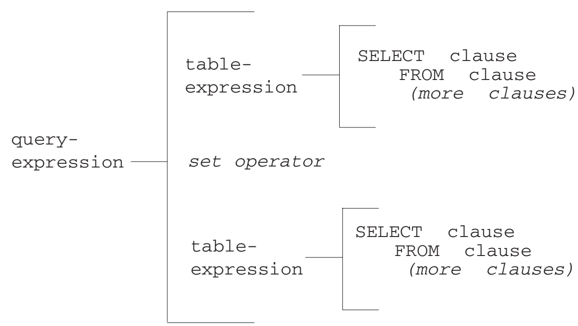
OUTER UNION <CORRESPONDING>

UNION <CORRESPONDING> <ALL>

EXCEPT <CORRESPONDING> <ALL>

Query Expressions and Table Expressions

A query-expression is one or more table-expressions. Multiple table expressions are linked by set operators. The following figure illustrates the relationship between table-expressions and query-expressions.



Set Operators

PROC SQL provides these set operators:

OUTER UNION
concatenates the query results.

UNION
produces all unique rows from both queries.

EXCEPT
produces rows that are part of the first query only.

INTERSECT
produces rows that are common to both query results.

A query-expression with set operators is evaluated as follows.

- Each table-expression is evaluated to produce an (internal) intermediate result table.
- Each intermediate result table then becomes an operand linked with a set operator to form an expression, for example, A UNION B.
- If the query-expression involves more than two table-expressions, then the result from the first two becomes an operand for the next set operator and operand, such as (A UNION B) EXCEPT C, ((A UNION B) EXCEPT C) INTERSECT D, and so on.
- Evaluating a query-expression produces a single output table.

Set operators follow this order of precedence unless they are overridden by parentheses in the expression(s): INTERSECT is evaluated first. OUTER UNION, UNION, and EXCEPT have the same level of precedence.

PROC SQL performs set operations even if the tables or views that are referred to in the table-expressions do not have the same number of columns. The reason for this behavior is that the ANSI Standard for SQL requires that tables or views that are involved in a set operation have the same number of columns and that the columns have matching data types. If a set operation is performed on a table or view that has fewer columns than the one(s) with which it is being linked, then PROC SQL extends the table or view with fewer columns by creating columns with missing values of the appropriate data type. This temporary alteration enables the set operation to be performed correctly.

CORRESPONDING (CORR) Keyword

The CORRESPONDING keyword is used only when a set operator is specified. CORR causes PROC SQL to match the columns in table-expressions *by name* and not by ordinal position. Columns that do not match by name are excluded from the result table, except for the OUTER UNION operator. See “OUTER UNION” on page 1095.

For example, when performing a set operation on two table-expressions, PROC SQL matches the first specified column-name (listed in the SELECT clause) from one table-expression with the first specified column-name from the other. If CORR is omitted, then PROC SQL matches the columns by ordinal position.

ALL Keyword

The set operators automatically eliminate duplicate rows from their output tables. The optional ALL keyword preserves the duplicate rows, reduces the execution by one step, and thereby improves the query-expression’s performance. You use it when you want to display all the rows resulting from the table-expressions, rather than just the unique rows. The ALL keyword is used only when a set operator is also specified.

OUTER UNION

Performing an OUTER UNION is very similar to performing the SAS DATA step with a SET statement. The OUTER UNION concatenates the intermediate results from the table-expressions. Thus, the result table for the query-expression contains all the rows produced by the first table-expression followed by all the rows produced by the second table-expression. Columns with the same name are in separate columns in the result table.

For example, the following query expression concatenates the ME1 and ME2 tables but does not overlay like-named columns. Output 44.1 shows the result.

```
                         ME1

IDnum     Jobcode      Salary     Bonus
---------------------------------------
1400      ME1           29769       587
1403      ME1           28072       342
1120      ME1           28619       986
1120      ME1           28619       986
```

```
                       ME2

           IDnum     Jobcode     Salary
           ----------------------------
           1653      ME2          35108
           1782      ME2          35345
           1244      ME2          36925
```

```
proc sql;
   title 'ME1 and ME2: OUTER UNION';
   select *
      from me1
   outer union
   select *
      from me2;
```

Output 44.1 OUTER UNION of ME1 and ME2 Tables

```
                ME1 and ME2: OUTER UNION

IDnum     Jobcode     Salary     Bonus  IDnum     Jobcode     Salary
--------------------------------------------------------------------
1400      ME1          29769       587                             .
1403      ME1          28072       342                             .
1120      ME1          28619       986                             .
1120      ME1          28619       986                             .
                           .         .  1653      ME2          35108
                           .         .  1782      ME2          35345
                           .         .  1244      ME2          36925
```

Concatenating tables with the OUTER UNION set operator is similar to performing a union join. See “Union Joins” on page 1088 for more information.

To overlay columns with the same name, use the CORRESPONDING keyword.

```
proc sql;
   title 'ME1 and ME2: OUTER UNION CORRESPONDING';
   select *
      from me1
   outer union corr
   select *
      from me2;
```

```
              ME1 and ME2: OUTER UNION CORRESPONDING

              IDnum     Jobcode     Salary     Bonus
              --------------------------------------
              1400      ME1          29769       587
              1403      ME1          28072       342
              1120      ME1          28619       986
              1120      ME1          28619       986
              1653      ME2          35108         .
              1782      ME2          35345         .
              1244      ME2          36925         .
```

In the resulting concatenated table, notice the following:

- OUTER UNION CORRESPONDING retains all nonmatching columns.
- For columns with the same name, if a value is missing from the result of the first table-expression, then the value in that column from the second table-expression is inserted.
- The ALL keyword is not used with OUTER UNION because this operator's default action is to include all rows in a result table. Thus, both rows from the table ME1 where IDnum is **1120** appear in the output.

UNION

The UNION operator produces a table that contains all the unique rows that result from both table-expressions. That is, the output table contains rows produced by the first table-expression, the second table-expression, or both.

Columns are appended by position in the tables, regardless of the column names. However, the data type of the corresponding columns must match or the union will not occur. PROC SQL issues a warning message and stops executing.

The names of the columns in the output table are the names of the columns from the first table-expression unless a column (such as an expression) has no name in the first table-expression. In such a case, the name of that column in the output table is the name of the respective column in the second table-expression.

In the following example, PROC SQL combines the two tables:

```
proc sql;
   title 'ME1 and ME2: UNION';
   select *
      from me1
   union
   select *
      from me2;
```

```
                        ME1 and ME2: UNION

              IDnum     Jobcode     Salary     Bonus
              --------------------------------------
              1120      ME1          28619       986
              1244      ME2          36925         .
              1400      ME1          29769       587
              1403      ME1          28072       342
              1653      ME2          35108         .
              1782      ME2          35345         .
```

In the following example, ALL includes the duplicate row from ME1. In addition, ALL changes the sorting by specifying that PROC SQL make one pass only. Thus, the values from ME2 are simply appended to the values from ME1.

```
proc sql;
   title 'ME1 and ME2: UNION ALL';
   select *
      from me1
   union all
   select *
      from me2;
```

```
                    ME1 and ME2: UNION ALL

             IDnum     Jobcode     Salary     Bonus
             --------------------------------------
             1400      ME1          29769       587
             1403      ME1          28072       342
             1120      ME1          28619       986
             1120      ME1          28619       986
             1653      ME2          35108         .
             1782      ME2          35345         .
             1244      ME2          36925         .
```

See Example 5 on page 1134 for another example.

EXCEPT

The EXCEPT operator produces (from the first table-expression) an output table that has unique rows that are not in the second table-expression. If the intermediate result from the first table-expression has at least one occurrence of a row that is not in the intermediate result of the second table-expression, then that row (from the first table-expression) is included in the result table.

In the following example, the IN_USA table contains flights to cities within and outside the USA. The OUT_USA table contains flights only to cities outside the USA. This example returns only the rows from IN_USA that are not also in OUT_USA:

```
proc sql;
    title 'Flights from IN_USA Only';
    select * from in_usa
    except
    select * from out_usa;
```

```
                          IN_USA

                     Flight    Dest
                     ------------------
                     145       ORD
                     156       WAS
                     188       LAX
                     193       FRA
                     207       LON
```

```
                    OUT_USA

               Flight    Dest
               ------------------
               193       FRA
               207       LON
               311       SJA
```

```
            Flights from IN_USA Only

               Flight    Dest
               ------------------
               145       ORD
               156       WAS
               188       LAX
```

INTERSECT

The INTERSECT operator produces an output table that has rows that are common to both tables. For example, using the IN_USA and OUT_USA tables shown above, the following example returns rows that are in both tables:

```
proc sql;
    title 'Flights from Both IN_USA and OUT_USA';
    select * from in_usa
    intersect
    select * from out_usa;
```

```
      Flights from Both IN_USA and OUT_USA

               Flight    Dest
               ------------------
               193       FRA
               207       LON
```

sql-expression

Produces a value from a sequence of operands and operators.

operand operator operand

Arguments

operand

is one of the following:

- a *constant*, which is a number or a quoted character string (or other special notation) that indicates a fixed value. Constants are also called *literals*. Constants are described in *SAS Language Reference: Dictionary*.

- a column-name, which is described in “column-name” on page 1078.
- a CASE expression, which is described in “CASE expression” on page 1073.
- a SAS function, which is any SAS function except LAG, DIF, and SOUND. Functions are described in *SAS Language Reference: Dictionary*.
- the ANSI SQL functions COALESCE, BTRIM, LOWER, UPPER, and SUBSTRING.
- a summary-function, which is described in “summary-function” on page 1107.
- a query-expression, which is described in “query-expression” on page 1093.
- the USER literal, which references the userid of the person who submitted the program. The userid that is returned is operating environment-dependent, but PROC SQL uses the same value that the &SYSJOBID macro variable has on the operating environment.

operator
is described in “Operators and the Order of Evaluation” on page 1100.

Note: SAS functions, including summary functions, can stand alone as SQL expressions. For example

```
select min(x) from table;

select scan(y,4) from table;
```

△

SAS Functions

PROC SQL supports the same SAS functions as the DATA step, except for the functions LAG, DIF, and SOUND. For example, the SCAN function is used in the following query:

```
select style, scan(street,1) format=$15.
   from houses;
```

See *SAS Language Reference: Dictionary* for complete documentation on SAS functions. Summary functions are also SAS functions. See “summary-function” on page 1107 for more information.

USER Literal

USER can be specified in a view definition, for example, to create a view that restricts access to those in the user’s department. Note that the USER literal value is stored in uppercase, so it is advisable to use the UPCASE function when comparing to this value:

```
create view myemp as
   select * from dept12.employees
      where upcase(manager)=user;
```

This view produces a different set of employee information for each manager who references it.

Operators and the Order of Evaluation

The order in which operations are evaluated is the same as in the DATA step with this one exception: NOT is grouped with the logical operators AND and OR in PROC SQL; in the DATA step, NOT is grouped with the unary plus and minus signs.

Unlike missing values in some versions of SQL, missing values in SAS always appear first in the collating sequence. Therefore, in Boolean and comparison operations, the following expressions resolve to true in a predicate:

```
 3>null
-3>null
 0>null
```

You can use parentheses to group values or to nest mathematical expressions. Parentheses make expressions easier to read and can also be used to change the order of evaluation of the operators. Evaluating expressions with parentheses begins at the deepest level of parentheses and moves outward. For example, SAS evaluates A+B*C as A+(B*C), although you can add parentheses to make it evaluate as (A+B)*C for a different result.

Higher priority operations are performed first: that is, group 0 operators are evaluated before group 5 operators. The following table shows the operators and their order of evaluation, including their priority groups.

Table 44.1 Operators and Order of Evaluation

<table>
<tr><th>Group</th><th>Operator</th><th>Description</th></tr>
<tr><td>0</td><td>( )</td><td>forces the expression enclosed to be evaluated first</td></tr>
<tr><td>1</td><td>case-expression</td><td>selects result values that satisfy specified conditions</td></tr>
<tr><td>2</td><td>**</td><td>raises to a power</td></tr>
<tr><td></td><td>unary +, unary -</td><td>indicates a positive or negative number</td></tr>
<tr><td>3</td><td>*</td><td>multiplies</td></tr>
<tr><td></td><td>/</td><td>divides</td></tr>
<tr><td>4</td><td>+</td><td>adds</td></tr>
<tr><td></td><td>–</td><td>subtracts</td></tr>
<tr><td>5</td><td>||</td><td>concatenates</td></tr>
<tr><td>6</td><td><NOT> BETWEEN condition</td><td>See “BETWEEN condition” on page 1071.</td></tr>
<tr><td></td><td><NOT> CONTAINS condition</td><td>see “CONTAINS condition” on page 1079.</td></tr>
<tr><td></td><td><NOT> EXISTS condition</td><td>See “EXISTS condition” on page 1080.</td></tr>
<tr><td></td><td><NOT> IN condition</td><td>See “IN condition” on page 1080.</td></tr>
<tr><td></td><td>IS <NOT> condition</td><td>See “IS condition” on page 1081.</td></tr>
<tr><td></td><td><NOT> LIKE condition</td><td>See “LIKE condition” on page 1091.</td></tr>
<tr><td>7</td><td>=, eq</td><td>equals</td></tr>
<tr><td></td><td>¬=, ^=, < >, ne</td><td>does not equal</td></tr>
<tr><td></td><td>>, gt</td><td>is greater than</td></tr>
<tr><td></td><td><, lt</td><td>is less than</td></tr>
<tr><td></td><td>>=, ge</td><td>is greater than or equal to</td></tr>
<tr><td></td><td><=, le</td><td>is less than or equal to</td></tr>
<tr><td></td><td>=*</td><td>sounds like (use with character operands only). See Example 11 on page 1150.</td></tr>
</table>

Group	Operator	Description
	eqt	equal to truncated strings (use with character operands only). See “Truncated String Comparison Operators” on page 1102.
	gtt	greater than truncated strings
	ltt	less than truncated strings
	get	greater than or equal to truncated strings
	let	less than or equal to truncated strings
	net	not equal to truncated strings
8	&, AND	indicates logical AND
9	|, OR	indicates logical OR
10	¬, ^, NOT	indicates logical NOT

Symbols for operators might vary, depending on your operating environment. See *SAS Language Reference: Dictionary* for more information on operators and expressions.

Truncated String Comparison Operators

PROC SQL supports truncated string comparison operators (see Group 7 in Table 44.1 on page 1101). In a truncated string comparison, the comparison is performed after making the strings the same length by truncating the longer string to be the same length as the shorter string. For example, the expression **`'TWOSTORY' eqt 'TWO'`** is true because the string ’TWOSTORY’ is reduced to ’TWO’ before the comparison is performed. Note that the truncation is performed internally; neither operand is permanently changed.

Note: Unlike the DATA step, PROC SQL does not support the colon operators (such as =:, >:, and <=:) for truncated string comparisons. Use the alphabetic operators (such as EQT, GTT, and LET). △

Query Expressions (Subqueries)

A query-expression is called a *subquery* when it is used in a WHERE or HAVING clause. A subquery is a query-expression that is nested as part of another query-expression. A subquery selects one or more rows from a table based on values in another table.

Depending on the clause that contains it, a subquery can return a single value or multiple values. If more than one subquery is used in a query-expression, then the innermost query is evaluated first, then the next innermost query, and so on, moving outward.

PROC SQL allows a subquery (contained in parentheses) at any point in an expression where a simple column value or constant can be used. In this case, a subquery must return a *single value*, that is, one row with only one column.

The following is an example of a subquery that returns one value. This PROC SQL step subsets the PROCLIB.PAYROLL table based on information in the PROCLIB.STAFF table. (PROCLIB.PAYROLL is shown in Example 2 on page 1127, and PROCLIB.STAFF is shown in Example 4 on page 1131.) PROCLIB.PAYROLL contains employee identification numbers (IdNumber) and their salaries (Salary) but does not contain their names. If you want to return only the row from PROCLIB.PAYROLL for one employee, then you can use a subquery that queries the PROCLIB.STAFF table, which contains the employees’ identification numbers and their names (Lname and Fname).

```
options ls=64 nodate nonumber;
proc sql;
    title 'Information for Earl Bowden';
    select *
       from proclib.payroll
       where idnumber=
             (select idnum
             from proclib.staff
             where upcase(lname)='BOWDEN');
```

```
                 Information for Earl Bowden

     Id
     Number  Gender  Jobcode    Salary    Birth    Hired
     -------------------------------------------------
     1403    M       ME1         28072  28JAN69  21DEC91
```

Subqueries can return *multiple values*. The following example uses the tables PROCLIB.DELAY and PROCLIB.MARCH. These tables contain information about the same flights and have the Flight column in common. The following subquery returns all the values for Flight in PROCLIB.DELAY for international flights. The values from the subquery complete the WHERE clause in the outer query. Thus, when the outer query is executed, only the international flights from PROCLIB.MARCH are in the output.

```
options ls=64 nodate nonumber;
proc sql outobs=5;
   title 'International Flights from';
   title2 'PROCLIB.MARCH';
   select Flight, Date, Dest, Boarded
      from proclib.march
      where flight in
            (select flight
             from proclib.delay
             where destype='International');
```

```
                  International Flights from
                        PROCLIB.MARCH

               Flight     Date  Dest   Boarded
               -------------------------------
               219     01MAR94  LON        198
               622     01MAR94  FRA        207
               132     01MAR94  YYZ        115
               271     01MAR94  PAR        138
               219     02MAR94  LON        147
```

Sometimes it is helpful to compare a value with a set of values returned by a subquery. The keywords ANY or ALL can be specified before a subquery when the subquery is the right-hand operand of a comparison. If ALL is specified, then the comparison is true only if it is true for all values that are returned by the subquery. If a subquery returns no rows, then the result of an ALL comparison is true for each row of the outer query.

If ANY is specified, then the comparison is true if it is true for any one of the values that are returned by the subquery. If a subquery returns no rows, then the result of an ANY comparison is false for each row of the outer query.

The following example selects all those in PROCLIB.PAYROLL who earn more than the highest paid **ME3**:

```
options ls=64 nodate nonumber ;
proc sql;
title ''Employees who Earn More than'';
title2 ''All ME's'';
   select *
     from proclib.payroll
     where salary > all (select salary
                         from proclib.payroll
                         where jobcode='ME3');
```

```
                Employees who Earn More than
                         All ME's

       Id
       Number  Gender  Jobcode    Salary    Birth    Hired
       ---------------------------------------------------
       1333    M       PT2         88606  30MAR61  10FEB81
       1739    M       PT1         66517  25DEC64  27JAN91
       1428    F       PT1         68767  04APR60  16NOV91
       1404    M       PT2         91376  24FEB53  01JAN80
       1935    F       NA2         51081  28MAR54  16OCT81
       1905    M       PT1         65111  16APR72  29MAY92
       1407    M       PT1         68096  23MAR69  18MAR90
       1410    M       PT2         84685  03MAY67  07NOV86
       1439    F       PT1         70736  06MAR64  10SEP90
       1545    M       PT1         66130  12AUG59  29MAY90
       1106    M       PT2         89632  06NOV57  16AUG84
       1442    F       PT2         84536  05SEP66  12APR88
       1417    M       NA2         52270  27JUN64  07MAR89
       1478    M       PT2         84203  09AUG59  24OCT90
       1556    M       PT1         71349  22JUN64  11DEC91
       1352    M       NA2         53798  02DEC60  16OCT86
       1890    M       PT2         91908  20JUL51  25NOV79
       1107    M       PT2         89977  09JUN54  10FEB79
       1830    F       PT2         84471  27MAY57  29JAN83
       1928    M       PT2         89858  16SEP54  13JUL90
       1076    M       PT1         66558  14OCT55  03OCT91
```

Note: See the first item in “Subqueries and Efficiency” on page 1105 for a note about efficiency when using ALL. △

In order to visually separate a subquery from the rest of the query, you can enclose the subquery in any number of pairs of parentheses.

Correlated Subqueries

In a correlated subquery, the WHERE expression in a subquery refers to values in a table in the outer query. The correlated subquery is evaluated for each row in the outer query. With correlated subqueries, PROC SQL executes the subquery and the outer query together.

The following example uses the PROCLIB.DELAY and PROCLIB.MARCH tables. A DATA step (“PROCLIB.DELAY” on page 1390) creates PROCLIB.DELAY. PROCLIB.MARCH is shown in Example 13 on page 1154. PROCLIB.DELAY has the Flight, Date, Orig, and Dest columns in common with PROCLIB.MARCH:

```
proc sql outobs=5;
   title 'International Flights';
   select *
      from proclib.march
      where 'International' in
           (select destype
            from proclib.delay
            where march.Flight=delay.Flight);
```

The subquery resolves by substituting every value for MARCH.Flight into the subquery’s WHERE clause, one row at a time. For example, when MARCH.Flight=**219**, the subquery resolves as follows:

1 PROC SQL retrieves all the rows from DELAY where Flight=**219** and passes their DESTYPE values to the WHERE clause.
2 PROC SQL uses the DESTYPE values to complete the WHERE clause:

```
where 'International' in
   ('International','International', ...)
```

3 The WHERE clause checks to see if **International** is in the list. Because it is, all rows from MARCH that have a value of **219** for Flight become part of the output.

The following output contains the rows from MARCH for international flights only.

Output 44.2 International Flights for March

```
                              International Flights

        Flight     Date  Depart  Orig  Dest     Miles   Boarded  Capacity
        -----------------------------------------------------------------
        219     01MAR94    9:31  LGA   LON       3442       198       250
        622     01MAR94   12:19  LGA   FRA       3857       207       250
        132     01MAR94   15:35  LGA   YYZ        366       115       178
        271     01MAR94   13:17  LGA   PAR       3635       138       250
        219     02MAR94    9:31  LGA   LON       3442       147       250
```

Subqueries and Efficiency

- Use the MAX function in a subquery instead of the ALL keyword before the subquery. For example, the following queries produce the same result, but the second query is more efficient:

```
proc sql;
   select * from proclib.payroll
   where salary> all(select salary
                     from proclib.payroll
                     where jobcode='ME3');
```

```
proc sql;
   select * from proclib.payroll
   where salary> (select max(salary)
                  from proclib.payroll
                  where jobcode='ME3');
```

- With subqueries, use IN instead of EXISTS when possible. For example, the following queries produce the same result, but the second query is usually more efficient:

```
proc sql;
   select *
      from proclib.payroll p
      where exists (select *
                    from staff s
                    where p.idnum=s.idnum
                       and state='CT');

proc sql;
   select *
      from proclib.payroll
      where idnum in (select idnum
                        from staff
                        where state='CT');
```

SUBSTRING function

Returns a part of a character expression.

SUBSTRING (sql-expression FROM *start* <FOR *length*>)

- sql-expression must be a character string and is described in “sql-expression” on page 1099.
- *start* is a number (not a variable or column name) that specifies the position, counting from the left end of the character string, at which to begin extracting the substring.
- *length* is a number (not a variable or column name) that specifies the length of the substring that is to be extracted.

Details

The SUBSTRING function operates on character strings. SUBSTRING returns a specified part of the input character string, beginning at the position that is specified by *start*. If *length* is omitted, then the SUBSTRING function returns all characters from *start* to the end of the input character string. The values of *start* and *length* must be numbers (not variables) and can be positive, negative, or zero.

If *start* is greater than the length of the input character string, then the SUBSTRING function returns a zero-length string.

If *start* is less than 1, then the SUBSTRING function begins extraction at the beginning of the input character string.

If *length* is specified, then the sum of *start* and *length* cannot be less than *start* or an error is returned. If the sum of *start* and *length* is greater than the length of the input character string, then the SUBSTRING function returns all characters from *start* to the end of the input character string. If the sum of *start* and *length* is less than 1, then the SUBSTRING function returns a zero-length string.

Note: The SUBSTRING function is provided for compatibility with the ANSI SQL standard. You can also use the SAS function SUBSTR. △

summary-function

Performs statistical summary calculations.

Restriction: A summary function cannot appear in an ON clause or a WHERE clause.

See also: GROUP BY on page 1065, HAVING Clause on page 1067, SELECT Clause on page 1058, and "table-expression" on page 1113

Featured in: Example 8 on page 1143, Example 12 on page 1152, and Example 15 on page 1160

summary-function (<DISTINCT | ALL> sql-expression)

Arguments

summary-function
is one of the following:

AVG|MEAN
arithmetic mean or average of values

COUNT|FREQ|N
number of nonmissing values

CSS
corrected sum of squares

CV
coefficient of variation (percent)

MAX
largest value

MIN
smallest value

NMISS
number of missing values

PRT
probability of a greater absolute value of Student's *t*

RANGE
range of values

STD
standard deviation

STDERR
standard error of the mean

SUM
sum of values

SUMWGT
sum of the WEIGHT variable values*

T
Student's *t* value for testing the hypothesis that the population mean is zero

USS
uncorrected sum of squares

VAR
variance

For a description and the formulas used for these statistics, see Appendix 1, "SAS Elementary Statistics Procedures," on page 1339.

DISTINCT
specifies that only the unique values of sql-expression be used in the calculation.

ALL
specifies that all values of sql-expression be used in the calculation. If neither DISTINCT nor ALL is specified, then ALL is used.

sql-expression
is described in "sql-expression" on page 1099.

Summarizing Data

Summary functions produce a statistical summary of the entire table or view that is listed in the FROM clause or for each group that is specified in a GROUP BY clause. If GROUP BY is omitted, then all the rows in the table or view are considered to be a single group. These functions reduce all the values in each row or column in a table to one *summarizing* or *aggregate* value. For this reason, these functions are often called *aggregate functions*. For example, the sum (one value) of a column results from the addition of all the values in the column.

Counting Rows

The COUNT function counts rows. COUNT(*) returns the total number of rows in a group or in a table. If you use a column name as an argument to COUNT, then the result is the total number of rows in a group or in a table that have a nonmissing value for that column. If you want to count the unique values in a column, then specify COUNT(DISTINCT *column*).

* Currently, there is no way to designate a WEIGHT variable for a table in PROC SQL. Thus, each row (or observation) has a weight of 1.

If the SELECT clause of a table-expression contains one or more summary functions and that table-expression resolves to no rows, then the summary function results are missing values. The following are exceptions that return zeros:

COUNT(*)

COUNT(<DISTINCT> sql-expression)

NMISS(<DISTINCT> sql-expression)

See Example 8 on page 1143 and Example 15 on page 1160 for examples.

Calculating Statistics Based on the Number of Arguments

The number of arguments that is specified in a summary function affects how the calculation is performed. If you specify a single argument, then the values in the column are calculated. If you specify multiple arguments, then the arguments or columns that are listed are calculated for each row. For example, consider calculations on the following table.

```
proc sql;
   title 'Summary Table';
   select * from summary;
```

```
                         Summary Table

                         X         Y         Z
                  ----------------------------
                         1         3         4
                         2         4         5
                         8         9         4
                         4         5         4
```

If you use one argument in the function, then the calculation is performed on that column only. If you use more than one argument, then the calculation is performed on each row of the specified columns. In the following PROC SQL step, the MIN and MAX functions return the minimum and maximum of the columns they are used with. The SUM function returns the sum of each row of the columns specified as arguments:

```
proc sql;
    select min(x) as Colmin_x,
           min(y) as Colmin_y,
           max(z) as Colmax_z,
           sum(x,y,z) as Rowsum
       from summary;
```

```
                         Summary Table

             Colmin_x  Colmin_y  Colmax_z    Rowsum
             --------------------------------------
                    1         3         5         8
                    1         3         5        11
                    1         3         5        21
                    1         3         5        13
```

Remerging Data

When you use a summary function in a SELECT clause or a HAVING clause, you might see the following message in the SAS log:

```
NOTE: The query requires remerging summary
      statistics back with the original
      data.
```

The process of *remerging* involves two passes through the data. On the first pass, PROC SQL

- calculates and returns the value of summary functions. It then uses the result to calculate the arithmetic expressions in which the summary function participates.
- groups data according to the GROUP BY clause.

On the second pass, PROC SQL retrieves any additional columns and rows that it needs to show in the output.

The following examples use the PROCLIB.PAYROLL table (shown in Example 2 on page 1127) to show when remerging of data is and is not necessary.

The first query requires remerging. The first pass through the data groups the data by Jobcode and resolves the AVG function for each group. However, PROC SQL must make a second pass in order to retrieve the values of IdNumber and Salary.

```
proc sql outobs=10;
   title 'Salary Information';
   title2 '(First 10 Rows Only)';
   select  IdNumber, Jobcode, Salary,
           avg(salary) as AvgSalary
      from proclib.payroll
      group by jobcode;
```

```
                    Salary Information
                   (First 10 Rows Only)

            Id
            Number  Jobcode    Salary  AvgSalary
            ------------------------------------
            1704    BCK         25465   25794.22
            1677    BCK         26007   25794.22
            1383    BCK         25823   25794.22
            1845    BCK         25996   25794.22
            1100    BCK         25004   25794.22
            1663    BCK         26452   25794.22
            1673    BCK         25477   25794.22
            1389    BCK         25028   25794.22
            1834    BCK         26896   25794.22
            1132    FA1         22413   23039.36
```

You can change the previous query to return only the average salary for each jobcode. The following query does not require remerging because the first pass of the data does the summarizing and the grouping. A second pass is not necessary.

```
proc sql outobs=10;
   title 'Average Salary for Each Jobcode';
   select Jobcode, avg(salary) as AvgSalary
   from proclib.payroll
   group by jobcode;
```

```
                Average Salary for Each Jobcode

                       Jobcode  AvgSalary
                       ------------------
                       BCK       25794.22
                       FA1       23039.36
                       FA2       27986.88
                       FA3       32933.86
                       ME1       28500.25
                       ME2       35576.86
                       ME3       42410.71
                       NA1        42032.2
                       NA2          52383
                       PT1          67908
```

When you use the HAVING clause, PROC SQL may have to remerge data to resolve the HAVING expression.

First, consider a query that uses HAVING but that does not require remerging. The query groups the data by values of Jobcode, and the result contains one row for each value of Jobcode and summary information for people in each Jobcode. On the first pass, the summary functions provide values for the **Number**, **Average Age**, and **Average Salary** columns. The first pass provides everything that PROC SQL needs to resolve the HAVING clause, so no remerging is necessary.

```
proc sql outobs=10;
title 'Summary Information for Each Jobcode';
title2 '(First 10 Rows Only)';
   select Jobcode,
          count(jobcode) as number
              label='Number',
          avg(int((today()-birth)/365.25))
              as avgage format=2.
              label='Average Age',
          avg(salary) as avgsal format=dollar8.
              label='Average Salary'
      from proclib.payroll
      group by jobcode
      having avgage ge 30;
```

```
                   Summary Information for Each Jobcode
                           (First 10 Rows Only)

                                      Average   Average
                   Jobcode    Number      Age    Salary
                   ------------------------------------
                   BCK             9       36   $25,794
                   FA1            11       33   $23,039
                   FA2            16       37   $27,987
                   FA3             7       39   $32,934
                   ME1             8       34   $28,500
                   ME2            14       39   $35,577
                   ME3             7       42   $42,411
                   NA1             5       30   $42,032
                   NA2             3       42   $52,383
                   PT1             8       38   $67,908
```

In the following query, PROC SQL remerges the data because the HAVING clause uses the SALARY column in the comparison and SALARY is not in the GROUP BY clause.

```
proc sql outobs=10;
title 'Employees who Earn More than the';
title2 'Average for Their Jobcode';
title3 '(First 10 Rows Only)';
   select Jobcode, Salary,
          avg(salary) as AvgSalary
      from proclib.payroll
      group by jobcode
      having salary > AvgSalary;
```

```
                    Employees who Earn More than the
                       Average for Their Jobcode
                          (First 10 Rows Only)

                      Jobcode    Salary  AvgSalary
                      ----------------------------
                      BCK         26007   25794.22
                      BCK         25823   25794.22
                      BCK         25996   25794.22
                      BCK         26452   25794.22
                      BCK         26896   25794.22
                      FA1         23177   23039.36
                      FA1         23738   23039.36
                      FA1         23979   23039.36
                      FA1         23916   23039.36
                      FA1         23644   23039.36
```

Keep in mind that PROC SQL remerges data when

- the values returned by a summary function are used in a calculation. For example, the following query returns the values of X and the percent of the total for each row. On the first pass, PROC SQL computes the sum of X, and on the second pass PROC SQL computes the percentage of the total for each value of X:

```
proc sql;
   title 'Percentage of the Total';
   select X, (100*x/sum(X)) as Pct_Total
      from summary;
```

```
                              Percentage of the Total

                                       x  Pct_Total
                                 -------------------
                                      32   14.81481
                                      86   39.81481
                                      49   22.68519
                                      49   22.68519
```

- the values returned by a summary function are compared to values of a column that is not specified in the GROUP BY clause. For example, the following query uses the PROCLIB.PAYROLL table. PROC SQL remerges data because the column Salary is not specified in the GROUP BY clause:

```
proc sql;
   select  jobcode,  salary,
           avg(salary) as avsal
      from proclib.payroll
      group by jobcode
      having salary > avsal;
```

- a column from the input table is specified in the SELECT clause and is not specified in the GROUP BY clause. This rule does not refer to columns used as arguments to summary functions in the SELECT clause.

  For example, in the following query, the presence of IdNumber in the SELECT clause causes PROC SQL to remerge the data because IdNumber is not involved in grouping or summarizing during the first pass. In order for PROC SQL to retrieve the values for IdNumber, it must make a second pass through the data.

```
proc sql;
   select IdNumber, jobcode,
          avg(salary) as avsal
      from proclib.payroll
      group by jobcode;
```

---

table-expression

Defines part or all of a query-expression.

See also: "query-expression" on page 1093

SELECT <**DISTINCT**> *object-item*<, … *object-item*>
<**INTO** :*macro-variable-specification*
<, … :*macro-variable-specification*>>
FROM *from-list*
<**WHERE** sql-expression>
<**GROUP BY** *group-by-item* <, … *group-by-item*>>
<**HAVING** sql-expression>

See "SELECT Statement" on page 1058 for complete information on the SELECT statement.

Details

A table-expression is a SELECT statement. It is the fundamental building block of most SQL procedure statements. You can combine the results of multiple table-expressions with set operators, which creates a query-expression. Use one ORDER BY clause for an entire query-expression. Place a semicolon only at the end of the entire query-expression. A query-expression is often only one SELECT statement or table-expression.

UPPER function

Converts the case of a character string to uppercase.

See also: "LOWER function" on page 1093

UPPER (sql-expression)

- sql-expression must be a character string and is described in "sql-expression" on page 1099.

Details

The UPPER function operates on character strings. UPPER converts the case of its argument to all uppercase.

Concepts: SQL Procedure

Using SAS Data Set Options with PROC SQL

In PROC SQL, you can apply most of the SAS data set options, such as KEEP= and DROP=, to tables or SAS/ACCESS views any time that you specify a table or SAS/ACCESS view. In the SQL procedure, SAS data set options that are separated by spaces are enclosed in parentheses, and they follow immediately after the table or SAS/ACCESS view name. In the following PROC SQL step, RENAME= renames LNAME to LASTNAME for the STAFF1 table. OBS= restricts the number of rows written to STAFF1 to 15:

```
proc sql;
   create table
          staff1(rename=(lname=lastname)) as
      select *
         from staff(obs=15);
```

SAS data set options can be combined with SQL statement arguments:

```
proc sql;
   create table test
      (a character, b numeric, pw=cat);
   create index staffidx on
      staff1 (lastname, alter=dog);
```

You cannot use SAS data set options with DICTIONARY tables because DICTIONARY tables are read-only objects.

The only SAS data set options that you can use with PROC SQL views are those that assign and provide SAS passwords: READ=, WRITE=, ALTER=, and PW=.

See *SAS Language Reference: Dictionary* for a description of SAS data set options.

Connecting to a DBMS Using the SQL Procedure Pass-Through Facility

What Is the Pass-Through Facility?

The SQL Procedure Pass-Through Facility enables you to send DBMS-specific SQL statements directly to a DBMS for execution. The Pass-Through Facility uses a SAS/ACCESS interface engine to connect to the DBMS. Therefore, you must have SAS/ACCESS software installed for your DBMS.

You submit SQL statements that are DBMS-specific. For example, you pass Transact-SQL statements to a SYBASE database. The Pass-Through Facility's basic syntax is the same for all the DBMSs. Only the statements that are used to connect to the DBMS and the SQL statements are DBMS-specific.

With the Pass-Through Facility, you can perform the following tasks:

- establish a connection with the DBMS using a CONNECT statement and terminate the connection with the DISCONNECT statement.
- send nonquery DBMS-specific SQL statements to the DBMS using the EXECUTE statement.
- retrieve data from the DBMS to be used in a PROC SQL query with the CONNECTION TO component in a SELECT statement's FROM clause.

You can use the Pass-Through Facility statements in a query, or you can store them in a PROC SQL view. When a view is stored, any options that are specified in the corresponding CONNECT statement are also stored. Thus, when the PROC SQL view is used in a SAS program, SAS can automatically establish the appropriate connection to the DBMS.

See "CONNECT Statement" on page 1042, "DISCONNECT Statement" on page 1053, "EXECUTE Statement" on page 1055, "CONNECTION TO" on page 1079, and "The Pass-Through Facility for Relational Databases" in *SAS/ACCESS for Relational Databases: Reference*.

Note: SAS procedures that do multipass processing cannot operate on PROC SQL views that store Pass-Through Facility statements, because the Pass-Through Facility does not allow reopening of a table after the first record has been retrieved. To work around this limitation, create a SAS data set from the view and use the SAS data set as the input data set. △

Return Codes

As you use PROC SQL statements that are available in the Pass-Through Facility, any errors are written to the SAS log. The return codes and messages that are generated by the Pass-Through Facility are available to you through the SQLXRC and SQLXMSG macro variables. Both macro variables are described in "Using Macro Variables Set by PROC SQL" on page 1119.

Connecting to a DBMS Using the LIBNAME Statement

For many DBMSs, you can directly access DBMS data by assigning a libref to the DBMS using the SAS/ACCESS LIBNAME statement. Once you have associated a libref with the DBMS, you can specify a DBMS table in a two-level SAS name and work with the table like any SAS data set. You can also embed the LIBNAME statement in a PROC SQL view (see "CREATE VIEW Statement" on page 1049).

PROC SQL will take advantage of the capabilities of a DBMS by passing it certain operations whenever possible. For example, before implementing a join, PROC SQL checks to see if the DBMS can do the join. If it can, then PROC SQL passes the join to the DBMS. This enhances performance by reducing data movement and translation. If the DBMS cannot do the join, then PROC SQL processes the join. Using the SAS/ACCESS LIBNAME statement can often provide you with the performance benefits of the SQL Procedure Pass-Through Facility without having to write DBMS-specific code.

To use the SAS/ACCESS LIBNAME statement, you must have SAS/ACCESS software installed for your DBMS. For more information about the SAS/ACCESS LIBNAME statement, refer to the SAS/ACCESS documentation for your DBMS.

Using the DICTIONARY Tables

What Are DICTIONARY Tables?

DICTIONARY tables are special, read-only SAS data views that contain information about your SAS session. For example, the DICTIONARY.COLUMNS table contains information, such as name, type, length, and format, about all columns in all tables that are known to the current SAS session. DICTIONARY tables are accessed by using the libref DICTIONARY in the FROM clause in a SELECT statement in PROC SQL. Additionally, there are PROC SQL views, stored in the SASHELP library and known as *SASHELP views*, that reference the DICTIONARY tables and that can be used in other SAS procedures and in the DATA step.

Note: You cannot use data set options with DICTIONARY tables. △

For an example that demonstrates the use of a DICTIONARY table, see Example 6 on page 1136.

The following table describes the DICTIONARY tables that are available and shows the associated SASHELP view(s) for each table.

Table 44.2 DICTIONARY Tables and Associated SASHELP Views

DICTIONARY table	SASHELP view	Description
CATALOGS	VCATALG	Contains information about known SAS catalogs.
CHECK_CONSTRAINTS	VCHKCON	Contains information about known check constraints.
COLUMNS	VCOLUMN	Contains information about columns in all known tables.
CONSTRAINT_COLUMN_USAGE	VCNCOLU	Contains information about columns that are referred to by integrity constraints.
CONSTRAINT_TABLE_USAGE	VCNTABU	Contains information about tables that have integrity constraints defined on them.
DICTIONARIES	VDCTNRY	Contains information about all DICTIONARY tables.
ENGINES	VENGINE	Contains information about SAS engines.

DICTIONARY table	SASHELP view	Description
EXTFILES	VEXTFL	Contains information about known external files.
FORMATS	VFORMAT	Contains information about currently accessible formats and informats.
GOPTIONS	VGOPT VALLOPT	Contains information about currently defined graphics options (SAS/GRAPH software). SASHELP.VALLOPT includes SAS system options as well as graphics options.
INDEXES	VINDEX	Contains information about known indexes.
LIBNAMES	VLIBNAM	Contains information about currently defined SAS data libraries.
MACROS	VMACRO	Contains information about currently defined macros.
MEMBERS	VMEMBER VSACCES VSCATLG VSLIB VSTABLE VSTABVW VSVIEW	Contains information about all objects that are in currently defined SAS data libraries. SASHELP.VMEMBER contains information for all member types; the other SASHELP views are specific to particular member types (such as tables or views).
OPTIONS	VOPTION VALLOPT	Contains information on SAS system options. SASHELP.VALLOPT includes graphics options as well as SAS system options.
REFERENTIAL_CONSTRAINTS	VREFCON	Contains information about referential constraints.
STYLES	VSTYLE	Contains information about known ODS styles.
TABLE_CONSTRAINTS	VTABCON	Contains information about integrity constraints in all known tables.
TABLES	VTABLE	Contains information about known tables.
TITLES	VTITLE	Contains information about currently defined titles and footnotes.
VIEWS	VVIEW	Contains information about known data views.

Retrieving Information about DICTIONARY Tables and SASHELP Views

To see how each DICTIONARY table is defined, submit a DESCRIBE TABLE statement. After you know how a table is defined, you can use its column names in a subsetting WHERE clause in order to retrieve more specific information. For example:

```
proc sql;
   describe table dictionary.indexes;
```

The results are written to the SAS log:

```
6    proc sql;
7       describe table dictionary.indexes;
NOTE: SQL table DICTIONARY.INDEXES was created like:

create table DICTIONARY.INDEXES
  (
   libname char(8) label='Library Name',
   memname char(32) label='Member Name',
   memtype char(8) label='Member Type',
   name char(32) label='Column Name',
   idxusage char(9) label='Column Index Type',
   indxname char(32) label='Index Name',
   indxpos num label='Position of Column in Concatenated Key',
   nomiss char(3) label='Nomiss Option',
   unique char(3) label='Unique Option'
  );
```

Use the DESCRIBE VIEW statement in PROC SQL to find out how a SASHELP view is defined. Here's an example:

```
proc sql;
   describe view sashelp.vstabvw;
```

The results are written to the SAS log:

```
6    proc sql;
7       describe view sashelp.vstabvw;
NOTE: SQL view SASHELP.VSTABVW is defined as:

        select libname, memname, memtype
          from DICTIONARY.MEMBERS
         where (memtype='VIEW') or (memtype='DATA')
      order by libname asc, memname asc;
```

Using DICTIONARY Tables

DICTIONARY tables are commonly used to monitor and manage SAS sessions because the data is more easily manipulated than the output from, for example, PROC DATASETS. You can query DICTIONARY tables the same way that you query any other table, including subsetting with a WHERE clause, ordering the results, and creating PROC SQL views. Note that many character values in the DICTIONARY tables are stored as all-uppercase characters; you should design your queries accordingly.

Because DICTIONARY tables are read-only objects, you cannot insert rows or columns, alter column attributes, or add integrity constraints to them.

Note: For DICTIONARY.TABLES and SASHELP.VTABLE, if a table is read-protected with a password, then the only information that is listed for that table is the library name, member name, member type, and type of password protection; all other information is set to missing. △

DICTIONARY Tables and Performance

When querying a DICTIONARY table, SAS launches a discovery process that gathers information that is pertinent to that table. Depending on the DICTIONARY table that is being queried, this discovery process can search libraries, open tables, and execute views. Unlike other SAS procedures and the DATA step, PROC SQL can mitigate this process by optimizing the query before the discovery process is launched. Therefore, although it is possible to access DICTIONARY table information with SAS procedures or the DATA step by using the SASHELP views, it is often more efficient to use PROC SQL instead.

For example, the following programs both produce the same result, but the PROC SQL step runs much faster because the WHERE clause is processed prior to opening the tables that are referenced by the SASHELP.VCOLUMN view:

```
data mytable;
   set sashelp.vcolumn;
   where libname='WORK' and memname='SALES';
run;


proc sql;
   create table mytable as
      select * from sashelp.vcolumn
      where libname='WORK' and memname='SALES';
quit;
```

Note: SAS does not maintain DICTIONARY table information between queries. Each query of a DICTIONARY table launches a new discovery process. △

If you are querying the same DICTIONARY table several times in a row, then you can get even faster performance by creating a temporary SAS data set (with the DATA step SET statement or PROC SQL CREATE TABLE AS statement) with the information that you want and running your query against that data set.

Using Macro Variables Set by PROC SQL

PROC SQL sets up macro variables with certain values after it executes each statement. These macro variables can be tested inside a macro to determine whether to continue executing the PROC SQL step. SAS/AF software users can also test them in a program after an SQL SUBMIT block of code, using the SYMGET function.

After each PROC SQL statement has executed, the following macro variables are updated with these values:

SQLOBS
: contains the number of rows executed by an SQL procedure statement. For example, it contains the number of rows formatted and displayed in SAS output by a SELECT statement or the number of rows deleted by a DELETE statement.

  When the NOPRINT option is specified, the value of the SQLOBS macro variable depends on whether an output table, single macro variable, macro variable range, or macro variable list is created:

  - If no output table, macro variable list, or macro variable range is created, then SQLOBS contains the value 1.
  - If an output table is created, then SQLOBS contains the number of rows in the output table.
  - If a single macro variable is created, then SQLOBS contains the value 1.
  - If a macro variable list or macro variable range is created, then SQLOBS contains the number of rows that are processed to create the macro variable list or range.

SQLRC
: contains the following status values that indicate the success of the SQL procedure statement:

  0
  : PROC SQL statement completed successfully with no errors.

  4
  : PROC SQL statement encountered a situation for which it issued a warning. The statement continued to execute.

  8
  : PROC SQL statement encountered an error. The statement stopped execution at this point.

  12
  : PROC SQL statement encountered an internal error, indicating a bug in PROC SQL that should be reported to SAS Technical Support. These errors can occur only during compile time.

  16
  : PROC SQL statement encountered a user error. This error code is used, for example, when a subquery (that can only return a single value) evaluates to more than one row. These errors can only be detected during run time.

  24
  : PROC SQL statement encountered a system error. This error is used, for example, if the system cannot write to a PROC SQL table because the disk is full. These errors can occur only during run time.

  28
  : PROC SQL statement encountered an internal error, indicating a bug in PROC SQL that should be reported to SAS Technical Support. These errors can occur only during run time.

SQLOOPS
: contains the number of iterations that the inner loop of PROC SQL executes. The number of iterations increases proportionally with the complexity of the query. See also the description of LOOPS= on page 1036.

SQLXRC
: contains the DBMS-specific return code that is returned by the Pass-Through Facility.

SQLXMSG
: contains descriptive information and the DBMS-specific return code for the error that is returned by the Pass-Through Facility.

  Note: Because the value of the SQLXMSG macro variable can contain special characters (such as &, %, /, *, and ;), use the %SUPERQ macro function when printing the value:

```
%put %superq(sqlxmsg);
```

  See *SAS Macro Language: Reference* for information about the %SUPERQ function. △

This example retrieves the data but does not display them in SAS output because of the NOPRINT option in the PROC SQL statement. The %PUT macro statement displays the macro variables values.

```
proc sql noprint;
   select *
```

```
      from proclib.payroll;

%put sqlobs=**&sqlobs**
     sqloops=**&sqloops**
     sqlrc=**&sqlrc**;
```

The message in Output 44.3 appears in the SAS log and gives you the macros' values.

Output 44.3 PROC SQL Macro Variable Values

```
40   options ls=80;
41
42   proc sql noprint;
43      select *
44         from proclib.payroll;
45
46   %put sqlobs=**&sqlobs**
47        sqloops=**&sqloops**
48        sqlrc=**&sqlrc**;
sqlobs=**1**        sqloops=**11**        sqlrc=**0**
```

Macro variables that are generated by PROC SQL follow the scoping rules for %LET. For more information about macro variable scoping, see *SAS Macro Language: Reference*.

Updating PROC SQL and SAS/ACCESS Views

You can update PROC SQL and SAS/ACCESS views using the INSERT, DELETE, and UPDATE statements, under the following conditions.

- If the view accesses a DBMS table, then you must have been granted the appropriate authorization by the external database management system (for example, DB2). You must have installed the SAS/ACCESS software for your DBMS. See the SAS/ACCESS interface guide for your DBMS for more information on SAS/ACCESS views.
- You can update only a single table through a view. The table cannot be joined to another table or linked to another table with a set-operator. The view cannot contain a subquery.
- You can update a column in a view using the column's alias, but you cannot update a derived column, that is, a column produced by an expression. In the following example, you can update the column SS, but not WeeklySalary.

  ```
  create view EmployeeSalaries as
     select Employee, SSNumber as SS,
            Salary/52 as WeeklySalary
            from employees;
  ```

- You cannot update a view containing an ORDER BY.

Note: Starting in SAS System 9, PROC SQL views, the Pass-Through Facility, and the SAS/ACCESS LIBNAME statement are the preferred ways to access relational DBMS data; SAS/ACCESS views are no longer recommended. You can convert existing SAS/ACCESS views to PROC SQL views by using the CV2VIEW procedure. See The CV2VIEW Procedure in *SAS/ACCESS for Relational Databases: Reference* for more information. △

PROC SQL and the ANSI Standard

Compliance

PROC SQL follows most of the guidelines set by the American National Standards Institute (ANSI) in its implementation of SQL. However, it is not fully compliant with the current ANSI Standard for SQL.*

The SQL research project at SAS has focused primarily on the expressive power of SQL as a query language. Consequently, some of the database features of SQL have not yet been implemented in PROC SQL.

SQL Procedure Enhancements

Reserved Words

PROC SQL reserves very few keywords and then only in certain contexts. The ANSI Standard reserves all SQL keywords in all contexts. For example, according to the Standard you cannot name a column GROUP because of the keywords GROUP BY.

The following words are reserved in PROC SQL:

- The keyword CASE is always reserved; its use in the CASE expression (an SQL2 feature) precludes its use as a column name.

  If you have a column named CASE in a table and you want to specify it in a PROC SQL step, then you can use the SAS data set option RENAME= to rename that column for the duration of the query. You can also surround CASE in double quotation marks (“CASE”) and set the PROC SQL option DQUOTE=ANSI.
- The keywords AS, ON, FULL, JOIN, LEFT, FROM, WHEN, WHERE, ORDER, GROUP, RIGHT, INNER, OUTER, UNION, EXCEPT, HAVING, and INTERSECT cannot normally be used for table aliases. These keywords all introduce clauses that appear after a table name. Since the alias is optional, PROC SQL deals with this ambiguity by assuming that any one of these words introduces the corresponding clause and is not the alias. If you want to use one of these keywords as an alias, then use the PROC SQL option DQUOTE=ANSI.
- The keyword USER is reserved for the current userid. If you specify USER on a SELECT statement in conjunction with a CREATE TABLE statement, then the column is created in the table with a temporary column name that is similar to _TEMA001. If you specify USER in a SELECT statement without using the CREATE TABLE statement, then the column is written to the output without a column heading. In either case, the value for the column varies by operating environment, but is typically the userid of the user who is submitting the program or the value of the &SYSJOBID automatic macro variable.

  If you have a column named USER in a table and you want to specify it in a PROC SQL step, then you can use the SAS data set option RENAME= to rename that column for the duration of the query. You can also enclose USER with double quotation marks (“USER”) and set the PROC SQL option DQUOTE=ANSI.

* International Organization for Standardization (ISO): *Database SQL*. Document ISO/IEC 9075:1992. Also available as American National Standards Institute (ANSI) Document ANSI X3.135-1992.

Column Modifiers

PROC SQL supports the SAS INFORMAT=, FORMAT=, and LABEL= modifiers for expressions within the SELECT clause. These modifiers control the format in which output data are displayed and labeled.

Alternate Collating Sequences

PROC SQL allows you to specify an alternate collating (sorting) sequence to be used when you specify the ORDER BY clause. See the description of the SORTSEQ= option in "PROC SQL Statement" on page 1033 for more information.

ORDER BY Clause in a View Definition

PROC SQL permits you to specify an ORDER BY clause in a CREATE VIEW statement. When the view is queried, its data are always sorted according to the specified order unless a query against that view includes a different ORDER BY clause. See "CREATE VIEW Statement" on page 1049 for more information.

In-Line Views

The ability to code nested query-expressions in the FROM clause is a requirement of the ANSI Standard. PROC SQL supports such nested coding.

Outer Joins

The ability to include columns that both match and do not match in a join-expression is a requirement of the ANSI Standard. PROC SQL supports this ability.

Arithmetic Operators

PROC SQL supports the SAS exponentiation (**) operator. PROC SQL uses the notation <> to mean not equal.

Orthogonal Expressions

PROC SQL permits the combination of comparison, Boolean, and algebraic expressions. For example, (X=3)*7 yields a value of 7 if X=3 is true because true is defined to be 1. If X=3 is false, then it resolves to 0 and the entire expression yields a value of 0.

PROC SQL permits a subquery in any expression. This feature is required by the ANSI Standard. Therefore, you can have a subquery on the left side of a comparison operator in the WHERE expression.

PROC SQL permits you to order and group data by any kind of mathematical expression (except those including summary functions) using ORDER BY and GROUP BY clauses. You can also group by an expression that appears on the SELECT clause by using the integer that represents the expression's ordinal position in the SELECT clause. You are not required to select the expression by which you are grouping or ordering. See ORDER BY Clause on page 1067 and GROUP BY Clause on page 1065 for more information.

Set Operators

The set operators UNION, INTERSECT, and EXCEPT are required by the ANSI Standard. PROC SQL provides these operators plus the OUTER UNION operator.

The ANSI Standard also requires that the tables being operated upon all have the same number of columns with matching data types. The SQL procedure works on tables that have the same number of columns, as well as on those that do not, by creating virtual columns so that a query can evaluate correctly. See “query-expression” on page 1093 for more information.

Statistical Functions

PROC SQL supports many more summary functions than required by the ANSI Standard for SQL.

PROC SQL supports the remerging of summary function results into the table’s original data. For example, computing the percentage of total is achieved with 100**x*/SUM(*x*) in PROC SQL. See “summary-function” on page 1107 for more information on the available summary functions and remerging data.

SAS DATA Step Functions

PROC SQL supports all the functions available to the SAS DATA step, except for LAG, DIF, and SOUND. Other SQL databases support their own set of functions.

SQL Procedure Omissions

COMMIT Statement

The COMMIT statement is not supported.

ROLLBACK Statement

The ROLLBACK statement is not supported. The UNDO_POLICY= option in the PROC SQL statement addresses rollback. See the description of the UNDO_POLICY= option in “PROC SQL Statement” on page 1033 for more information.

Identifiers and Naming Conventions

In SAS, table names, column names, and aliases are limited to 32 characters and can contain mixed case. For more information on SAS naming conventions, see *SAS Language Reference: Dictionary*. The ANSI Standard for SQL allows longer names.

Granting User Privileges

The GRANT statement, PRIVILEGES keyword, and authorization-identifier features of SQL are not supported. You might want to use operating environment-specific means of security instead.

Three-Valued Logic

ANSI-compatible SQL has three-valued logic, that is, special cases for handling comparisons involving NULL values. Any value compared with a NULL value evaluates to NULL.

PROC SQL follows the SAS convention for handling missing values: when numeric NULL values are compared to non-NULL numbers, the NULL values are less than or smaller than all the non-NULL values; when character NULL values are compared to non-NULL characters, the character NULL values are treated as a string of blanks.

Embedded SQL

Currently there is no provision for embedding PROC SQL statements in other SAS programming environments, such as the DATA step or SAS/IML software.

Examples: SQL Procedure

Example 1: Creating a Table and Inserting Data into It

Procedure features:

CREATE TABLE statement

column-modifier

INSERT statement

VALUES clause

SELECT clause

FROM clause

Table: PROCLIB.PAYLIST

This example creates the table PROCLIB.PAYLIST and inserts data into it.

Program

Declare the PROCLIB library. The PROCLIB library is used in these examples to store created tables.

```
libname proclib 'SAS-data-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.PAYLIST table. The CREATE TABLE statement creates PROCLIB.PAYLIST with six empty columns. Each column definition indicates whether the column is character or numeric. The number in parentheses specifies the width of the column. INFORMAT= and FORMAT= assign date informats and formats to the Birth and Hired columns.

```
proc sql;
   create table proclib.paylist
       (IdNum char(4),
```

```
          Gender char(1),
          Jobcode char(3),
          Salary num,
          Birth num informat=date7.
                    format=date7.,
          Hired num informat=date7.
                    format=date7.);
```

Insert values into the PROCLIB.PAYLIST table. The INSERT statement inserts data values into PROCLIB.PAYLIST according to the position in the VALUES clause. Therefore, in the first VALUES clause, **1639** is inserted into the first column, **F** into the second column, and so forth. Dates in SAS are stored as integers with 0 equal to January 1, 1960. Suffixing the date with a **d** is one way to use the internal value for dates.

```
insert into proclib.paylist
    values('1639','F','TA1',42260,'26JUN70'd,'28JAN91'd)
    values('1065','M','ME3',38090,'26JAN54'd,'07JAN92'd)
    values('1400','M','ME1',29769.'05NOV67'd,'16OCT90'd)
```

Include missing values in the data. The value **null** represents a missing value for the character column Jobcode. The period represents a missing value for the numeric column Salary.

```
    values('1561','M',null,36514,'30NOV63'd,'07OCT87'd)
    values('1221','F','FA3',.,'22SEP63'd,'04OCT94'd);
```

Specify the title.

```
   title 'PROCLIB.PAYLIST Table';
```

Display the entire PROCLIB.PAYLIST table. The SELECT clause selects columns from PROCLIB.PAYLIST. The asterisk (*) selects all columns. The FROM clause specifies PROCLIB.PAYLIST as the table to select from.

```
select *
   from proclib.paylist;
```

Output Table

PROCLIB.PAYLIST

```
                    PROCLIB.PAYLIST Table

Id
Num   Gender  Jobcode    Salary    Birth    Hired
-------------------------------------------------
1639  F       TA1         42260  26JUN70  28JAN91
1065  M       ME3         38090  26JAN54  07JAN92
1400  M       ME1         29769  05NOV67  16OCT90
1561  M                   36514  30NOV63  07OCT87
1221  F       FA3             .  22SEP63  04OCT94
```

Example 2: Creating a Table from a Query's Result

Procedure features:

CREATE TABLE statement

AS query-expression

SELECT clause

column alias
FORMAT= column-modifier
object-item

Other features:

data set option

OBS=

Tables:

PROCLIB.PAYROLL, PROCLIB.BONUS

This example builds a column with an arithmetic expression and creates the PROCLIB.BONUS table from the query's result.

Input Table

PROCLIB.PAYROLL (Partial Listing)

```
                        PROCLIB.PAYROLL
                       First 10 Rows Only

         Id
         Number  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
```

Program

Declare the PROCLIB library. The PROCLIB library is used in these examples to store created tables.

```
libname proclib 'SAS-data-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.BONUS table. The CREATE TABLE statement creates the table PROCLIB.BONUS from the result of the subsequent query.

```
proc sql;
   create table proclib.bonus as
```

Select the columns to include. The SELECT clause specifies that three columns will be in the new table: IdNumber, Salary, and Bonus. FORMAT= assigns the DOLLAR8. format to Salary. The Bonus column is built with the SQL expression **salary*.025**.

```
   select IdNumber, Salary format=dollar8.,
          salary*.025 as Bonus format=dollar8.
      from proclib.payroll;
```

Specify the title.

```
title 'BONUS Information';
```

Display the first 10 rows of the PROCLIB.BONUS table. The SELECT clause selects columns from PROCLIB.BONUS. The asterisk (*) selects all columns. The FROM clause specifies PROCLIB.BONUS as the table to select from. The OBS= data set option limits the printing of the output to 10 rows.

```
select *
   from proclib.bonus(obs=10);
```

Output

PROCLIB.BONUS

```
                    BONUS Information

                Id
                Number    Salary     Bonus
                --------------------------
                1919     $34,376      $859
                1653     $35,108      $878
                1400     $29,769      $744
                1350     $32,886      $822
                1401     $38,822      $971
                1499     $43,025    $1,076
                1101     $18,723      $468
                1333     $88,606    $2,215
                1402     $32,615      $815
                1479     $38,785      $970
```

Example 3: Updating Data in a PROC SQL Table

Procedure features:

ALTER TABLE statement

- DROP clause
- MODIFY clause

UPDATE statement

- SET clause

CASE expression

Table: EMPLOYEES

This example updates data values in the EMPLOYEES table and drops a column.

Input

```
data Employees;
   input IdNum $4. +2 LName $11. FName $11. JobCode $3.
          +1 Salary 5. +1 Phone $12.;
   datalines;
1876  CHIN       JACK       TA1 42400 212/588-5634
1114  GREENWALD  JANICE     ME3 38000 212/588-1092
1556  PENNINGTON MICHAEL    ME1 29860 718/383-5681
1354  PARKER     MARY       FA3 65800 914/455-2337
1130  WOOD       DEBORAH    PT2 36514 212/587-0013
;
```

Program

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;
```

Display the entire EMPLOYEES table. The SELECT clause displays the table before the updates. The asterisk (*) selects all columns for display. The FROM clause specifies EMPLOYEES as the table to select from.

```
proc sql;
   title 'Employees Table';
   select * from Employees;
```

Update the values in the Salary column. The UPDATE statement updates the values in EMPLOYEES. The SET clause specifies that the data in the Salary column be multiplied by 1.04 when the job code ends with a `1` and 1.025 for all other job codes. (The two underscores represent any character.) The CASE expression returns a value for each row that completes the SET clause.

```
update employees
      set salary=salary*
      case when jobcode like '__1' then 1.04
           else 1.025
      end;
```

Modify the format of the Salary column and delete the Phone column. The ALTER TABLE statement specifies EMPLOYEES as the table to alter. The MODIFY clause permanently modifies the format of the Salary column. The DROP clause permanently drops the Phone column.

```
   alter table employees
      modify salary num format=dollar8.
      drop phone;
```

Specify the title.

```
title 'Updated Employees Table';
```

Display the entire updated EMPLOYEES table. The SELECT clause displays the EMPLOYEES table after the updates. The asterisk (*) selects all columns.

```
select * from employees;
```

Output

```
                          Employees Table                          1

        Id                              Job
        Num   LName         FName         Code    Salary  Phone
        ------------------------------------------------------------
        1876  CHIN          JACK          TA1      42400  212/588-5634
        1114  GREENWALD     JANICE        ME3      38000  212/588-1092
        1556  PENNINGTON    MICHAEL       ME1      29860  718/383-5681
        1354  PARKER        MARY          FA3      65800  914/455-2337
        1130  WOOD          DEBORAH       PT2      36514  212/587-0013
```

```
                      Updated Employees Table                      2

               Id                              Job
               Num   LName         FName         Code    Salary
               ----------------------------------------------
               1876  CHIN          JACK          TA1    $44,096
               1114  GREENWALD     JANICE        ME3    $38,950
               1556  PENNINGTON    MICHAEL       ME1    $31,054
               1354  PARKER        MARY          FA3    $67,445
               1130  WOOD          DEBORAH       PT2    $37,427
```

Example 4: Joining Two Tables

Procedure features:

- FROM clause
  - table alias
- inner join
- joined-table component
- PROC SQL statement option
  - NUMBER
- WHERE clause
  - IN condition

Tables: PROCLIB.STAFF, PROCLIB.PAYROLL

This example joins two tables in order to get more information about data that are common to both tables.

Input Tables

PROCLIB.STAFF (Partial Listing)

```
                              PROCLIB.STAFF
                           First 10 Rows Only

 Id
 Num   Lname            Fname            City             State  Hphone
 ----------------------------------------------------------------------------
 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
```

PROCLIB.PAYROLL (Partial Listing)

```
                             PROCLIB.PAYROLL
                           First 10 Rows Only

             Id
             Number  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
```

Program

Declare the PROCLIB library. The PROCLIB library is used in these examples to store created tables.

```
libname proclib 'SAS-data-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;
```

Add row numbers to PROC SQL output. NUMBER adds a column that contains the row number.

```
proc sql number;
```

Specify the title.

```
    title 'Information for Certain Employees Only';
```

Select the columns to display. The SELECT clause selects the columns to show in the output.

```
    select Lname, Fname, City, State,
           IdNumber, Salary, Jobcode
```

Specify the tables from which to obtain the data. The FROM clause lists the tables to select from.

```
       from proclib.staff, proclib.payroll
```

Specify the join criterion and subset the query. The WHERE clause specifies that the tables are joined on the ID number from each table. WHERE also further subsets the query with the IN condition, which returns rows for only four employees.

```
       where idnumber=idnum and idnum in
              ('1919', '1400', '1350', '1333');
```

Output

```
                   Information for Certain Employees Only

                                                                 Id
       Row  Lname            Fname            City             State  Number
              Salary  Jobcode
     -----------------------------------------------------------------------
         1  ADAMS            GERALD           STAMFORD         CT     1919
               34376  TA2

         2  ALHERTANI        ABDULLAH         NEW YORK         NY     1400
               29769  ME1

         3  ALVAREZ          MERCEDES         NEW YORK         NY     1350
               32886  FA3

         4  BANADYGA         JUSTIN           STAMFORD         CT     1333
               88606  PT2
```

Example 5: Combining Two Tables

Procedure features:

- DELETE statement
- IS condition
- RESET statement option
  - DOUBLE
- UNION set operator

Tables: PROCLIB.NEWPAY, PROCLIB.PAYLIST, PROCLIB.PAYLIST2

This example creates a new table, PROCLIB.NEWPAY, by concatenating two other tables: PROCLIB.PAYLIST and PROCLIB.PAYLIST2.

Input Tables

PROCLIB.PAYLIST

```
                   Information for Certain Employees Only

             Id
             Num   Gender  Jobcode    Salary    Birth    Hired
             -------------------------------------------------
             1639  F       TA1         42260  26JUN70  28JAN91
             1065  M       ME3         38090  26JAN54  07JAN92
             1400  M       ME1         29769  05NOV67  16OCT90
             1561  M                   36514  30NOV63  07OCT87
             1221  F       FA3             .  22SEP63  04OCT94
```

PROCLIB.PAYLIST2

```
                    PROCLIB.PAYLIST2 Table

          Id
          Num   Gender  Jobcode    Salary    Birth    Hired
          ------------------------------------------------
          1919  M       TA2         34376  12SEP66  04JUN87
          1653  F       ME2         31896  15OCT64  09AUG92
          1350  F       FA3         36886  31AUG55  29JUL91
          1401  M       TA3         38822  13DEC55  17NOV93
          1499  M       ME1         23025  26APR74  07JUN92
```

Program

Declare the PROCLIB library. The PROCLIB library is used in these examples to store created tables.

```
libname proclib 'SAS-data-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=60;
```

Create the PROCLIB.NEWPAY table. The SELECT clauses select all the columns from the tables that are listed in the FROM clauses. The UNION set operator concatenates the query results that are produced by the two SELECT clauses.

```
proc sql;
   create table proclib.newpay as
      select * from proclib.paylist
      union
      select * from proclib.paylist2;
```

Delete rows with missing Jobcode or Salary values. The DELETE statement deletes rows from PROCLIB.NEWPAY that satisfy the WHERE expression. The IS condition specifies rows that contain missing values in the Jobcode or Salary column.

```
   delete
      from proclib.newpay
      where jobcode is missing or salary is missing;
```

Reset the PROC SQL environment and double-space the output. RESET changes the procedure environment without stopping and restarting PROC SQL. The DOUBLE option double-spaces the output. (The DOUBLE option has no effect on ODS output.)

```
reset double;
```

Specify the title.

```
title 'Personnel Data';
```

Display the entire PROCLIB.NEWPAY table. The SELECT clause selects all columns from the newly created table, PROCLIB.NEWPAY.

```
select *
   from proclib.newpay;
```

Output

```
                    Personnel Data

Id
Num   Gender  Jobcode    Salary    Birth    Hired
-------------------------------------------------
1065  M       ME3         38090  26JAN54  07JAN92

1350  F       FA3         36886  31AUG55  29JUL91

1400  M       ME1         29769  05NOV67  16OCT90

1401  M       TA3         38822  13DEC55  17NOV93

1499  M       ME1         23025  26APR74  07JUN92

1639  F       TA1         42260  26JUN70  28JAN91

1653  F       ME2         31896  15OCT64  09AUG92

1919  M       TA2         34376  12SEP66  04JUN87
```

Example 6: Reporting from DICTIONARY Tables

Procedure features:
DESCRIBE TABLE statement
DICTIONARY.*table-name* component

Table: DICTIONARY.MEMBERS

This example uses DICTIONARY tables to show a list of the SAS files in a SAS data library. If you do not know the names of the columns in the DICTIONARY table that you are querying, then use a DESCRIBE TABLE statement with the table.

Program

Declare the PROCLIB library. The PROCLIB library is used in these examples to store created tables.

```
libname proclib 'SAS-data-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. SOURCE writes the programming statements to the SAS log.

```
options nodate pageno=1 source linesize=80 pagesize=60;
```

List the column names from the DICTIONARY.MEMBERS table. DESCRIBE TABLE writes the column names from DICTIONARY.MEMBERS to the SAS log.

```
proc sql;
   describe table dictionary.members;
```

Specify the title.

```
   title 'SAS Files in the PROCLIB Library';
```

Display a list of files in the PROCLIB library. The SELECT clause selects the MEMNAME and MEMTYPE columns. The FROM clause specifies DICTIONARY.MEMBERS as the table to select from. The WHERE clause subsets the output to include only those rows that have a libref of *PROCLIB* in the LIBNAME column.

```
   select memname, memtype
      from dictionary.members
      where libname='PROCLIB';
```

Log

```
277  options nodate pageno=1 source linesize=80 pagesize=60;
278
279  proc sql;
280     describe table dictionary.members;
NOTE: SQL table DICTIONARY.MEMBERS was created like:

create table DICTIONARY.MEMBERS
  (
   libname char(8) label='Library Name',
   memname char(32) label='Member Name',
   memtype char(8) label='Member Type',
   engine char(8) label='Engine Name',
   index char(32) label='Indexes',
   path char(1024) label='Path Name'
  );

281     title 'SAS Files in the PROCLIB Library';
282
283     select memname, memtype
284        from dictionary.members
285        where libname='PROCLIB';
```

Output

```
                         SAS Files in the PROCLIB Library

                                                          Member
                   Member Name                            Type
                   ------------------------------------------
                   ALL                                    DATA
                   BONUS                                  DATA
                   BONUS95                                DATA
                   DELAY                                  DATA
                   HOUSES                                 DATA
                   INTERNAT                               DATA
                   MARCH                                  DATA
                   NEWPAY                                 DATA
                   PAYLIST                                DATA
                   PAYLIST2                               DATA
                   PAYROLL                                DATA
                   PAYROLL2                               DATA
                   SCHEDULE                               DATA
                   SCHEDULE2                              DATA
                   STAFF                                  DATA
                   STAFF2                                 DATA
                   SUPERV                                 DATA
                   SUPERV2                                DATA
```

Example 7: Performing an Outer Join

Procedure features:

joined-table component
left outer join
SELECT clause

COALESCE function
WHERE clause
CONTAINS condition

Tables: PROCLIB.PAYROLL, PROCLIB.PAYROLL2

This example illustrates a left outer join of the PROCLIB.PAYROLL and PROCLIB.PAYROLL2 tables.

Input Tables

PROCLIB.PAYROLL (Partial Listing)

```
                              PROCLIB.PAYROLL
                             First 10 Rows Only

              Id
              Number  Gender  Jobcode    Salary    Birth    Hired
              ---------------------------------------------------
              1009    M       TA1         28880  02MAR59  26MAR92
              1017    M       TA3         40858  28DEC57  16OCT81
              1036    F       TA3         39392  19MAY65  23OCT84
              1037    F       TA1         28558  10APR64  13SEP92
              1038    F       TA1         26533  09NOV69  23NOV91
              1050    M       ME2         35167  14JUL63  24AUG86
              1065    M       ME2         35090  26JAN44  07JAN87
              1076    M       PT1         66558  14OCT55  03OCT91
              1094    M       FA1         22268  02APR70  17APR91
              1100    M       BCK         25004  01DEC60  07MAY88
```

PROCLIB.PAYROLL2

```
                              PROCLIB.PAYROLL2

                 Id
                 Num   Sex  Jobcode    Salary    Birth    Hired
                 ----------------------------------------------
                 1036  F    TA3         42465  19MAY65  23OCT84
                 1065  M    ME3         38090  26JAN44  07JAN87
                 1076  M    PT1         69742  14OCT55  03OCT91
                 1106  M    PT3         94039  06NOV57  16AUG84
                 1129  F    ME3         36758  08DEC61  17AUG91
                 1221  F    FA3         29896  22SEP67  04OCT91
                 1350  F    FA3         36098  31AUG65  29JUL90
                 1369  M    TA3         36598  28DEC61  13MAR87
                 1447  F    FA1         22123  07AUG72  29OCT92
                 1561  M    TA3         36514  30NOV63  07OCT87
                 1639  F    TA3         42260  26JUN57  28JAN84
                 1998  M    SCP         23100  10SEP70  02NOV92
```

Program

Declare the PROCLIB library. The PROCLIB library is used in these examples to store created tables.

```
libname proclib 'SAS-data-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=60;
```

Limit the number of output rows. OUTOBS= limits the output to 10 rows.

```
proc sql outobs=10;
```

Specify the title for the first query.

```
   title 'Most Current Jobcode and Salary Information';
```

Select the columns. The SELECT clause lists the columns to select. Some column names are prefixed with a table alias because they are in both tables. LABEL= and FORMAT= are column modifiers.

```
   select p.IdNumber, p.Jobcode, p.Salary,
          p2.jobcode label='New Jobcode',
          p2.salary label='New Salary' format=dollar8.
```

Specify the type of join. The FROM clause lists the tables to join and assigns table aliases. The keywords LEFT JOIN specify the type of join. The order of the tables in the FROM clause is important. PROCLIB.PAYROLL is listed first and is considered the “left” table. PROCLIB.PAYROLL2 is the “right” table.

```
      from proclib.payroll as p left join proclib.payroll2 as p2
```

Specify the join criterion. The ON clause specifies that the join be performed based on the values of the ID numbers from each table.

```
      on p.IdNumber=p2.idnum;
```

Output

As the output shows, all rows from the left table, PROCLIB.PAYROLL, are returned. PROC SQL assigns missing values for rows in the left table, PAYROLL, that have no matching values for IdNum in PAYROLL2.

```
                Most Current Jobcode and Salary Information

                Id                            New           New
                Number  Jobcode    Salary  Jobcode       Salary
                --------------------------------------------
                1009    TA1         28880                    .
                1017    TA3         40858                    .
                1036    TA3         39392  TA3         $42,465
                1037    TA1         28558                    .
                1038    TA1         26533                    .
                1050    ME2         35167                    .
                1065    ME2         35090  ME3         $38,090
                1076    PT1         66558  PT1         $69,742
                1094    FA1         22268                    .
                1100    BCK         25004                    .
```

Specify the title for the second query.

```
title 'Most Current Jobcode and Salary Information';
```

Select the columns and coalesce the Jobcode columns. The SELECT clause lists the columns to select. COALESCE overlays the like-named columns. For each row, COALESCE returns the first nonmissing value of either P2.JOBCODE or P.JOBCODE. Because P2.JOBCODE is the first argument, if there is a nonmissing value for P2.JOBCODE, COALESCE returns that value. Thus, the output contains the most recent job code information for every employee. LABEL= assigns a column label.

```
select p.idnumber, coalesce(p2.jobcode,p.jobcode)
     label='Current Jobcode',
```

Coalesce the Salary columns. For each row, COALESCE returns the first nonmissing value of either P2.SALARY or P.SALARY. Because P2.SALARY is the first argument, if there is a nonmissing value for P2.SALARY, then COALESCE returns that value. Thus, the output contains the most recent salary information for every employee.

```
        coalesce(p2.salary,p.salary) label='Current Salary'
              format=dollar8.
```

Specify the type of join and the join criterion. The FROM clause lists the tables to join and assigns table aliases. The keywords LEFT JOIN specify the type of join. The ON clause specifies that the join is based on the ID numbers from each table.

```
from proclib.payroll p left join proclib.payroll2 p2
on p.IdNumber=p2.idnum;
```

Output

```
                  Most Current Jobcode and Salary Information

                           Id      Current   Current
                           Number  Jobcode    Salary
                           -------------------------
                           1009    TA1       $28,880
                           1017    TA3       $40,858
                           1036    TA3       $42,465
                           1037    TA1       $28,558
                           1038    TA1       $26,533
                           1050    ME2       $35,167
                           1065    ME3       $38,090
                           1076    PT1       $69,742
                           1094    FA1       $22,268
                           1100    BCK       $25,004
```

Subset the query. The WHERE clause subsets the left join to include only those rows containing the value **TA**.

```
title 'Most Current Information for Ticket Agents';
select p.IdNumber,
       coalesce(p2.jobcode,p.jobcode) label='Current Jobcode',
       coalesce(p2.salary,p.salary) label='Current Salary'
   from proclib.payroll p left join proclib.payroll2 p2
   on p.IdNumber=p2.idnum
   where p2.jobcode contains 'TA';
```

Output

```
                   Most Current Information for Ticket Agents

                           Id      Current   Current
                           Number  Jobcode    Salary
                           -------------------------
                           1036    TA3         42465
                           1369    TA3         36598
                           1561    TA3         36514
                           1639    TA3         42260
```

Example 8: Creating a View from a Query's Result

Procedure features:

CREATE VIEW statement

GROUP BY clause

SELECT clause

COUNT function

HAVING clause

Other features:

AVG summary function

data set option

PW=

Tables: PROCLIB.PAYROLL, PROCLIB.JOBS

This example creates the PROC SQL view PROCLIB.JOBS from the result of a query-expression.

Input Table

PROCLIB.PAYROLL (Partial Listing)

```
                         PROCLIB.PAYROLL
                        First 10 Rows Only

Id
Number  Gender  Jobcode    Salary    Birth    Hired
---------------------------------------------------
1009    M       TA1         28880  02MAR59  26MAR92
1017    M       TA3         40858  28DEC57  16OCT81
1036    F       TA3         39392  19MAY65  23OCT84
1037    F       TA1         28558  10APR64  13SEP92
1038    F       TA1         26533  09NOV69  23NOV91
1050    M       ME2         35167  14JUL63  24AUG86
1065    M       ME2         35090  26JAN44  07JAN87
1076    M       PT1         66558  14OCT55  03OCT91
1094    M       FA1         22268  02APR70  17APR91
1100    M       BCK         25004  01DEC60  07MAY88
```

Program

Declare the PROCLIB library. The PROCLIB library is used in these examples to store created tables.

```
libname proclib 'SAS-data-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=60;
```

Create the PROCLIB.JOBS view. CREATE VIEW creates the PROC SQL view PROCLIB.JOBS. The PW= data set option assigns password protection to the data that is generated by this view.

```
proc sql;
   create view proclib.jobs(pw=red) as
```

Select the columns. The SELECT clause specifies four columns for the view: Jobcode and three columns, Number, AVGAGE, and AVGSAL, whose values are the products functions. COUNT returns the number of nonmissing values for each job code because the data is grouped by Jobcode. LABEL= assigns a label to the column.

```
      select Jobcode,
             count(jobcode) as number label='Number',
```

Calculate the Avgage and Avgsal columns. The AVG summary function calculates the average age and average salary for each job code.

```
             avg(int((today()-birth)/365.25)) as avgage
                format=2. label='Average Age',
             avg(salary) as avgsal
                format=dollar8. label='Average Salary'
```

Specify the table from which the data is obtained. The FROM clause specifies PAYROLL as the table to select from. PROC SQL assumes the libref of PAYROLL to be PROCLIB because PROCLIB is used in the CREATE VIEW statement.

```
      from payroll
```

Organize the data into groups and specify the groups to include in the output. The GROUP BY clause groups the data by the values of Jobcode. Thus, any summary statistics are calculated for each grouping of rows by value of Jobcode. The HAVING clause subsets the grouped data and returns rows for job codes that contain an average age of greater than or equal to 30.

```
      group by jobcode
      having avgage ge 30;
```

Specify the titles.

```
   title 'Current Summary Information for Each Job Category';
   title2 'Average Age Greater Than or Equal to 30';
```

Display the entire PROCLIB.JOBS view. The SELECT statement selects all columns from PROCLIB.JOBS. PW=RED is necessary because the view is password protected.

```
select * from proclib.jobs(pw=red);
```

Output

```
              Current Summary Information for Each Job Category
                   Average Age Greater Than Or Equal to 30

                                          Average    Average
                     Jobcode    Number        Age     Salary
                     ------------------------------------
                     BCK             9         36    $25,794
                     FA1            11         33    $23,039
                     FA2            16         37    $27,987
                     FA3             7         39    $32,934
                     ME1             8         34    $28,500
                     ME2            14         39    $35,577
                     ME3             7         42    $42,411
                     NA1             5         30    $42,032
                     NA2             3         42    $52,383
                     PT1             8         38    $67,908
                     PT2            10         43    $87,925
                     PT3             2         54    $10,505
                     SCP             7         37    $18,309
                     TA1             9         36    $27,721
                     TA2            20         36    $33,575
                     TA3            12         40    $39,680
```

Example 9: Joining Three Tables

Procedure features:

FROM clause

joined-table component

WHERE clause

Tables: PROCLIB.STAFF2, PROCLIB.SCHEDULE2, PROCLIB.SUPERV2

This example joins three tables and produces a report that contains columns from each table.

Input Tables

PROCLIB.STAFF2

```
                              PROCLIB.STAFF2

 Id
 Num   Lname            Fname            City             State  Hphone
 ----------------------------------------------------------------------------
 1106  MARSHBURN        JASPER           STAMFORD         CT     203/781-1457
 1430  DABROWSKI        SANDRA           BRIDGEPORT       CT     203/675-1647
 1118  DENNIS           ROGER            NEW YORK         NY     718/383-1122
 1126  KIMANI           ANNE             NEW YORK         NY     212/586-1229
 1402  BLALOCK          RALPH            NEW YORK         NY     718/384-2849
 1882  TUCKER           ALAN             NEW YORK         NY     718/384-0216
 1479  BALLETTI         MARIE            NEW YORK         NY     718/384-8816
 1420  ROUSE            JEREMY           PATERSON         NJ     201/732-9834
 1403  BOWDEN           EARL             BRIDGEPORT       CT     203/675-3434
 1616  FUENTAS          CARLA            NEW YORK         NY     718/384-3329
```

PROCLIB.SCHEDULE2

```
                            PROCLIB.SCHEDULE2

                                                  Id
                         Flight     Date  Dest  Num
                         ---------------------------
                         132     01MAR94  BOS   1118
                         132     01MAR94  BOS   1402
                         219     02MAR94  PAR   1616
                         219     02MAR94  PAR   1478
                         622     03MAR94  LON   1430
                         622     03MAR94  LON   1882
                         271     04MAR94  NYC   1430
                         271     04MAR94  NYC   1118
                         579     05MAR94  RDU   1126
                         579     05MAR94  RDU   1106
```

PROCLIB.SUPERV2

```
                    PROCLIB.SUPERV2

              Supervisor         Job
              Id          State  Category
              ---------------------------
              1417        NJ     NA
              1352        NY     NA
              1106        CT     PT
              1442        NJ     PT
              1118        NY     PT
              1405        NJ     SC
              1564        NY     SC
              1639        CT     TA
              1126        NY     TA
              1882        NY     ME
```

Program

Declare the PROCLIB library. The PROCLIB library is used in these examples to store created tables.

```
libname proclib 'SAS-data-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=60;
```

Select the columns. The SELECT clause specifies the columns to select. IdNum is prefixed with a table alias because it appears in two tables.

```
proc sql;
title 'All Flights for Each Supervisor';
select s.IdNum, Lname, City 'Hometown', Jobcat,
       Flight, Date
```

Specify the tables to include in the join. The FROM clause lists the three tables for the join and assigns an alias to each table.

```
from proclib.schedule2 s, proclib.staff2 t, proclib.superv2 v
```

Specify the join criteria. The WHERE clause specifies the columns that join the tables. The STAFF2 and SCHEDULE2 tables have an IdNum column, which has related values in both tables. The STAFF2 and SUPERV2 tables have the IdNum and SUPID columns, which have related values in both tables.

```
where s.idnum=t.idnum and t.idnum=v.supid;
```

Output

```
                    All Flights for Each Supervisor

      Id                                 Job
      Num   Lname            Hometown         Category  Flight     Date
      -----------------------------------------------------------------
      1106  MARSHBURN        STAMFORD         PT        579     05MAR94
      1118  DENNIS           NEW YORK         PT        132     01MAR94
      1118  DENNIS           NEW YORK         PT        271     04MAR94
      1126  KIMANI           NEW YORK         TA        579     05MAR94
      1882  TUCKER           NEW YORK         ME        622     03MAR94
```

Example 10: Querying an In-Line View

Procedure features:

FROM clause

in-line view

Tables: PROCLIB.STAFF2, PROCLIB.SCHEDULE2, PROCLIB.SUPERV2

This example shows an alternative way to construct the query that is explained in Example 9 on page 1145 by joining one of the tables with the results of an in-line view. The example also shows how to rename columns with an in-line view.

Program

Declare the PROCLIB library. The PROCLIB library is used in these examples to store created tables.

```
libname proclib 'SAS-data-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=60;
```

Select the columns. The SELECT clause selects all columns that are returned by the in-line view (which will have the alias Three assigned to it), plus one column from the third table (which will have the alias V assigned to it).

```
proc sql;
   title 'All Flights for Each Supervisor';
   select three.*, v.jobcat
```

Specify the in-line query. Instead of including the name of a table or view, the FROM clause includes a query that joins two of the three tables. In the in-line query, the SELECT clause lists the columns to select. IdNum is prefixed with a table alias because it appears in both tables. The FROM clause lists the two tables for the join and assigns an alias to each table. The WHERE clause specifies the columns that join the tables. The STAFF2 and SCHEDULE2 tables have an IdNum column, which has related values in both tables.

```
      from (select lname, s.idnum, city, flight, date
               from proclib.schedule2 s, proclib.staff2 t
               where s.idnum=t.idnum)
```

Specify an alias for the query and names for the columns. The alias Three refers to the results of the in-line view. The names in parentheses become the names for the columns in the view.

```
                as three (Surname, Emp_ID, Hometown,
                           FlightNumber, FlightDate),
```

Join the results of the in-line view with the third table. The WHERE clause specifies the columns that join the table with the in-line view. Note that the WHERE clause specifies the renamed Emp_ID column from the in-line view.

```
      proclib.superv2 v
      where three.Emp_ID=v.supid;
```

Output

```
                       All Flights for Each Supervisor                        1

                                                                    Job
  Surname          Emp_ID  Hometown         FlightNumber  FlightDate  Category
  ----------------------------------------------------------------------------
  MARSHBURN        1106    STAMFORD         579              05MAR94  PT
  DENNIS           1118    NEW YORK         132              01MAR94  PT
  DENNIS           1118    NEW YORK         271              04MAR94  PT
  KIMANI           1126    NEW YORK         579              05MAR94  TA
  TUCKER           1882    NEW YORK         622              03MAR94  ME
```

Example 11: Retrieving Values with the SOUNDS-LIKE Operator

Procedure features:

ORDER BY clause

SOUNDS-LIKE operator

Table: PROCLIB.STAFF

This example returns rows based on the functionality of the SOUNDS-LIKE operator in a WHERE clause.

Note: The SOUNDS-LIKE operator is based on the SOUNDEX algorithm for identifying words that sound alike. The SOUNDEX algorithm is English-biased and is less useful for languages other than English. For more information on the SOUNDEX algorithm, see *SAS Language Reference: Dictionary*. △

Input Table

PROCLIB.STAFF

```
                              PROCLIB.STAFF
                           First 10 Rows Only

  Id
  Num   Lname             Fname            City             State  Hphone
  ----------------------------------------------------------------------------
  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
```

Program

Declare the PROCLIB library. The PROCLIB library is used in these examples to store created tables.

```
libname proclib 'SAS-data-library';
```

```
options nodate pageno=1 linesize=80 pagesize=60;
```

Select the columns and the table from which the data is obtained. The SELECT clause selects all columns from the table in the FROM clause, PROCLIB.STAFF.

```
proc sql;
   title "Employees Whose Last Name Sounds Like 'Johnson'";
   select idnum, upcase(lname), fname
      from proclib.staff
```

Subset the query and sort the output. The WHERE clause uses the SOUNDS-LIKE operator to subset the table by those employees whose last name sounds like **Johnson**. The ORDER BY clause orders the output by the second column.

```
      where lname=*"Johnson"
      order by 2;
```

Output

```
                Employees Whose Last Name Sounds Like 'Johnson'                1

                     Id
                     Num                    Fname
                     --------------------------------------
                     1411  JOHNSEN          JACK
                     1113  JOHNSON          LESLIE
                     1369  JONSON           ANTHONY
```

SOUNDS-LIKE is useful, but there might be instances where it does not return every row that seems to satisfy the condition. PROCLIB.STAFF has an employee with the last name **SANDERS** and an employee with the last name **SANYERS**. The algorithm does not find **SANYERS**, but it does find **SANDERS** and **SANDERSON**.

```
   title "Employees Whose Last Name Sounds Like 'Sanders'";
   select *
      from proclib.staff
      where lname=*"Sanders"
      order by 2;
```

```
                Employees Whose Last Name Sounds Like 'Sanders'                2

  Id
  Num   Lname             Fname             City              State  Hphone
  ----------------------------------------------------------------------------
  1561  SANDERS           RAYMOND           NEW YORK          NY     212/588-6615
  1414  SANDERSON         NATHAN            BRIDGEPORT        CT     203/675-1715
  1434  SANDERSON         EDITH             STAMFORD          CT     203/781-1333
```

Example 12: Joining Two Tables and Calculating a New Value

Procedure features:

GROUP BY clause

HAVING clause

SELECT clause

ABS function
FORMAT= column-modifier
LABEL= column-modifier
MIN summary function
** operator, exponentiation
SQRT function

Tables: STORES, HOUSES

This example joins two tables in order to compare and analyze values that are unique to each table yet have a relationship with a column that is common to both tables.

```
options ls=80 ps=60 nodate pageno=1 ;
data stores;
  input Store $ x y;
  datalines;
store1 5 1
store2 5 3
store3 3 5
store4 7 5
;
data houses;
   input House $ x y;
   datalines;
house1 1 1
house2 3 3
house3 2 3
house4 7 7
;
```

Input Tables

STORES and HOUSES

The tables contain X and Y coordinates that represent the location of the stores and houses.

```
                          STORES Table                               1
                      Coordinates of Stores

                   Store           x         y
                   ----------------------------
                   store1          6         1
                   store2          5         2
                   store3          3         5
                   store4          7         5
```

```
                         HOUSES Table                                2
                    Coordinates of Houses

               House             x          y
               -----------------------------
               house1            1          1
               house2            3          3
               house3            2          3
               house4            7          7
```

Program

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 query. The SELECT clause specifies three columns: HOUSE, STORE, and DIST. The arithmetic expression uses the square root function (SQRT) to create the values of DIST, which contain the distance from HOUSE to STORE for each row. The double asterisk (**) represents exponentiation. LABEL= assigns a label to STORE and to DIST.

```
proc sql;
   title 'Each House and the Closest Store';
   select house, store label='Closest Store',
          sqrt((abs(s.x-h.x)**2)+(abs(h.y-s.y)**2)) as dist
              label='Distance' format=4.2
      from stores s, houses h
```

Organize the data into groups and subset the query. The minimum distance from each house to all the stores is calculated because the data are grouped by house. The HAVING clause specifies that each row be evaluated to determine if its value of DIST is the same as the minimum distance from that house to any store.

```
      group by house
      having dist=min(dist);
```

Output

Note that two stores are tied for shortest distance from house2.

```
                    Each House and the Closest Store                     1

                            Closest
                      House     Store     Distance
                      ----------------------------
                      house1    store1        4.00
                      house2    store2        2.00
                      house2    store3        2.00
                      house3    store3        2.24
                      house4    store4        2.00
```

Example 13: Producing All the Possible Combinations of the Values in a Column

Procedure features:

- CASE expression
- joined-table component
- Cross join
- SELECT clause
    - DISTINCT keyword

Tables: PROCLIB.MARCH, FLIGHTS

This example joins a table with itself to get all the possible combinations of the values in a column.

Input Table

PROCLIB.MARCH (Partial Listing)

```
                         PROCLIB.MARCH                        1
                       First 10 Rows Only

 Flight     Date  Depart  Orig  Dest     Miles   Boarded  Capacity
 -----------------------------------------------------------------
 114     01MAR94    7:10  LGA   LAX       2475       172       210
 202     01MAR94   10:43  LGA   ORD        740       151       210
 219     01MAR94    9:31  LGA   LON       3442       198       250
 622     01MAR94   12:19  LGA   FRA       3857       207       250
 132     01MAR94   15:35  LGA   YYZ        366       115       178
 271     01MAR94   13:17  LGA   PAR       3635       138       250
 302     01MAR94   20:22  LGA   WAS        229       105       180
 114     02MAR94    7:10  LGA   LAX       2475       119       210
 202     02MAR94   10:43  LGA   ORD        740       120       210
 219     02MAR94    9:31  LGA   LON       3442       147       250
```

Program to Create the Flights Table

Declare the PROCLIB library. The PROCLIB library is used in these examples to store created tables.

```
libname proclib 'SAS-data-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=60;
```

Create the FLIGHTS table. The CREATE TABLE statement creates the table FLIGHTS from the output of the query. The SELECT clause selects the unique values of Dest. DISTINCT specifies that only one row for each value of city be returned by the query and stored in the table FLIGHTS. The FROM clause specifies PROCLIB.MARCH as the table to select from.

```
proc sql;
   create table flights as
      select distinct dest
         from proclib.march;
```

Specify the title.

```
title 'Cities Serviced by the Airline';
```

Display the entire FLIGHTS table.

```
select * from flights;
```

Output

FLIGHTS Table

```
                     Cities Serviced by the Airline                    1

                                 Dest
                                 ----
                                 FRA
                                 LAX
                                 LON
                                 ORD
                                 PAR
                                 WAS
                                 YYZ
```

Program Using Conventional Join

Specify the title.

```
title 'All Possible Connections';
```

Select the columns. The SELECT clause specifies three columns for the output. The prefixes on DEST are table aliases to specify which table to take the values of Dest from. The CASE expression creates a column that contains the character string **`to and from`**.

```
select f1.Dest, case
                   when f1.dest ne ' ' then 'to and from'
                end,
       f2.Dest
```

Specify the type of join. The FROM clause joins FLIGHTS with itself and creates a table that contains every possible combination of rows (a Cartesian product). The table contains two rows for each possible route, for example, **`PAR <-> WAS`** and **`WAS <-> PAR`**.

```
from flights as f1, flights as f2
```

Specify the join criterion. The WHERE clause subsets the internal table by choosing only those rows where the name in F1.Dest sorts before the name in F2.Dest. Thus, there is only one row for each possible route.

```
where f1.dest < f2.dest
```

Sort the output. ORDER BY sorts the result by the values of F1.Dest.

```
order by f1.dest;
```

Output

```
                All Possible Connections                     2

                Dest               Dest
                -----------------------
                FRA   to and from  LAX
                FRA   to and from  LON
                FRA   to and from  WAS
                FRA   to and from  ORD
                FRA   to and from  PAR
                FRA   to and from  YYZ
                LAX   to and from  LON
                LAX   to and from  PAR
                LAX   to and from  WAS
                LAX   to and from  ORD
                LAX   to and from  YYZ
                LON   to and from  ORD
                LON   to and from  WAS
                LON   to and from  PAR
                LON   to and from  YYZ
                ORD   to and from  WAS
                ORD   to and from  PAR
                ORD   to and from  YYZ
                PAR   to and from  WAS
                PAR   to and from  YYZ
                WAS   to and from  YYZ
```

Program Using Cross Join

Specify a cross join. Because a cross join is functionally the same as a Cartesian product join, the cross join syntax can be substituted for the conventional join syntax.

```
proc sql;
   title 'All Possible Connections';
   select f1.Dest, case
                      when f1.dest ne ' ' then 'to and from'
                   end,
       f2.Dest
       from flights as f1 cross join flights as f2
       where f1.dest < f2.dest
       order by f1.dest;
```

Output

```
                    All Possible Connections                          1

                    Dest              Dest
                    -----------------------
                    FRA   to and from  LAX
                    FRA   to and from  LON
                    FRA   to and from  WAS
                    FRA   to and from  ORD
                    FRA   to and from  PAR
                    FRA   to and from  YYZ
                    LAX   to and from  LON
                    LAX   to and from  PAR
                    LAX   to and from  WAS
                    LAX   to and from  ORD
                    LAX   to and from  YYZ
                    LON   to and from  ORD
                    LON   to and from  WAS
                    LON   to and from  PAR
                    LON   to and from  YYZ
                    ORD   to and from  WAS
                    ORD   to and from  PAR
                    ORD   to and from  YYZ
                    PAR   to and from  WAS
                    PAR   to and from  YYZ
                    WAS   to and from  YYZ
```

Example 14: Matching Case Rows and Control Rows

Procedure features:
joined-table component

Tables: MATCH_11 on page 1388, MATCH

This example uses a table that contains data for a case-control study. Each row contains information for a case or a control. To perform statistical analysis, you need a table with one row for each case-control pair. PROC SQL joins the table with itself in order to match the cases with their appropriate controls. After the rows are matched, differencing can be performed on the appropriate columns.

The input table MATCH_11 contains one row for each case and one row for each control. Pair contains a number that associates the case with its control. Low is 0 for the controls and 1 for the cases. The remaining columns contain information about the cases and controls.

Input Table

MATCH_11 Table 1
First 10 Rows Only

Pair	Low	Age	Lwt	Race	Smoke	Ptd	Ht	UI	race1	race2
1	0	14	135	1	0	0	0	0	0	0
1	1	14	101	3	1	1	0	0	0	1
2	0	15	98	2	0	0	0	0	1	0
2	1	15	115	3	0	0	0	1	0	1
3	0	16	95	3	0	0	0	0	0	1
3	1	16	130	3	0	0	0	0	0	1
4	0	17	103	3	0	0	0	0	0	1
4	1	17	130	3	1	1	0	1	0	1
5	0	17	122	1	1	0	0	0	0	0
5	1	17	110	1	1	0	0	0	0	0

Program

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 MATCH table. The SELECT clause specifies the columns for the table MATCH. SQL expressions in the SELECT clause calculate the differences for the appropriate columns and create new columns.

```
proc sql;
   create table match as
      select
         one.Low,
         one.Pair,
         (one.lwt - two.lwt) as Lwt_d,
         (one.smoke - two.smoke) as Smoke_d,
         (one.ptd - two.ptd) as Ptd_d,
         (one.ht - two.ht) as Ht_d,
         (one.ui - two.ui) as UI_d
```

Specify the type of join and the join criterion. The FROM clause lists the table MATCH_11 twice. Thus, the table is joined with itself. The WHERE clause returns only the rows for each pair that show the difference when the values for control are subtracted from the values for case.

```
      from match_11 one, match_11 two
      where (one.pair=two.pair and one.low>two.low);
```

Specify the title.

```
title 'Differences for Cases and Controls';
```

Display the first five rows of the MATCH table. The SELECT clause selects all the columns from MATCH. The OBS= data set option limits the printing of the output to five rows.

```
select *
   from match(obs=5);
```

Output

MATCH Table

```
                 Differences for Cases and Controls                   1

      Low      Pair     Lwt_d   Smoke_d     Ptd_d      Ht_d      UI_d
  ---------------------------------------------------------------------
        1         1       -34         1         1         0         0
        1         2        17         0         0         0         1
        1         3        35         0         0         0         0
        1         4        27         1         1         0         1
        1         5       -12         0         0         0         0
```

Example 15: Counting Missing Values with a SAS Macro

Procedure feature:
COUNT function

Table: SURVEY

This example uses a SAS macro to create columns. The SAS macro is not explained here. See *SAS Macro Language: Reference* for information on SAS macros.

Input Table

SURVEY contains data from a questionnaire about diet and exercise habits. SAS enables you to use a special notation for missing values. In the EDUC column, the **.x** notation indicates that the respondent gave an answer that is not valid, and **.n** indicates that the respondent did not answer the question. A period as a missing value indicates a data entry error.

```
data survey;
  input id $ diet $ exer $ hours xwk educ;
  datalines;
1001 yes yes 1 3 1
1002 no  yes 1 4 2
1003 no  no  . . .n
1004 yes yes 2 3 .x
1005 no  yes 2 3 .x
1006 yes yes 2 4 .x
1007 no  yes .5 3 .
1008 no  no  . .  .
;
```

Program

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;
```

Count the nonmissing responses. The COUNTM macro uses the COUNT function to perform various counts for a column. Each COUNT function uses a CASE expression to select the rows to be counted. The first COUNT function uses only the column as an argument to return the number of nonmissing rows.

```
 %macro countm(col);
    count(&col) "Valid Responses for &col",
```

Count missing or invalid responses. The NMSS function returns the number of rows for which the column has any type of missing value: **.n**, **.x**, or a period.

```
   nmiss(&col) "Missing or NOT VALID Responses for &col",
```

Count the occurrences of various sources of missing or invalid responses. The last three COUNT functions use CASE expressions to count the occurrences of the three notations for missing values. The "count me" character string gives the COUNT function a nonmissing value to count.

```
   count(case
            when &col=.n  then "count me"
            end) "Coded as NO ANSWER for &col",
   count(case
            when &col=.x  then "count me"
            end) "Coded as NOT VALID answers for &col",
   count(case
            when &col=.  then "count me"
            end) "Data Entry Errors for &col"
%mend;
```

Use the COUNTM macro to create the columns. The SELECT clause specifies the columns that are in the output. COUNT(*) returns the total number of rows in the table. The COUNTM macro uses the values of the EDUC column to create the columns that are defined in the macro.

```
proc sql;
   title 'Counts for Each Type of Missing Response';
   select count(*)  "Total No. of Rows",
          %countm(educ)
      from survey;
```

Output

```
                   Counts for Each Type of Missing Response                  1

                              Missing            Coded as
                               or NOT  Coded as       NOT      Data
           Total      Valid     VALID        NO     VALID     Entry
          No. of  Responses Responses    ANSWER   answers    Errors
            Rows   for educ  for educ  for educ  for educ  for educ
        ------------------------------------------------------------
               8          2         6         1         3         2
```

CHAPTER

45

The 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

Output 45.1 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 45.1 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
```

Output 45.2 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

- 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 on page 1173.

Output 45.2 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

Tip: Supports the Output Delivery System. See "Output Delivery System" on page 32 for details.

ODS Table Name: Standard

Reminder: You can use the ATTRIB, FORMAT, LABEL, and WHERE statements. See Chapter 3, "Statements with the Same Function in Multiple Procedures," on page 57 for details. You can also use any global statements. See "Global Statements" on page 18 for a list.

PROC STANDARD <*option(s)*>;

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

  FREQ *variable*;

  VAR *variable(s)*;

  WEIGHT *variable*;

To do this	Use this statement
Calculate separate standardized values for each BY group	BY
Identify a variable whose values represent the frequency of each observation	FREQ

To do this	Use this statement
Select the variables to standardize and determine the order in which they appear in the printed output	VAR
Identify a variable whose values weight each observation in the statistical calculations	WEIGHT

PROC STANDARD Statement

PROC STANDARD <*option(s)*>;

To do this	Use this option
Specify the input data set	DATA=
Specify the output data set	OUT=
Computational options	
Exclude observations with nonpositive weights	EXCLNPWGT
Specify the mean value	MEAN=
Replace missing values with a variable mean or MEAN= value	REPLACE
Specify the standard deviation value	STD=
Specify the divisor for variance calculations	VARDEF=
Control printed output	
Print statistics for each variable to standardize	PRINT

Without Options

If you do not specify MEAN=, REPLACE, or STD=, the output data set is an identical copy of the input data set.

Options

DATA=*SAS-data-set*
identifies the input SAS data set.

Main discussion: "Input Data Sets" on page 19

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.

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.

MEAN=*mean-value*
standardizes variables to a mean of *mean-value*.

Alias: M=

Default: mean of the input values

Featured in: Example 1 on page 1171

OUT=*SAS-data-set*
identifies 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*

Featured in: Example 1 on page 1171

PRINT
prints the original frequency, mean, and standard deviation for each variable to standardize.

Featured in: Example 2 on page 1173

REPLACE
replaces missing values with the variable mean.

Interaction: If you use MEAN=, PROC STANDARD replaces missing values with the given mean.

Featured in: Example 2 on page 1173

STD=*std-value*
standardizes variables to a standard deviation of *std-value*.

Alias: S=

Default: standard deviation of the input values

Featured in: Example 1 on page 1171

VARDEF=*divisor*
specifies the divisor to use in the calculation of variances and standard deviation. Table 45.1 on page 1167 shows the possible values for *divisor* and the associated divisors.

Table 45.1 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 $\sum (x_i - \overline{x})^2$. When you weight the analysis variables, CSS equals $\sum w_i (x_i - \overline{x}_w)^2$ where $\overline{x}_w$ is the weighted mean.

Default: DF

Tip: 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.

Tip: When you use the WEIGHT statement and VARDEF=WGT, the computed variance is asymptotically (for large n) an estimate of $\sigma^2/\overline{w}$, where $\overline{w}$ is the average weight. This yields an asymptotic estimate of the variance of an observation with average weight.

See also: “WEIGHT” on page 63

Main discussion: “Keywords and Formulas” on page 1340

BY Statement

Calculates standardized values separately for each BY group.

Main discussion: “BY” on page 58

Featured in: Example 2 on page 1173

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, 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*.

Options

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 also: For an example that uses the FREQ statement, see "FREQ" on page 61

FREQ *variable*;

Required Arguments

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.

Featured in: Example 1 on page 1171

VAR *variable(s)*;

Required Arguments

variable(s)
identifies one or more variables to standardize.

WEIGHT Statement

Specifies weights for analysis variables in the statistical calculations.

See also: For information about calculating weighted statistics and for an example that uses the WEIGHT statement, see "WEIGHT" on page 63

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. If the value of the weight variable is

Weight value…	PROC STANDARD…
0	counts the observation in the total number of observations
less than 0	converts the weight value to zero and counts the observation in the total number of observations
missing	excludes the observation from the calculation of mean and standard deviation

To exclude observations that contain negative and zero weights from the calculation of mean and standard deviation, use EXCLNPWGT. Note that most SAS/STAT procedures, such as PROC GLM, exclude negative and zero weights by default.

Tip: When you use the WEIGHT statement, consider which value of the VARDEF= option is appropriate. See VARDEF= on page 1167 and the calculation of weighted statistics in “Keywords and Formulas” on page 1340 for more information.

Note: Prior to Version 7 of the SAS System, the procedure did not exclude the observations with missing weights from the count of observations. △

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.

Statistical Computations: STANDARD Procedure

Standardizing values removes the location and scale attributes from a set of data. The formula to compute standardized values is

$$x_i' = \frac{S * (x_i - \overline{x})}{s_x} + M$$

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
$\overline{x}$	is a variable's mean
s_x	is a variable's standard deviation.

PROC STANDARD calculates the mean ($\overline{x}$) 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.

Examples: STANDARD Procedure

Example 1: Standardizing to a Given Mean and Standard Deviation

Procedure features:

PROC STANDARD statement options:

- MEAN=
- OUT=
- STD=

VAR statement

Other features:

PRINT procedure

This example

- □ 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

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 Z*w.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;
```

Output

The 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                           Std                 Std
 Student     Number     Section     Test1     Test1     Test2     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

Procedure features:

PROC STANDARD statement options:

PRINT
REPLACE

BY statement

Other features:

FORMAT procedure
PRINT procedure
SORT procedure

This example

- calculates Z scores separately for each BY group using a mean of 1 and standard deviation of 0
- replaces missing values with the given mean
- prints the mean and standard deviation for the variables to standardize
- prints the output data set.

Program

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 on 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 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
```

* Data are from *Vital Signs 1994: The Trends That Are Shaping Our Future*, Lester R. Brown, Hal Kane, and David Malin Roodman, eds. Copyright © 1994 by Worldwatch Institute. Reprinted by permission of W.W. Norton & Company, Inc.

```
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;
```

Output

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 -----------------------------

                              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                    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

46

The 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 27, "The MEANS Procedure," on page 523. Except for the differences that are discussed here, all the PROC MEANS information also applies to PROC SUMMARY.

Syntax: SUMMARY Procedure

Tip: Supports the Output Delivery System. See "Output Delivery System" on page 32 for details.

ODS Table Name: Summary

Reminder: You can use the ATTRIB, FORMAT, LABEL, and WHERE statements. See Chapter 3, "Statements with the Same Function in Multiple Procedures," on page 57 for details. You can also use any global statements. See "Global Statements" on page 18 for a list.

Reminder: Full syntax descriptions are in "Syntax: MEANS Procedure" on page 526.

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*;

PROC SUMMARY Statement

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 on 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.

CHAPTER

47

The 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

- 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

Output 47.1 shows a simple table that was produced by PROC TABULATE. The data set“ENERGY” on page 1387 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 headers without hyphenating them.)

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

proc tabulate data=energy;
   class region division type;
```

```
   var expenditures;
   table region*division, type*expenditures /
         rts=20;
run;
```

Output 47.1 Simple Table Produced by PROC TABULATE

```
                    The SAS System                     1

        ---------------------------------------------
        |                  |            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

Output 47.2 is a more complicated table using the same data set that was used to create Output 47.1. The statements that create this report

- □ customize column and row headers
- □ 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 on page 1246.

Output 47.2 Complex Table Produced by PROC TABULATE

```
              Energy Expenditures for Each Region            2
                     (millions of dollars)

 ----------------------------------------------------------------
 |                        |      Customer Base       |            |
 |                        |--------------------------|            |
 |                        |Residential |  Business   |    All     |
 |                        | Customers  | Customers   | 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

Display 47.1 on page 1183 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 on page 1279.

Display 47.1 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

Terminology: TABULATE Procedure

The following figures illustrate some of the terms that are commonly used in discussions of PROC TABULATE.

Figure 47.1 Parts of a PROC TABULATE Table

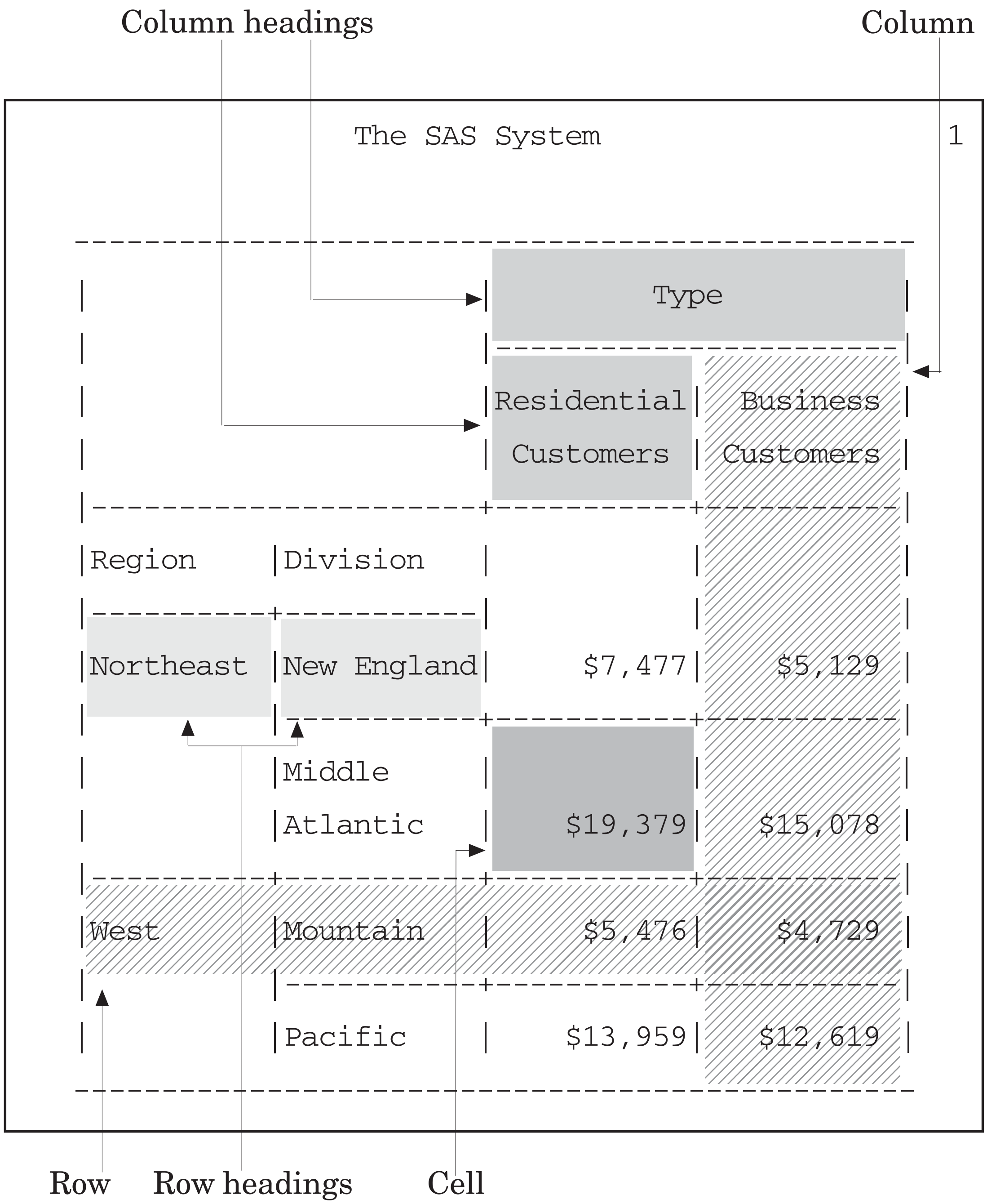
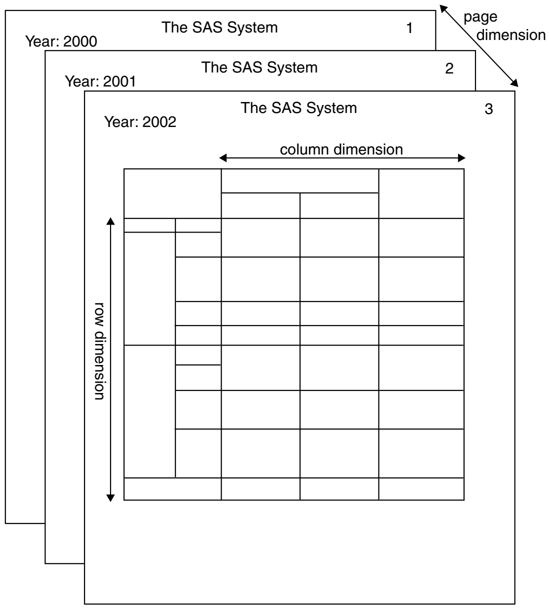


Figure 47.2 PROC TABULATE Table Dimensions



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 47.1 on page 1184 contains three class variables: Region, Division, and Type. These class variables form the eight categories listed in Table 47.1 on page 1185. (For convenience, the categories are described in terms of their formatted values.)

Table 47.1 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 47.1 on page 1184, 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 47.2 on page 1185, 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 STYLE= on page 1194 in the PROC TABULATE statement and "Output Delivery System" on page 32.

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 47.1 on page 1184 contains no subtables. For an illustration of a table that is composed of multiple subtables, see Figure 47.18 on page 1274.

Syntax: TABULATE Procedure

Requirements: At least one TABLE statement is required.

Requirements: Depending on the variables that appear in the TABLE statement, a CLASS statement, a VAR statement, or both are required.

Tip: Supports the Output Delivery System. See "Output Delivery System" on page 32 for details.

ODS Table Name: Table

Reminder: You can use the ATTRIB, FORMAT, LABEL, and WHERE statements. See Chapter 3, "Statements with the Same Function in Multiple Procedures," on page 57 for details. You can also use any global statements. See "Global Statements" on page 18 for a list.

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*;

To do this	Use this statement
Create a separate table for each BY group	BY
Identify variables in the input data set as class variables	CLASS
Specify a style for class variable level value headings	CLASSLEV
Identify a variable in the input data set whose values represent the frequency of each observation	FREQ
Specify a label for a keyword	KEYLABEL
Specify a style for keyword headings	KEYWORD
Describe the table to create	TABLE
Identify variables in the input data set as analysis variables	VAR
Identify a variable in the input data set whose values weight each observation in the statistical calculations	WEIGHT

PROC TABULATE Statement

PROC TABULATE <*option(s)*>;

To do this	Use this option
Customize the HTML contents link to the output	CONTENTS=
Specify the input data set	DATA=
Specify the output data set	OUT=
Override the SAS system option THREADS | NOTHREADS	THREADS | NOTHREADS
Enable floating point exception recovery	TRAP
Identify categories of data that are of interest	
Specify a secondary data set that contains the combinations of values of class variables to include in tables and output data sets	CLASSDATA=
Exclude from tables and output data sets all combinations of class variable values that are not in the CLASSDATA= data set	EXCLUSIVE
Consider missing values as valid values for class variables	MISSING
Control the statistical analysis	
Specify the confidence level for the confidence limits	ALPHA=

To do this	Use this option
Exclude observations with nonpositive weights	EXCLNPWGTS
Specify the sample size to use for the P^2 quantile estimation method	QMARKERS=
Specify the quantile estimation method	QMETHOD=
Specify the mathematical definition to calculate quantiles	QNTLDEF=
Specify the variance divisor	VARDEF=
Customize the appearance of the table	
Specify a default format for each cell in the table	FORMAT=
Define the characters to use to construct the table outlines and dividers	FORMCHAR=
Eliminate horizontal separator lines from the row titles and the body of the table	NOSEPS
Order the values of a class variable according to the specified order	ORDER=
Specify the default style element or style elements (for the Output Delivery System) to use for each cell of the table	STYLE=

Options

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.

Featured in: Example 2 on page 1235

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.

Main Discussion: "Input Data Sets" on page 19

EXCLNPWGTS
excludes observations with nonpositive weight values (zero or negative) from the analysis. By default, PROC TABULATE treats observations with negative weights like those with zero weights and counts them in the total number of observations.

Alias: EXCLNPWGT

See also: WEIGHT= on page 1212 and "WEIGHT Statement" on page 1212

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.

Featured in: Example 2 on page 1235

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.

Featured in: Example 1 on page 1232 and Example 6 on page 1246

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.

Range: PROC TABULATE uses 11 of the 20 formatting characters that SAS provides. Table 47.2 on page 1190 shows the formatting characters that PROC TABULATE uses. Figure 47.3 on page 1191 illustrates the use of each formatting character in the output from PROC TABULATE.

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 that 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.

Restriction: The FORMCHAR= option affects only the traditional SAS monospace output destination.

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 instance, 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
```

Tip: 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)
```

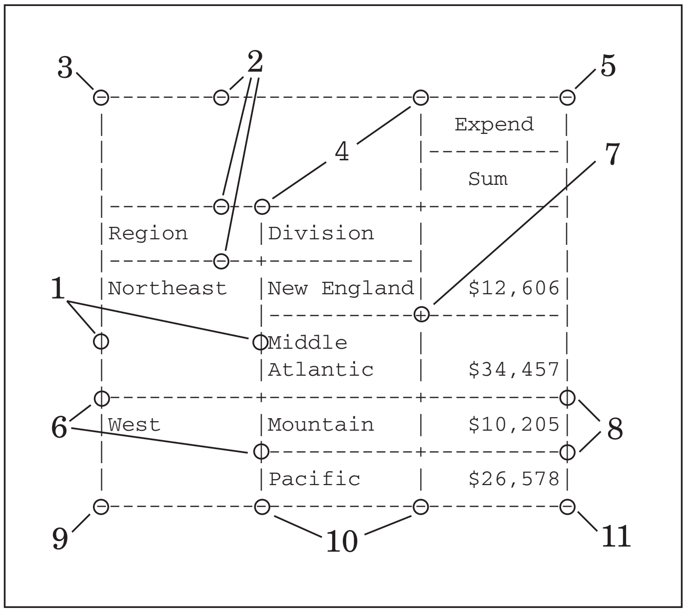
See also: For more information about formatting output, see Chapter 5, "Controlling the Table's Appearance," in the *SAS Guide to TABULATE Processing*.

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

Table 47.2 Formatting Characters Used by PROC TABULATE

Position	Default	Used to draw
1	|	the right and left borders and the vertical separators between columns
2	-	the top and bottom borders and the horizontal separators between rows
3	-	the top character in the left border
4	-	the top character in a line of characters that separate columns
5	-	the top character in the right border
6	|	the leftmost character in a row of horizontal separators
7	+	the intersection of a column of vertical characters and a row of horizontal characters
8	|	the rightmost character in a row of horizontal separators
9	-	the bottom character in the left border
10	-	the bottom character in a line of characters that separate columns
11	-	the bottom character in the right border

Figure 47.3 Formatting Characters in PROC TABULATE Output



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.

Main Discussion: "Including Observations with Missing Class Variables" on page 1225

See also: *SAS Language Reference: Concepts* 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 headers.

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 the FORMCHAR= option on page 1189.

Featured in: Example 8 on page 1251

NOTHREADS

See THREADS | NOTHREADS on page 1195.

NOTRAP

See TRAP | NOTRAP on page 1195.

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.

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 that you define them.

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.

Featured in: "Understanding the Order of Headings with ORDER=DATA" on page 1230

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 this 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; if it is not, then 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 1's 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 1's and 0's.

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.

Featured in: Example 3 on page 1237

PCTLDEF=
See QNTLDEF= on page 1193.

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.

Main Discussion: "Quantiles" on page 555

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. This is the technique that PROC UNIVARIATE uses.

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 does not compute weighted quantiles.

Tip: When QMETHOD=P2, reliable estimates of some quantiles (P1, P5, P95, P99) may not be possible for some types of data.

Main Discussion: "Quantiles" on page 555

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.

Default: 5

Alias: PCTLDEF=

Main discussion: "Quantile and Related Statistics" on page 1345

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=[background=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 also: See *SAS Output Delivery System: User's Guide* for information about PROC TEMPLATE and the default style definitions.

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: 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 statement that uses STYLE=, except for the TABLE statement). Note that not all attributes are valid in all destinations.

ASIS=	FONT_WIDTH=
BACKGROUND=	HREFTARGET=
BACKGROUNDIMAGE=	HTMLCLASS=
BORDERCOLOR=	JUST=
BORDERCOLORDARK=	NOBREAKSPACE=
BORDERCOLORLIGHT=	POSTHTML=
BORDERWIDTH=	POSTIMAGE=
CELLHEIGHT=	POSTTEXT=
CELLWIDTH=	PREHTML=
FLYOVER=	PREIMAGE=

FONT=	PRETEXT=
FONT_FACE=	PROTECTSPECIALCHARS=
FONT_SIZE=	TAGATTR=
FONT_STYLE=	URL=
FONT_WEIGHT=	VJUST=

style-attribute-value
specifies a value for the attribute. Each attribute has a different set of valid values. See *SAS Output Delivery System: User's Guide* for more information about these style attributes, their valid values, and their applicable destinations.

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 also: "Using Style Elements in PROC TABULATE" on page 1220

Featured in: Example 14 on page 1279

THREADS | NOTHREADS
enables or disables parallel processing of the input data set. This option overrides the SAS system option THREADS | NOTHREADS. See *SAS Language Reference: Concepts* for more information about parallel processing.

Default: value of SAS system option THREADS | NOTHREADS.

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 equal to 1. In those cases, you can use THREADS in the PROC TABULATE statement to force PROC TABULATE to use parallel processing.

TRAP | NOTRAP
enables or disables floating point exception (FPE) recovery during data processing beyond that provided by normal SAS FPE handling, which terminates PROC TABULATE in the case of math exceptions. Note that with NOTRAP, normal SAS FPE handling is still in effect so that PROC TABULATE terminates in the case of math exceptions.

Default: NOTRAP

VARDEF=*divisor*
specifies the divisor to use in the calculation of the variance and standard deviation. Table 47.3 on page 1195 shows the possible values for *divisor* and the associated divisors.

Table 47.3 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 $\sum (x_i - \overline{x})^2$. When you weight the analysis variables, CSS equals $\sum w_i (x_i - \overline{x}_w)^2$ where $\overline{x}_w$ is the weighted mean.

Default: DF

Requirement: To compute standard error of the mean, use the default value of VARDEF=.

Tip: 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.

Tip: When you use the WEIGHT statement and VARDEF=WGT, the computed variance is asymptotically (for large *n*) an estimate of $\sigma^2/\overline{w}$, where $\overline{w}$ is the average weight. This yields an asymptotic estimate of the variance of an observation with average weight.

See also: "Weighted Statistics Example" on page 65

BY Statement

Creates a separate table on a separate page for each BY group.

Main discussion: "BY" on page 58

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 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*.

Options

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.

CLASS Statement

Identifies class variables for the table. Class variables determine the categories that PROC TABULATE uses to calculate statistics.

Tip: You can use multiple CLASS statements.

Tip: Some CLASS statement options are also available in the PROC TABULATE statement. They affect all CLASS variables rather than just the one(s) that you specify in a CLASS statement.

CLASS *variable(s)* <*/option(s)*>;

Required Arguments

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.

Options

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.

Featured in: Example 3 on page 1237

GROUPINTERNAL
specifies not to apply formats to the class variables when PROC TABULATE groups the values to create combinations of class variables.

Interaction: If you specify the PRELOADFMT option in the CLASS statement, then PROC TABULATE ignores the GROUPINTERNAL option and uses the formatted values.

Interaction: 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 also: *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.

Requirement: You must use PROC FORMAT and the MULTILABEL option in the VALUE statement to create a multilabel format.

Interaction: Using MLF with ORDER=FREQ may not produce the order that you expect for the formatted values.

Interaction: 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 also: The MULTILABEL option on page 449 in the VALUE statement of the FORMAT procedure.

Featured in: Example 4 on page 1242

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). △

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 that 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.

Featured in: "Understanding the Order of Headings with ORDER=DATA" on page 1230

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.

Note: If you specify PRELOADFMT without also specifying EXCLUSIVE, ORDER=DATA, or PRINTMISS, then SAS writes a warning message to the SAS log. △

Interaction: 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.

Interaction: To include all ranges and values of the user-defined formats in the output, use the PRINTMISS option in the TABLE statement.

Note: Use care when you use PRELOADFMT with PRINTMISS. This feature creates all possible combinations of formatted class variables. Some of these combinations may not make sense. △

Featured in: Example 3 on page 1237

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 information about the arguments of this option, and how it is used, see STYLE= on page 1194 in the PROC TABULATE statement.

Note: When you use STYLE= in the CLASS statement, it differs slightly from its use in the PROC TABULATE statement. In the CLASS statement, the parent of the heading is the page dimension text or heading under which the current heading is nested. △

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.

Tip: 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.

Tip: 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.

Featured in: Example 14 on page 1279

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 variable(s) 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.

CLASSLEV *variable(s)* / **STYLE**=<*style-element-name* | PARENT>
[*style-attribute-name=style-attribute-value*<*…*
style-attribute-name=style-attribute-value>] ;

Required Arguments

variable(s)
specifies one or more class variables from the CLASS statement for which you want to specify a style element.

Options

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 on the arguments of this option and how it is used, see STYLE= on page 1194 in the PROC TABULATE statement.

Note: When you use STYLE= in the CLASSLEV statement, it differs slightly from its use in the PROC TABULATE statement. In the CLASSLEV statement, the parent of the heading is the heading under which the current heading is nested. △

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.

Featured in: Example 14 on page 1279

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 also: For an example that uses the FREQ statement, see "FREQ" on page 61.

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 *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.

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 1213 or is the universal class variable ALL (see "Elements That You Can Use in a Dimension Expression" on page 1208).

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.

KEYWORD *keyword(s)* / **STYLE**=<*style-element-name* | PARENT>
[*style-attribute-name=style-attribute-value*<*...*
style-attribute-name=style-attribute-value>] ;

Required Arguments

keyword
is one of the keywords for statistics that is discussed in "Statistics That Are Available in PROC TABULATE" on page 1213 or is the universal class variable ALL (see "Elements That You Can Use in a Dimension Expression" on page 1208).

Options

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 on the arguments of this option and how it is used, see STYLE= on page 1194 in the PROC TABULATE statement.

Note: When you use STYLE= in the KEYWORD statement, it differs slightly from its use in the PROC TABULATE statement. In the KEYWORD statement, the parent of the heading is the heading under which the current heading is nested. △

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.

Featured in: Example 14 on page 1279

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.

Tip: Use multiple TABLE statements to create several tables.

TABLE <<*page-expression*,> *row-expression*,>
column-expression </ *table-option(s)*>;

Required Arguments

column-expression

defines the columns in the table. For information on constructing dimension expressions, see "Constructing Dimension Expressions" on page 1208.

Restriction: A column dimension is the last dimension in a TABLE statement. A row dimension or a row dimension and a page dimension may precede a column dimension.

Options

To do this	Use this option
Add dimensions	
Define the pages in a table	*page-expression*
Define the rows in a table	*row-expression*
Customize the HTML contents entry link to the output	CONTENTS=
Modify the appearance of the table	
Change the order of precedence for specified format modifiers	FORMAT_PRECEDENCE=
Specify a style element for various parts of the table	STYLE=

To do this	Use this option
Change the order of precedence for specified style attribute values	STYLE_PRECEDENCE=
Customize text in the table	
Specify the text to place in the empty box above row titles	BOX=
Supply up to 256 characters to print in table cells that contain missing values	MISSTEXT=
Suppress the continuation message for tables that span multiple physical pages	NOCONTINUED
Modify the layout of the table	
Print as many complete logical pages as possible on a single printed page or, if possible, print 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.	CONDENSE
Create the same row and column headings for all logical pages of the table	PRINTMISS
Customize row headings	
Specify the number of spaces to indent nested row headings	INDENT=
Control allocation of space for row titles within the available space	ROW=
Specify the number of print positions available for row titles	RTSPACE=

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 1194 in the PROC TABULATE statement.

Featured in: Example 9 on page 1253 and Example 14 on page 1279

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.

Restrictions: 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.

Featured in: Example 9 on page 1253

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.

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.

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.

Featured in: Example 8 on page 1251 (with crossings) and Example 9 on page 1253 (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 on the arguments of the STYLE= option and how it is used, see STYLE= on page 1194 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 cell(s).

Featured in: "Providing Text for Cells That Contain Missing Values" on page 1228 and Example 14 on page 1279

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. △

page-expression

defines the pages in a table. For information on constructing dimension expressions, see "Constructing Dimension Expressions" on page 1208.

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.

Featured in: Example 9 on page 1253

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.

Restrictions: If an entire logical page contains only missing values, then that page does not print regardless of the PRINTMISS option.

See also: CLASSDATA= option on page 1188

Featured in: "Providing Headings for All Categories" on page 1227

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

Featured in: Example 7 on page 1249

row-expression

defines the rows in the table. For information on constructing dimension expressions, see "Constructing Dimension Expressions" on page 1208.

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 may precede a row dimension.

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 also: For more information about controlling the space for row titles, see Chapter 5, "Controlling the Table's Appearance," in *SAS Guide to TABULATE Processing*.

Featured in: Example 1 on page 1232

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. For information about the arguments of this option and how it is used, see STYLE= on page 1194 in the PROC TABULATE statement.

Note: The list of attributes that you can set or change with the STYLE= option in the TABLE statement differs from that of the PROC TABULATE statement. △

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 *SAS Output Delivery System: User's Guide* for more information about these style attributes, their valid values, and their applicable destinations.

BACKGROUND=	FONT_WIDTH=*
BACKGROUNDIMAGE=	FOREGROUND=*
BORDERCOLOR=	FRAME=
BORDERCOLORDARK=	HTMLCLASS=
BORDERCOLORLIGHT=	JUST=
BORDERWIDTH=	OUTPUTWIDTH=
CELLPADDING=	POSTHTML=
CELLSPACING=	POSTIMAGE=
FONT=*	POSTTEXT=
FONT_FACE=*	PREHTML=
FONT_SIZE=*	PREIMAGE=
FONT_STYLE=*	PRETEXT=
FONT_WEIGHT=*	RULES=

* When you use these attributes in this location, they affect only the text that is specified with the PRETEXT=, POSTTEXT=, PREHTML=, and POSTHTML= attributes. To alter the foreground color or the font for the text that appears in the table, you must set the corresponding attribute in a location that affects the cells rather than the table.

Note: You can use braces ({ and }) instead of square brackets ([and]). △

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.

Featured in: Example 14 on page 1279

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

Constructing Dimension Expressions

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 is composed of one or more elements and operators.

Elements That You Can Use in a Dimension Expression

analysis variables
(see "VAR Statement" on page 1211).

class variables
(see "CLASS Statement" on page 1197).

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).

Featured in: Example 6 on page 1246, Example 9 on page 1253, and Example 13 on page 1269

Note: If the input data set contains a variable named ALL, then enclose the name of the universal class variable in quotation marks. △

keywords for statistics
See "Statistics That Are Available in PROC TABULATE" on page 1213 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.

Restriction: Statistic keywords other than N must be associated with an analysis variable.

Default: For analysis variables, the default statistic is SUM. Otherwise, the default statistic is N.

Examples:

```
n
Region*n
Sales*max
```

Featured in: Example 10 on page 1255 and Example 13 on page 1269

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*

Example:

```
Sales*f=dollar8.2
```

Tip: Format modifiers have no effect on CLASS variables.

See also: For more information on specifying formats in tables, see "Formatting Values in Tables" on page 1215.

Featured in: Example 6 on page 1246

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.

Examples:

```
Region='Geographical Region'
Sales*max='Largest Sale'
```

Featured in: Example 5 on page 1244 and Example 7 on page 1249

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 1210.

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 headers 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*.

Examples:

```
Region*Division
Quarter*Sales*f=dollar8.2
```

Featured in: Example 1 on page 1232

(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
```

Featured in: Example 6 on page 1246

parentheses ()
: group elements and associate an operator with each concatenated element in the group.

Examples:

```
Division*(Sales*max Sales*min)
(Region ALL)*Sales
```

Featured in: Example 6 on page 1246

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 1216.

Featured in: Example 10 on page 1255 and Example 13 on page 1269

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=[foreground=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 1194 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)=
            [foreground=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 on page 1279.

VAR Statement

Identifies numeric variables to use as analysis variables.

Alias: VARIABLES

Tip: You can use multiple VAR statements.

VAR *analysis-variable(s)* </ *option(s)*>;

Required Arguments

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.

Options

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 information on the arguments of this option and how it is used, see STYLE= on page 1194 in the PROC TABULATE statement.

Note: When you use STYLE= in the VAR statement, it differs slightly from its use in the PROC TABULATE statement. In the VAR statement, the parent of the heading is the heading under which the current heading is nested. △

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.

Featured in: Example 14 on page 1279

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…
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.

Tip: When you use the WEIGHT= option, consider which value of the VARDEF= option is appropriate (see the discussion of VARDEF= on page 1195).

Tip: Use the WEIGHT option in multiple VAR statements to specify different weights for the analysis variables.

Note: Prior to Version 7 of SAS, the procedure did not exclude the observations with missing weights from the count of observations. △

WEIGHT Statement

Specifies weights for analysis variables in the statistical calculations.

See also: For information on calculating weighted statistics and for an example that uses the WEIGHT statement, see “Calculating Weighted Statistics” on page 64

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. 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.

Restriction: To compute weighted quantiles, use QMETHOD=OS in the PROC statement.

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= on page 1195 and the calculation of weighted statistics in "Keywords and Formulas" on page 1340 for more information.

Note: Prior to Version 7 of SAS, the procedure did not exclude the observations with missing weights from the count of observations. △

Concepts: TABULATE Procedure

Statistics That Are Available in PROC TABULATE

Use the following keywords to request statistics in the TABLE statement or to specify statistic keywords in the KEYWORD or KEYLABEL statement. 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

N	SUM
NMISS	SUMWGT
PAGEPCTN	UCLM
PAGEPCTSUM	USS
PCTN	VAR
Quantile statistic keywords	
MEDIAN | P50	Q3 | P75
P1	P90
P5	P95
P10	P99
Q1 | P25	QRANGE
Hypothesis testing keywords	
PROBT	T

These statistics, the formulas that are used to calculate them, and their data requirements are discussed in "Keywords and Formulas" on page 1340.

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 on creating user-defined formats, see Chapter 22, “The FORMAT Procedure,” on page 429.

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

- 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. 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.

Table 47.4 on page 1216 contrasts the two methods.

Table 47.4 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 on page 1246.)
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 1216.)
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 20).	The BOX= option in the TABLE statement customizes the page headers, 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 may 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 of 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 may 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 on page 1266 for an example.

Using PCTN and PCTSUM

PCTN and PCTSUM statistics can be used to calculate these same percentages. They allow 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 on page 1269 illustrates how to specify denominator definitions in a table that is composed of multiple subtables. For more examples of denominator definitions, see "How Percentages Are Calculated" in Chapter 3, "Details of TABULATE Processing," in *SAS Guide to TABULATE Processing*.

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"ENERGY" on page 1387.

```
proc tabulate data=energy;
   class division type;
   table division*
           (n='Number of customers'
            pctn<type>='% of row' ❶
            pctn<division>='% of column'  ❷
            pctn='% of all customers'), ❸
          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 Figure 47.4 on page 1218). Each occurrence of PCTN uses a different denominator definition.

Figure 47.4 Three Different Uses of the PCTN Statistic with Frequency Counts Highlighted

```
              Number of Users in Each Division
1

------------------------------------------------------------
|                                |         Type            |
|                                |-------------------------|
|                                |     1      |     2      |
|--------------------------------+------------+------------|
|Division |                      |            |            |
|---------+----------------------|            |            |
|1        |Number of customers   |        6.00|        6.00|
|         |----------------------+------------+------------|
|         |% of row ❶            |       50.00|       50.00|
|         |----------------------+------------+------------|
|         |% of column ❷         |       27.27|       27.27|
|         |----------------------+------------+------------|
|         |% of all customers ❸  |       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|
```

❶ **<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.

❷ **<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.

❸ 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' ❶
```

```
            pctsum<division>='% of column' ❷
            pctsum='% of all customers'), ❸
         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 Figure 47.5 on page 1219). Each occurrence of PCTSUM uses a different denominator definition.

Figure 47.5 Three Different Uses of the PCTSUM Statistic with Sums Highlighted

```
               Expenditures in Each Division           1

 --------------------------------------------------------
 |                                 |        Type         |
 |                                 |---------------------|
 |                                 |    1     |    2     |
 |                                 |----------+----------|
 |                                 |  Expend  |  Expend  |
 |---------------------------------+----------+----------|
 |Division   |                     |          |          |
 |-----------+---------------------|          |          |
 |1          |Expenditures         | $7,477.00| $5,129.00|
 |           |---------------------+----------+----------|
 |           |% of row ❶           |     59.31|     40.69|
 |           |---------------------+----------+----------|
 |           |% of column ❷        |     16.15|     13.66|
 |           |---------------------+----------+----------|
 |           |% of all customers ❸ |      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|
 --------------------------------------------------------
```

❶ **`<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.

❷ **`<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.

❸ 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 forth. See "Output Delivery System" on page 32 for more information. The following table lists the default styles for various regions of a table.

Table 47.5 Default Styles for Table Regions

Region	Style
column headings	Header
box	Header
page dimension text	Beforecaption
row headings	Rowheader
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 instance, 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 on STYLE= is provided in the documentation for individual statements.

Table 47.6 Using the STYLE= Option in PROC TABULATE

To set the style element for	Use STYLE in this statement
data cells	PROC TABULATE or dimension expression(s)
page dimension text and class variable name headings	CLASS
class level value headings	CLASSLEV
keyword headings	KEYWORD

To set the style element for	Use STYLE in this statement
table borders, rules, and other parts that are not specified elsewhere	TABLE
box text	TABLE statement, BOX= option
missing values	TABLE statement, MISSTEXT= option
analysis variable name headings	VAR

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. 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 value(s) 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=[background=expfmt.];
   class region division type;
   var expenditures;
   table (region all)*(division all),
           type*expenditures;
run;
ods html close;
```

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. Table 47.7 on page 1222 summarizes how the procedure treats missing values.

Table 47.7 Summary of How PROC TABULATE Treats Missing Values

If …	PROC TABULATE, by default, …	To override the 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

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.

This section presents a series of PROC TABULATE steps that illustrate how PROC TABULATE treats missing values. The following program creates the data set and formats that are used in this section and prints the data set. The data set COMPREV contains no missing values (see Figure 47.6 on page 1223).

```
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 47.6 The Data Set COMPREV

```
                     The Data Set COMPREV                      1

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 Figure 47.7 on page 1224:

```
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;
```

Figure 47.7 Computer Sales Data: No Missing Values

Because the data set contains no missing values, the table includes all observations. All headers and cells contain nonmissing values.

Revenues from Computer Sales 1
for 1990 to 1992

Computer	Country	Rev90 Sum	Rev91 Sum	Rev92 Sum
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 47.8 on page 1225 is the same as the program that produces Figure 47.7 on page 1224. 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;
```

Figure 47.8 Computer Sales Data: Midrange, Japan, Deleted

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 fewer row than Figure 47.7 on page 1224.

```
                  Revenues from Computer Sales                 1
                       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      |United States | 18660.90|  18428.00| 23531.10|
 |Computer      |-------------+---------+----------+---------|
 |              |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 variable(s). The MISSING option includes observations with missing values of a class variable in the report (see Figure 47.9 on page 1226).

```
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;
```

Figure 47.9 Computer Sales Data: Missing Values for Computer

This table includes a category with missing values of Computer. This category makes up the first row of data in the table.

```
                                                          1

 -----------------------------------------------------------
|                        Animal                             |
|-----------------------------------------------------------|
|           cat             |             dog               |
|---------------------------+-------------------------------|
|           Food            |             Food              |
|---------------------------+-------------------------------|
|  fish   |  meat   |  milk   |  fish   |  meat   |  bones  |
|---------+---------+---------+---------+---------+---------|
|    N    |    N    |    N    |    N    |    N    |    N    |
|---------+---------+---------+---------+---------+---------|
|        1|        1|        1|        1|        1|        1|
 -----------------------------------------------------------
```

Formatting Headings for Observations with Missing Class Variables

By default, as shown in Figure 47.9 on page 1226, 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 Figure 47.10 on page 1227):

```
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;
```

Figure 47.10 Computer Sales Data: Text Supplied for Missing Computer Value

In this table, the missing value appears as the text that the MISSCOMP. format specifies.

Revenues for Computer Sales 1
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 47.10 on page 1227 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 Figure 47.11 on page 1228):

```
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;
```

Figure 47.11 Computer Sales Data: Missing Statistics Values

This table contains a row for the categories **`No type given`**, **`United States`** and **`Midrange, Japan`**. Because there are no data in these categories, the values for the statistics are all missing.

Revenues for Computer Sales 1
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 Figure 47.12 on page 1229).

```
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;
```

Figure 47.12 Computer Sales Data: Text Supplied for Missing Statistics Values

This table replaces the period normally used to display missing values with the text of the MISSTEXT= option.

Revenues for Computer Sales 1
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 on page 1237 for another example that uses PRELOADFMT.
- Use the CLASSDATA= option in the PROC TABULATE statement. See Example 2 on page 1235 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 Figure 47.13 on page 1230.

```
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;
```

Figure 47.13 Computer Sales Data: All Possible Computer Values Included

This table contains a heading for each possible value of Computer.

```
                 Revenues for Computer Sales            1
                       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      |United States|18660.90|18428.00|23531.10|
 |Computer      |-------------+--------+--------+--------|
 |              |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 doing this 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 Figure 47.14 on page 1231). 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 header for **`bones`** in the table in Figure 47.14 on page 1231 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 47.14 Ordering the Headings of Class Variables

```
                                                             1

-----------------------------------------------------------
|                          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

Procedure features:
- PROC TABULATE statement options:
  - FORMAT=
- TABLE statement
  - crossing (*) operator
- TABLE statement options:
  - RTS=

Other features: FORMAT statement

This example

- 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

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 1387 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;
```

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 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

```
                      Energy Expenditures for Each Region                      1
                             (millions of dollars)

               ------------------------------------------------
               |                       |         Type         |
               |                       |-------------------------|
               |                       |Residential | Business  |
               |                       | Customers  | 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

Procedure features:

PROC TABULATE Statement options:

CLASSDATA=
EXCLUSIVE

Data set: ENERGY "ENERGY" on page 1387

Formats: REGFMT., DIVFMT., and USETYPE. on page 1233

This example

- 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 on page 1232.

Program

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
;
```

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 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 1
(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

Procedure features:

PROC TABULATE statement option:

OUT=

CLASS statement options:

EXCLUSIVE
PRELOADFMT

TABLE statement option:

PRINTMISS

Other features: PRINT procedure

Data set: ENERGY “ENERGY” on page 1387

Formats: REGFMT., DIVFMT., and USETYPE. on page 1233

This example

- 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

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 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 1
(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 on page 1232.

```
                     Energy Expenditures for Each Region                  1
                             (millions of dollars)

              ---------------------------------------------------
               |                        |          Type           |
               |                        |-------------------------|
               |                        |Residential |  Business  |
               |                        | Customers  | 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 is a listing of 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)

                                                                        E
                                                                        x
                                                                        p
                                                                        e
                                                                        n
                                                                        d
                                                                        i
                                                                        t
                 D                                                      u
                 i                                                 _    r
     R           v                                       _    _    T    e
     e           i                                       T    P    A    s
     g           s                         T             Y    A    B    _
 O   i           i                         y             P    G    L    S
 b   o           o                         p             E    E    E    u
 s   n           n                         e             _    _    _    m

 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

Procedure 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

This example

- 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

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 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 option on page 449. A multilabel format is one in which multiple labels can be assigned to the same value, in this case because of overlapping ranges. Each value is represented in the table for each range in which it occurs. The NOTSORTED option stores the ranges in the order in which they are defined.

```
proc format;
   value agefmt (multilabel notsorted)
         15 - 29 = 'Below 30 years'
         30 - 50 = 'Between 30 and 50'
       51 - high = 'Over 50 years'
         15 - 19 = '15 to 19'
         20 - 25 = '20 to 25'
         25 - 39 = '25 to 39'
         40 - 55 = '40 to 55'
       56 - high = '56 and above';
run;
```

Specify the table options. The FORMAT= option specifies up to 10 digits as the default format for the value in each table cell.

```
proc tabulate data=carsurvey format=10.;
```

Specify subgroups for the analysis. The CLASS statement identifies Age as the class variable and uses the MLF option to activate multilabel format processing.

```
   class age / mlf;
```

Specify the analysis variables. The VAR statement specifies that PROC TABULATE calculate statistics on the Progressa, Remark, Jupiter, and Dynamo variables.

```
   var progressa remark jupiter dynamo;
```

Define the table rows and columns. The row dimension of the TABLE statement creates a row for each formatted value of Age. Multilabel formatting allows an observation to be included in multiple rows or age categories. The row dimension uses the ALL class variable to summarize information for all rows. The column dimension uses the N statistic to calculate the number of observations for each age group. Notice that the result of the N statistic crossed with the ALL class variable in the row dimension is the total number of observations instead of the sum of the N statistics for the rows. The column dimension uses the ALL class variable at the beginning of a crossing to assign a label, **`Potential Car Names`**. The four nested columns calculate the mean ratings of the car names for each age group.

```
   table age all, n all='Potential Car Names'*(progressa remark
      jupiter dynamo)*mean;
```

Specify the titles.

```
   title1 "Rating Four Potential Car Names";
   title2 "Rating Scale 0-100 (100 is the highest rating)";
```

Format the output. The FORMAT statement assigns the user-defined format AGEFMT. to Age for this analysis.

```
   format age agefmt.;
run;
```

Output

Output 47.3

Rating Four Potential Car Names 1
Rating Scale 0-100 (100 is the highest rating)

		Potential Car Names			
		Progressa	Remark	Jupiter	Dynamo
	N	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

Procedure features:
TABLE statement
labels

Data set: ENERGY"ENERGY" on page 1387

Formats: REGFMT., DIVFMT., and USETYPE. on page 1233

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

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 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                 1
                           (millions of dollars)

          ---------------------------------------------------
          |                       |      Customer Base      |
          |                       |-------------------------|
          |                       |Residential |  Business  |
          |                       | Customers  | 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

Procedure features:

PROC TABULATE statement options:

- FORMAT=

TABLE statement:

- ALL class variable
- concatenation (blank operator)
- format modifiers
- grouping elements (parentheses operator)

Data set: ENERGY“ENERGY” on page 1387

Formats: REGFMT., DIVFMT., and USETYPE. on page 1233

This example shows how to use the universal class variable ALL to summarize information from multiple categories.

Program

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=60;
```

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;
```

Define the table rows. The row dimension of the TABLE statement creates a row for each formatted value of Region. Nested within each row are rows for each formatted value of Division and a row (labeled **`Subtotal`**) that summarizes all divisions in the region. The last row of the report (labeled **`Total for All Regions`**) summarizes all regions. The format modifier f=DOLLAR12. assigns the DOLLAR12. format to the cells in this row.

```
table region*(division all='Subtotal')
         all='Total for All Regions'*f=dollar12.,
```

Define the table columns. The column dimension of the TABLE statement creates a column for each formatted value of Type and a column that is labeled **`All customers`** that shows expenditures for all customers in a row of the table. 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.

```
         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 1
(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

Example 7: Eliminating Row Headings

Procedure features:

TABLE statement:

labels
ROW=FLOAT

Data set: ENERGY“ENERGY” on page 1387

Formats: REGFMT., DIVFMT., and USETYPE. on page 1233

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

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 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 on page 1244. 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                  1
                            (millions of dollars)

               ----------------------------------------------------
               |                       |      Customer Base       |
               |                       |--------------------------|
               |                       |Residential |  Business   |
               |                       | Customers  | 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

Procedure features:

PROC TABULATE statement options:

NOSEPS

TABLE statement options:

INDENT=

Data set: ENERGY“ENERGY” on page 1387

Formats: REGFMT., DIVFMT., and USETYPE. on page 1233

This example shows how to condense the structure of a table by

- 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

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 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;
```

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                     1
                             (millions of dollars)

              -----------------------------------------------------
              |                       |      Customer Base        |
              |                       |---------------------------|
              |                       |Residential  |  Business   |
              |                       | Customers   | 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

Procedure features:
TABLE statement
ALL class variable
BOX=
CONDENSE
INDENT=
page expression

Data set: ENERGY "ENERGY" on page 1387

Formats: REGFMT., DIVFMT., and USETYPE. on page 1233

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

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 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 1
(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

Procedure 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

The two tables in this example show

- 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

- 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

Figure 47.15 on page 1256 shows the survey form that is used to collect data.

Figure 47.15 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 sales person effective?

____ Product knowledge ____ Personality ____ Appearance

Program

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 FORMDLIM= option replaces the character that delimits page breaks with a single blank. By default, a new physical page starts whenever a page break occurs.

```
options nodate pageno=1 linesize=80 pagesize=18 formdlim=' ';
```

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 1380 creates the data set. Using missing values rather than 0's 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 headers 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 headers.

```
   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;
```

Suppress page numbers. The SAS system option NONUMBER suppresses page numbers for subsequent pages.

```
options nonumber;
```

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                1

              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

Procedure features:

TABLE statement:

N statistic

Other features:

FORMAT procedure

TRANSPOSE procedure

Data set options:

RENAME=

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

Figure 47.16 on page 1261 shows the survey form that is used to collect data.

Figure 47.16 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 question 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 1405 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

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=132 pagesize=40;
```

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;
```

Read the appropriate data line, assign a unique number to each respondent, and write an observation to RADIO. Each raw-data record contains two lines of information about each listener. The INPUT statement reads only the information that this example needs. The / line control skips the first line of information in each record. The rest of the INPUT statement reads Time1-Time7 from the beginning of the second line. These variables represent the listener's radio programming preference for each of seven time periods on weekdays (see Figure 47.16 on page 1261). The **listener=_N_** statement assigns a unique identifier to each listener. An observation is automatically written to RADIO at the end of each iteration.

```
   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 "A Closer Look" on page 1264 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;
```

Format the transposed variables. The FORMAT statement permanently associates these formats with the variables in the output data set.

```
   format timespan $timefmt. choice $pgmfmt.;
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.;
```

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 1

	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

A Closer Look

Reshape the data

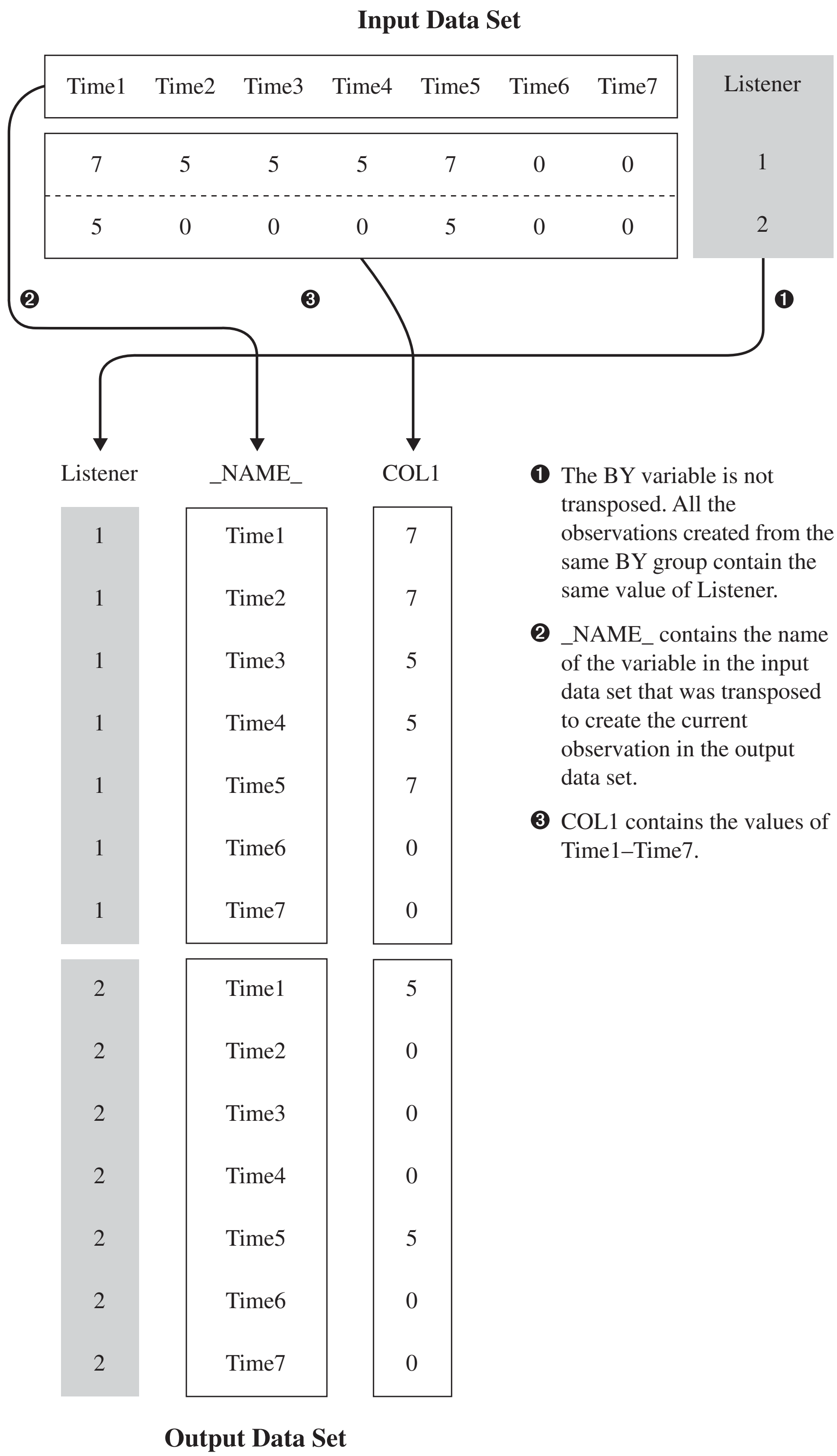
The original input data set has all the information that you need to make the crosstabular report, but PROC TABULATE cannot use the information in that form. PROC TRANSPOSE rearranges the data so that each observation in the new data set contains the variable Listener, a variable for time of day, and a variable for programming preference. Figure 47.17 on page 1265 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.

Figure 47.17 Transposing Two Observations



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 ❶
               out=radio_transposed(rename=(col1=Choice)) ❷
               name=Timespan; ❸
   by listener; ❹
   var time1-time7; ❺
   format timespan $timefmt. choice $pgmfmt.; ❻
run;
```

❶ The DATA= option specifies the input data set.

❷ The OUT= option specifies the output data set. The RENAME= data set option renames the transposed variable from COL1 (the default name) to Choice.

❸ 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_.

❹ The BY statement identifies Listener as the BY variable.

❺ The VAR statement identifies Time1 through Time7 as the variables to transpose.

❻ 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

Procedure 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

This example shows how to use three percentage sum statistics: COLPCTSUM, REPPCTSUM, and ROWPCTSUM.

Program

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=105 pagesize=60;
```

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                                   1

-------------------------------------------------------------------------------------------
|                  |                             Classroom                         |       |
|                  |---------------------------------------------------------------|       |
|                  |                A                |               B             |  All  |
|                  |---------------------------------+-----------------------------+-------|
|                  |  Sum  |ColPctSum|RowPctSum|RepPctSum|  Sum  |ColPctSum|RowPctSum|RepPctSum|  Sum  |
|------------------+-------+---------+---------+---------+-------+---------+---------+---------+-------|
|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|
-------------------------------------------------------------------------------------------
```

A Closer Look

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

Procedure features:

TABLE statement:

ALL class variable
denominator definitions (angle bracket operators)
N statistic
PCTN statistic

Other features:

FORMAT procedure

Crosstabulation tables (also called *contingency tables* and *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 of

- 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

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 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;
```

Define the table rows. The TABLE statement creates a set of rows for each formatted value of Occupation and for all jobs together. Text in quotation marks supplies a header for the corresponding row.

The asterisk in the row dimension indicates that the statistics that follow in parentheses are nested within the values of Occupation and All to form sets of rows. Each set of rows includes four statistics:

- N, the frequency count. The format modifier (F=9.) writes the values of N without the decimal places that the default format would use. It also extends the column width to nine characters so that the word **`Employees`** fits on one line.
- the percentage of the row total (row percent).
- the percentage of the column total (column percent).
- the overall percent. Text in quotation marks supplies the heading for the corresponding row. A comma separates the row definition from the column definition.

For detailed explanations of the structure of this table and of the use of denominator definitions, see “A Closer Look” on page 1272.

```
     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 1

Job Class		Gender: Female	Gender: Male	All Employees
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

A Closer Look

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';
```

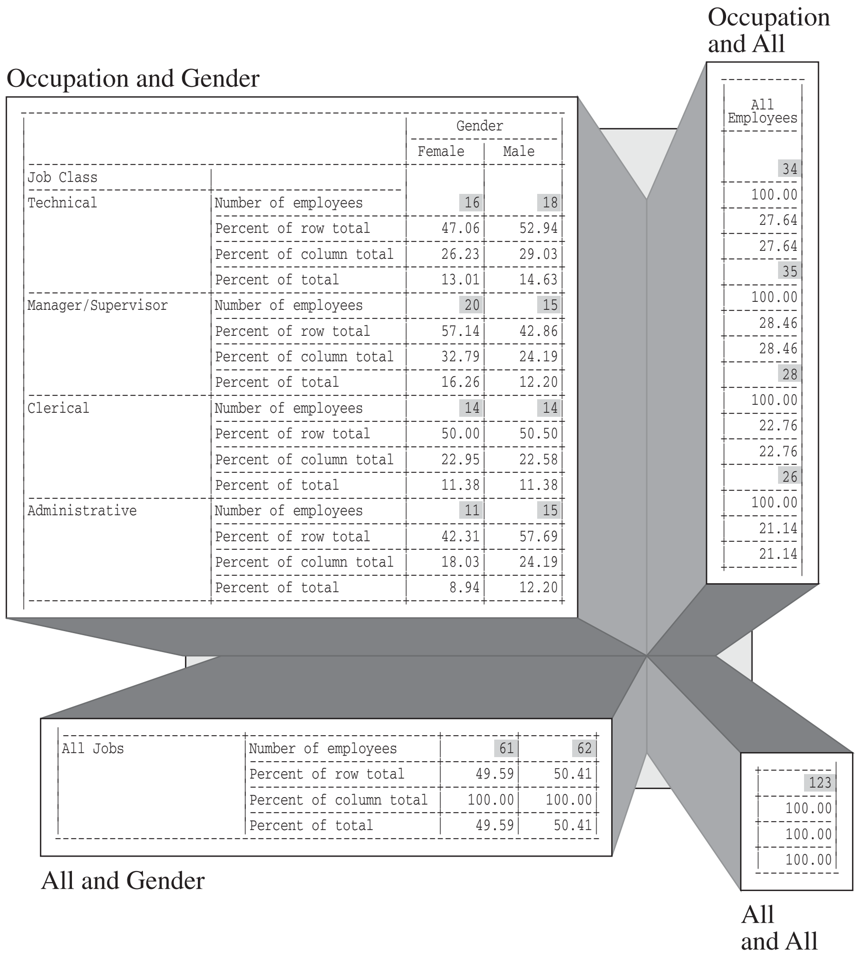
The table is a concatenation of four subtables. In this report, each subtable is a crossing of one class variable in the row dimension and one class variable in the column dimension. Each crossing establishes one or more categories. A *category* is a combination of unique values of class variables, such as **`female, technical`** or **`all, clerical`**. Table 47.8 on page 1273 describes each subtable.

Table 47.8 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

Figure 47.18 on page 1274 highlights these subtables and the frequency counts for each category.

Figure 47.18 Illustration of the Four Subtables



Interpreting Denominator Definitions

The following fragment of the TABLE statement defines the denominator definitions for this report. The PCTN keyword and the denominator definitions are highlighted.

```
table (occupation='Job Class' all='All Jobs')
          *(n='Number of employees'*f=5.
            pctn<gender all>='Row percent'
            pctn<occupation all>='Column percent'
            pctn='Percent of total'),
```

Each use of PCTN nests a row of statistics within each value of Occupation and All. Each denominator definition tells PROC TABULATE which frequency counts to sum for the denominators in that row. This section explains how PROC TABULATE interprets these denominator definitions.

Row Percentages

The part of the TABLE statement that calculates the row percentages and that labels the row is

```
pctn<gender all>='Row percent'
```

Consider how PROC TABULATE interprets this denominator definition for each subtable.

Subtable 1: Occupation and Gender

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	50.50	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 if 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.

Subtable 2: All and Gender

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	50.50	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 if 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.

Subtable 3: Occupation 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	50.50	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 if 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. △

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	50.50	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 if 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.

Subtable 1: Occupation and Gender

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	50.50	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 if 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.

Subtable 2: All and Gender

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	50.50	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 if 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. △

Subtable 3: Occupation 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	50.50	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 if 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.

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	50.50	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 if 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. Table 47.9 on page 1279 summarizes the process for all subtables in this example.

Table 47.9 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
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

Procedure features:

STYLE= option in

- 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“ENERGY” on page 1387

Formats: REGFMT, DIVFMT, and USETYPE. on page 1233

This example creates HTML, RTF, and PDF files and specifies style elements for various table regions.

Program

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=[font_weight=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=[just=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=[just=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=[font_size=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=[font_width=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=[background=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=[font_weight=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=[font_style=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;
```

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

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

RTF Output

Energy Expenditures
(millions of dollars)

Region by Division by Type		Type: Residential Customers	Type: 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

References

Jain, Raj and Chlamtac, Imrich (1985), "The P^2 Algorithm for Dynamic Calculation of Quantiles and Histograms without Storing Observations," *Communications of the Association of Computing Machinery*, 28:10.

CHAPTER 48

The TEMPLATE Procedure

Information about the TEMPLATE Procedure

See: For complete documentation of the TEMPLATE procedure, see *SAS Output Delivery System: User's Guide*.

CHAPTER

49

The 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 those 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 may 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 may include variables that do not appear in the plot.

Output 49.1 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 stepExample 1 on page 1299 creates the data set SALES.

```
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 49.1 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                         |
                               *-------------------------------*
```

Output 49.2 is a more complicated report of the same data set that is used to create Output 49.1. The statements that create this report

- 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 on page 1308.

Output 49.2 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

Requirements: At least one PLOT statement

Tip: Supports the Output Delivery System. See “Output Delivery System” on page 32 for details.

ODS Table Names: See: “ODS Table Names” on page 1298

Reminder: You can use the ATTRIB, FORMAT, LABEL, and WHERE statements. See Chapter 3, “Statements with the Same Function in Multiple Procedures,” on page 57 for details. You can also use any global statements. See “Global Statements” on page 18 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)*;

To do this	Use this statement
Produce a separate plot for each BY group	BY
Group data according to the values of the class variables	CLASS
Print in the listing the values of the variables that you identify	ID
Specify the plots to produce	PLOT

PROC TIMEPLOT Statement

PROC TIMEPLOT <*option(s)*>;

Options

DATA=*SAS-data-set*
identifies the input data set.

MAXDEC=*number*
specifies the maximum number of decimal places to print in the listing.

Interaction: A decimal specification in a format overrides a MAXDEC= specification.

Default: 2

Range: 0-12

Featured in: Example 4 on page 1306

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= on page 1296).

BY Statement

Produces a separate plot for each BY group.

Main discussion: "BY" on page 58

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. These variables are called *BY variables*.

Options

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.

Featured in: Example 5 on page 1308

CLASS *variable(s)*;

Required Arguments

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.

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 1294). In this case, the listing of the plot variable contains a column for each value of the symbol variable, and 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.

Featured in: Example 1 on page 1299

ID *variable(s)*;

Required Arguments

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.

PLOT *plot-request(s)/option(s)*;

Table 49.1 on page 1293 summarizes the options that are available in the PLOT statement.

Table 49.1 Summary of Options for the PLOT Statement

To do this	Use this option
Customize the axis	
Specify the range of values to plot on the horizontal axis, as well as the interval represented by each print position on the horizontal axis	AXIS=
Order the values on the horizontal axis with the largest value in the leftmost position	REVERSE
Control the appearance of the plot	
Connect the leftmost plotting symbol to the rightmost plotting symbol with a line of hyphens (-)	HILOC
Connect 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	JOINREF
Suppress the name of the symbol variable in column headings when you use a CLASS statement	NOSYMNAME
Suppress the listing of the values of the variables that appear in the PLOT statement	NPP
Specify the number of print positions to use for the horizontal axis	POS=
Create and customize a reference line	
Draw lines on the plot that are perpendicular to the specified values on the horizontal axis	REF=
Specify the character for drawing reference lines	REFCHAR=
Display multiple plots on the same set of axes	
Plot all requests in one PLOT statement on one set of axes	OVERLAY
Specify the character to print if multiple plotting symbols coincide	OVPCHAR=

Required Arguments

plot-request(s)
specifies the variable or variables to plot and, optionally, 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 on page 1306).

variable(s)
identifies one or more numeric variables to plot. PROC TIMEPLOT uses the first character of the variable name as the plotting symbol.

Featured in: Example 1 on page 1299

(*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.

Featured in: Example 2 on page 1301

(*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.

Featured in: Example 3 on page 1303

Options

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= on page 1296) overrides an interval set with AXIS=.

Tip: If the range that you specify does not include all your data, then PROC TIMEPLOT uses angle brackets (< or >) on the left or right border of the plot to indicate a value that is outside the range.

Featured in: Example 2 on page 1301

HILOC
connects the leftmost plotting symbol to the rightmost plotting symbol with a line of hyphens (-).

Interactions: 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.

Featured in: Example 3 on page 1303

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.

Featured in: Example 5 on page 1308

NPP
suppresses the listing of the values of the variables that appear in the PLOT statement.

Featured in: Example 3 on page 1303

OVERLAY
plots all requests in one PLOT statement on one set of axes. Otherwise, PROC TIMEPLOT produces a separate plot for each plot request.

Featured in: Example 4 on page 1306

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 (@)

Featured in: Example 5 on page 1308

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= on page 1294).

See also: “Page Layout” on page 1297

Featured in: Example 1 on page 1299

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)* may be constants, or you may 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.

Interaction: 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.

Interaction: If a plotting symbol and a reference character coincide, then PROC TIMEPLOT prints the plotting symbol.

Featured in: Example 3 on page 1303 and Example 4 on page 1306

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, because PROC TIMEPLOT will interpret the plotting symbols as reference characters and will not connect the symbols as you expect.

Featured in: Example 3 on page 1303

REVERSE
orders the values on the horizontal axis with the largest value in the leftmost position.

Featured in: Example 4 on page 1306

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 1291.)

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
  - 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
  - □ 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 for a particular plot request, 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 or not you use a CLASS statement. If you do not use a CLASS statement (see Example 1 on page 1299), 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 or not you specify a symbol variable (see "Using a Symbol Variable" on page 1292).

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 *SAS Output Delivery System: User's Guide*.

Table 49.2 ODS Tables Produced by the TIMEPLOT Procedure

Table Name	Description	The TIMEPLOT procedure generates the table:
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 headers). 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 1294), 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

Procedure features:

ID statement

PLOT statement arguments:

simple plot request

POS=

This example

- uses a single PLOT statement to plot sales of refrigerators
- specifies the number of print positions to use for the horizontal axis of the plot
- provides context for the points in the plot by printing in the listing the values of two variables that are not in the plot.

Program

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 SALES data set. SALES contains weekly information on 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 headers 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.

```
                           Weekly Sales of Iceboxes                          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                                          |
                            *--------------------------------------------------*
```

Example 2: Customizing an Axis and a Plotting Symbol

Procedure features:
ID statement
PLOT statement arguments:
using a plotting symbol
AXIS=

Other features:
LABEL statement
PROC FORMAT
SAS system options:
FMTSEARCH=

Data set: SALES on page 1299

This example

- specifies the character to use as the plotting symbol
- specifies the minimum and maximum values for the horizontal axis as well as the interval represented by each print position
- provides context for the points in the plot by printing in the listing the values of two variables that are not in the plot
- uses a variable's label as a column header in the listing
- creates and uses a permanent format.

Program

Declare the PROCLIB SAS data library.

```
libname proclib 'SAS-data-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. FMTSEARCH= adds the SAS data library PROCLIB to the search path that is used to locate formats.

```
options nodate pageno=1 linesize=80 pagesize=60
        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 header 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 headers 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).

```
                         Weekly Sales of Refrigerators                         1
                                    for the
                          First Six Weeks of the Year

   Month    Week   Refrigerator    min                                      max
                                   2500                                    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

Procedure features:

ID statement

PLOT statement arguments:

using a variable as the plotting symbol
JOINREF
NPP
REF=
REFCHAR=

Data set: SALES on page 1299

Formats: MONTHFMT. on page 1302

This example

- specifies a variable to use as the plotting symbol to distinguish between points for each of two sales representatives
- suppresses the printing of the values of the plot variable in the listing
- draws a reference line to a specified value on the axis and specifies the character to use to draw the line
- connects the leftmost and rightmost symbols on each line of the plot.

Program

Declare the PROCLIB SAS data library.

```
libname proclib 'SAS-data-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. FMTSEARCH= adds the SAS data library PROCLIB to the search path that is used to locate formats.

```
options nodate pageno=1 linesize=80 pagesize=60
        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.

```
                             Weekly Sales of Stoves                            1
                       Compared to Target Sales of $1500
                          K for Kreitz; L for LeGrange

   Month   Week     min                                                      max
                    100                                                     3000
                   *-----------------------------------------------------------*
January       1    |                        K---:                              |
January       1    |             L--------------:                              |
January       2    |  K-------------------------:                              |
January       2    |  L-------------------------:                              |
January       3    |                            :--------------K               |
January       3    |   L------------------------:                              |
January       4    |                            :-----K                        |
January       4    |   L------------------------:                              |
February      1    |                            :---------------------------K  |
February      1    |      L---------------------:                              |
February      2    |              K-------------:                              |
February      2    |                            :--------------L               |
                   *-----------------------------------------------------------*
```

Example 4: Superimposing Two Plots

Procedure features:
PROC TIMEPLOT statement options:
MAXDEC=
PLOT statement arguments:
using two types of plot requests
OVERLAY
REF=MEAN(*variable(s)*)
REVERSE

Data set: SALES on page 1299

This example

- □ superimposes two plots on one set of axes
- □ specifies a variable to use as the plotting symbol for one plot and a character to use as the plotting symbol for the other plot
- □ draws a reference line to the mean value of each of the two variables plotted
- □ reverses the labeling of the axis so that the largest value is at the far left of the plot.

Program

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 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 header 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 header 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**.

```
                    Weekly Sales of Stoves and Refrigerators                  1
                                   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

Procedure features:

CLASS statement

PLOT statement arguments:

creating multiple plots
NOSYMNAME
OVPCHAR=

Data set: SALES on page 1299

Formats: MONTHFMT. on page 1302

This example

- groups 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
- specifies a variable to use as the plotting symbol
- suppresses the name of the plotting variable from one plot
- specifies a size for the plots so that they both occupy the same amount of space.

Program

Declare the PROCLIB SAS data library.

```
libname proclib 'SAS-data-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. FMTSEARCH= adds the SAS data library PROCLIB to the search path that is used to locate formats.

```
options nodate pageno=1 linesize=80 pagesize=60
        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

```
                 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    |
                                                     *-------------------------*
```

CHAPTER

50

The 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.

A *transposed variable* is a variable that the procedure creates by transposing 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 50.1 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 50.2 A Transposition with BY Groups

```
                                 Input Data Set                                1

     L
     o                     L      W      L      W      L      W      L      W
     c                     e      e      e      e      e      e      e      e
     a                     n      i      n      i      n      i      n      i
     t              D      g      g      g      g      g      g      g      g
     i              a      t      h      t      h      t      h      t      h
     o              t      h      t      h      t      h      t      h      t
     n              e      1      1      2      2      3      3      4      4

 Cole Pond     02JUN95    31    0.25    32    0.30    32    0.25    33    0.30
 Cole Pond     03JUL95    33    0.32    34    0.41    37    0.48    32    0.28
 Cole Pond     04AUG95    29    0.23    30    0.25    34    0.47    32    0.30
 Eagle Lake    02JUN95    32    0.35    32    0.25    33    0.30     .      .
 Eagle Lake    03JUL95    30    0.20    36    0.45     .      .       .      .
 Eagle Lake    04AUG95    33    0.30    33    0.28    34    0.42     .      .
```

```
                Fish Length Data for Each Location and Date                    2

                 Location        Date    _NAME_     Measurement

                Cole Pond     02JUN95    Length1         31
                Cole Pond     02JUN95    Length2         32
                Cole Pond     02JUN95    Length3         32
                Cole Pond     02JUN95    Length4         33
                Cole Pond     03JUL95    Length1         33
                Cole Pond     03JUL95    Length2         34
                Cole Pond     03JUL95    Length3         37
                Cole Pond     03JUL95    Length4         32
                Cole Pond     04AUG95    Length1         29
                Cole Pond     04AUG95    Length2         30
                Cole Pond     04AUG95    Length3         34
                Cole Pond     04AUG95    Length4         32
                Eagle Lake    02JUN95    Length1         32
                Eagle Lake    02JUN95    Length2         32
                Eagle Lake    02JUN95    Length3         33
                Eagle Lake    02JUN95    Length4          .
                Eagle Lake    03JUL95    Length1         30
                Eagle Lake    03JUL95    Length2         36
                Eagle Lake    03JUL95    Length3          .
                Eagle Lake    03JUL95    Length4          .
                Eagle Lake    04AUG95    Length1         33
                Eagle Lake    04AUG95    Length2         33
                Eagle Lake    04AUG95    Length3         34
                Eagle Lake    04AUG95    Length4          .
```

For a complete explanation of the SAS program that produces these results, see Example 4 on page 1325.

Syntax: TRANSPOSE Procedure

Tip: Does not support the Output Delivery System

Reminder: You can use the ATTRIB, FORMAT, LABEL, and WHERE statements. See Chapter 3, "Statements with the Same Function in Multiple Procedures," on page 57 for details. You can also use any global statements. See "Global Statements" on page 18 for a list.

PROC TRANSPOSE <DATA=*input-data-set*> <LABEL=*label*> <LET>
<NAME=*name*> <OUT=*output-data-set*> <PREFIX=*prefix*>;
BY <DESCENDING> *variable-1*
<...<DESCENDING> *variable-n*>
<NOTSORTED>;
COPY *variable(s)*;
ID *variable*;
IDLABEL *variable*;
VAR *variable(s)*;

To do this	Use this statement
Transpose each BY group	BY
Copy variables directly without transposing them	COPY
Specify a variable whose values name the transposed variables	ID
Create labels for the transposed variables	IDLABEL
List the variables to transpose	VAR

PROC TRANSPOSE Statement

Reminder: You can use data set options with the DATA= and OUT= options. See "Data Set Options" on page 18 for a list.

PROC TRANSPOSE <DATA=*input-data-set*> <LABEL=*label*> <LET>
<NAME=*name*> <OUT=*output-data-set*> <PREFIX=*prefix*>;

Options

DATA= ***input-data-set***
names the SAS data set to transpose.

Default: most recently created SAS data set

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_

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.

Featured in: Example 5 on page 1328

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_

Featured in: Example 2 on page 1323

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*

Featured in: Example 1 on page 1321

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 value prefixes to the ID value.

Featured in: Example 2 on page 1323

BY Statement

Defines BY groups.

Main discussion: "BY" on page 58

Featured in: Example 4 on page 1325

Restriction: You cannot use PROC TRANSPOSE with a BY statement or an ID statement with an engine that supports concurrent access if another user is updating the data set at the same time.

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

Required Arguments

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*.

Options

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.

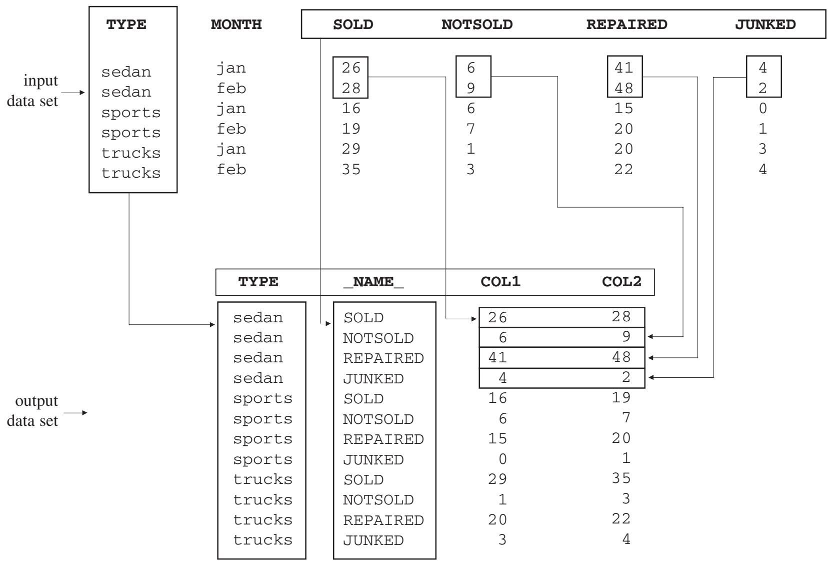
The requirement for ordering or indexing observations according to the values of BY variables is suspended for BY-group processing when you use the NOTSORTED option. In fact, the procedure does not use an index if you specify NOTSORTED. The procedure defines a BY group as a set of contiguous observations that have the same values for all BY variables. If observations with the same values for the BY variables are not contiguous, then the procedure treats each contiguous set as a separate BY group.

Transpositions with BY Groups

PROC TRANSPOSE does not transpose BY groups. Instead, for each BY group, PROC TRANSPOSE creates one observation for each variable that it transposes.

Figure 50.1 on page 1317 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 50.1 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.

Featured in: Example 6 on page 1329

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 a variable in the input data set whose formatted values name the transposed variables in the output data set.

Featured in: Example 2 on page 1323

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.

ID *variable*;

Required Argument

variable
names the variable whose formatted values name the transposed variables.

Duplicate ID Values

Typically, each formatted ID value occurs only once in the input data set or, if you use a BY statement, only once within a BY group. Duplicate values cause PROC TRANSPOSE to issue a warning message and stop. However, if you use the LET option in the PROC TRANSPOSE statement, then the procedure issues a warning message about duplicate ID values and transposes the observation that contains the last occurrence of the duplicate ID value.

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.

However, SAS variable names cannot begin with a number. Thus, when the first character of the formatted value is numeric, the procedure prefixes an underscore to the value, truncating the last character of a 32-character value. Any remaining invalid characters are replaced by underscores. The procedure truncates to 32 characters any ID value that is longer than 32 characters when it 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 numerics, 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 eight characters. △

Missing Values

If you use an ID variable that contains a missing value, then PROC TRANSPOSE writes an error message to the log. The procedure does not transpose observations that have a missing value for the ID variable.

IDLABEL Statement

Creates labels for the transposed variables.

Restriction: Must appear after an ID statement.

Featured in: Example 3 on page 1324

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, or print the contents of the output data set by using the CONTENTS statement in the DATASETS procedure. △

VAR Statement

Lists the variables to transpose.

Featured in: Example 4 on page 1325 and Example 6 on page 1329

VAR *variable(s)*;

Required Argument

variable(s)
names one or more variables to transpose.

Details

- If you omit the VAR statement, the then 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.

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.
- 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. Thus, 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 names transposed variables by using the following rules:

1. An ID statement specifies a variable in the input data set whose formatted values become names for the transposed variables.
2. The PREFIX= option specifies a prefix to use in constructing the names of transposed variables.
3. If you do not use an ID statement or the PREFIX= option, then PROC TRANSPOSE looks for an input variable called _NAME_ from which 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

Procedure features:

PROC TRANSPOSE statement option:

OUT=

This example performs a default transposition and uses no subordinate statements.

Program

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.

```
                    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

Procedure features:

PROC TRANSPOSE statement options:

NAME=
PREFIX=

ID statement

Data set: SCORE on page 1322

This example uses the values of a variable and a user-supplied value to name transposed variables.

Program

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

This is the output data set, IDNUMBER.

```
                           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

Procedure features:
- PROC TRANSPOSE statement option:
  - PREFIX=
- IDLABEL statement

Data set: SCORE on page 1322

This example uses the values of the variable in the IDLABEL statement to label transposed variables.

Program

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 headers. The NOOBS option suppresses the printing of observation numbers.

```
proc print data=idlabel label noobs;
   title 'Student Test Scores';
run;
```

Output

This is the output data set, 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
```

Example 4: Transposing BY Groups

Procedure features:

BY statement

VAR statement

Other features: Data set option:
RENAME=

This example illustrates transposing BY groups and selecting variables to transpose.

Program

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

This 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.

```
                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

Procedure 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

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

This is the output data set, CLOSE.

```
        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

Procedure 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

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

```
                          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

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

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.

```
                          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

51

The TRANTAB Procedure

Information about the TRANTAB Procedure

See: For documentation of the TRANTAB procedure, see *SAS National Language Support (NLS): User's Guide*.

CHAPTER

52

The UNIVARIATE Procedure

Information about the UNIVARIATE Procedure

See: The documentation for the UNIVARIATE procedure has moved to Volume 3 of this book.

PART 3

Appendices

APPENDIX

1

SAS Elementary Statistics Procedures

Overview

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 1341 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

$$\frac{1}{d} \sum w_i \left(x_i - \bar{x} \right)^2$$

where d is the variance divisor (the VARDEF= option) that you specify in the PROC statement. Valid values are as follows:

When VARDEF=	d equals . . .
N	n
DF	$n - 1$
WEIGHT	$\sum w_i$
WDF	$\sum w_i - 1$

The default is DF.

z_i
is the standardized variable

$$(x_i - \bar{x})\,/\,s$$

The standard keywords and formulas for each statistic follow. Some formulas use keywords to designate the corresponding statistic.

Table 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	
Standard deviation	X	X	X	X	X	X

Statistic	**PROC MEANS and SUMMARY**	**PROC UNIVARIATE**	**PROC TABULATE**	**PROC REPORT**	**PROC CORR**	**PROC SQL**
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				
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

$$\sum w_i \left(x_i - \bar{x}\right)^2$$

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

$$c_{4_n} \sum z_i^4 - \frac{3\,(n-1)^2}{(n-2)\,(n-3)}$$

where c_{4_n} is $\frac{n(n+1)}{(n-1)(n-2)(n-3)}$. The weighted kurtosis is computed as

$$= c_{4_n} \sum \left((x_i - \overline{x}) / \hat{\sigma}_i\right)^4 - \frac{3\,(n-1)^2}{(n-2)\,(n-3)}$$
$$= c_{4_n} \sum w_i^2 \left((x_i - \overline{x}) / \hat{\sigma}\right)^4 - \frac{3\,(n-1)^2}{(n-2)\,(n-3)}$$

When VARDEF=N, the kurtosis is computed as

$$= \frac{1}{n} \sum z_i^4 - 3$$

and the weighted kurtosis is computed as

$$= \frac{1}{n} \sum \left((x_i - \overline{x}) / \hat{\sigma}_i\right)^4 - 3$$
$$= \frac{1}{n} \sum w_i^2 \left((x_i - \overline{x}) / \hat{\sigma}\right)^4 - 3$$

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 kurtosisis 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 $\overline{x}$.

MIN
: is the minimum value of x_i.

MODE
: is the most frequent value of x_i.

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

$$\mathrm{c}_{3_n} \sum z_i^3$$

where c_{3_n} is $\frac{n}{(n-1)(n-2)}$. The weighted skewness is computed as

$$= \mathrm{c}_{3_n} \sum \left(\left(x_i - \overline{x} \right) / \hat{\sigma}_j \right)^3$$
$$= c_{3_n} \sum w_i^{3/2} \left(\left(x_i - \overline{x} \right) / \hat{\sigma} \right)^3$$

When VARDEF=N, the skewness is computed as

$$= \frac{1}{n} \sum z_i^3$$

and the weighted skewness is computed as

$$= \frac{1}{n} \sum \left(\left(x_i - \overline{x} \right) / \hat{\sigma}_j \right)^3$$
$$= \frac{1}{n} \sum w_i^{3/2} \left(\left(x_i - \overline{x} \right) / \hat{\sigma} \right)^3$$

The formula is invariant under the transformation $w_i^* = zw_i, \ z > 0$. When you use VARDEF=WDF or VARDEF=WEIGHT, the skewnessis set to missing.

Note: PROC MEANS and PROC TABULATE do not compute weighted skewness. △

STDDEV|STD
: is the standard deviation *s* and is computed as the square root of the variance, s^2.

STDERR | STDMEAN
: is the standard error of the mean, computed as

$$s / \sqrt{\sum w_i}$$

when VARDEF=DF, which is the default. Otherwise, STDERR is set to missing.

SUM
: is the sum, computed as

$$\sum w_i x_i$$

SUMWGT
: is the sum of the weights, W, computed as

$$\sum w_i$$

USS
: is the uncorrected sum of squares, computed as

$$\sum w_i x_i^2$$

VAR
: is the variance s^2.

Quantile and Related Statistics

The keywords for quantiles and related statistics are

MEDIAN
: is the middle value.

P1
: is the 1^{st} percentile.

P5
: is the 5^{th} percentile.

P10
: is the 10^{th} percentile.

P90
: is the 90^{th} percentile.

P95
: is the 95^{th} percentile.

P99
: is the 99^{th} percentile.

Q1
: is the lower quartile (25^{th} percentile).

Q3
: is the upper quartile (75^{th} 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

$$\begin{array}{ll} np = j + g & \text{when QNTLDEF} = 1, 2, 3, \text{or } 5 \\ (n+1)\,p = j + g & \text{when QNTLDEF} = 4 \end{array}$$

Here, QNTLDEF= specifies the method that the procedure uses to compute the tth percentile, as shown in the table that follows.

When you use the WEIGHT statement, the tth percentile is computed as

$$y = \begin{cases} \frac{1}{2}\left(x_i + x_{i+1}\right) & \textbf{if } \sum_{j=1}^{i} w_j = pW \\ x_{i+1} & \textbf{if } \sum_{j=1}^{i} w_j < pW < \sum_{j=1}^{i+1} w_j \end{cases}$$

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	

QNTLDEF=	Description	Formula	
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

$$\frac{\overline{x} - \mu_0}{s/\sqrt{\sum w_i}}$$

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
: is the two-tailed *p*-value for Student's t statistic, T, with $n - 1$ degrees of freedom. This 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 $100\left(1 - \alpha\right)$percent confidence interval for the mean has upper and lower limits

$$\overline{x} \pm t_{(1-\alpha/2;n-1)} \frac{s}{\sqrt{\sum w_i}}$$

where s is $\sqrt{\frac{1}{n-1}\sum\left(x_i - \overline{x}\right)^2}$, $t_{(1-\alpha/2;n-1)}$ 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 $100\left(1 - \alpha\right)$percent confidence interval for the mean has the lower limit

$$\overline{x} - t_{(1-\alpha;n-1)} \frac{s}{\sqrt{\sum w_i}}$$

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

$$\overline{x} + t_{(1-\alpha;n-1)} \frac{s}{\sqrt{\sum w_i}}$$

Unless you use VARDEF=DF, which is the default variance divisor, UCLM is set to missing.

Using Weights

For more information on using weights and an example, see "WEIGHT" on page 63.

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 these 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, and PRT require at least two nonmissing observations.
- SKEWNESS requires at least three nonmissing observations.
- KURTOSIS requires at least four nonmissing observations.
- SKEWNESS, KURTOSIS, T, and PROBT require that STD is greater than zero.
- CV requires that MEAN is not equal to zero.
- CLM, LCLM, UCLM, STDERR, T, 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 per se. 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 may 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.

$\mathrm{E}(x)$ 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 $\mathrm{E}\left(x^2\right)$ 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 may 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

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 $\mu = \mathrm{E}\left(x\right)$ 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 may 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 percent 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         1

                   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


                   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         2

  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         3

Frequency

120 +  *
    |  *
    |  **
    | ***
 90 +*****
    |*****
    |*******
    |*******
 60 +*******
    |*********
    |*********
    |*********
 30 +************
    |************   *
    |**************** *
    |***********************  * *
    ---------------------------------------------------
     123456789111111111122222222223333333333444444444444>
              0123456789012345678901234567890123456789=
                                                      5
     PPQ M  MQ                                        0
     511 e  e3      P     P
      0  d  a       9     9
         i  n       0     5
       M a
       o n
       d
       e

                         X Midpoint


                 P5  =  5TH PERCENTILE
                 P10 = 10TH PERCENTILE
                 P90 = 90TH PERCENTILE
                 P95 = 95TH PERCENTILE
                 Q1  =  1ST QUARTILE
                 Q3  =  3RD QUARTILE
```

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 may 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:

$$\sigma^2 = \mathrm{E}\,(x - \mu)^2$$

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 may 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

$$\mathrm{E}\,(x-\mu)^3\,/\sigma^3$$

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

$$\frac{\mathrm{E}\,(x-\mu)^4}{\sigma^4}-3$$

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 may merit attention even in small samples because such values indicate that statistical methods that are based on normality assumptions may 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 percent of the values in a normal population are within one standard deviation of the population mean; approximately 95 percent of the values are within two standard deviations of the mean; and about 99.7 percent are within three standard deviations. Use of the term *normal* to describe this particular kind of distribution does not imply that other kinds 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 belllike, 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         1
            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
```

```
10000 Obs Sample from a Normal Distribution          2
   with Mean=50 and Standard Deviation=10

          The UNIVARIATE Procedure
                Variable:  X

          Quantiles (Definition 5)

           Quantile      Estimate

           100% Max       90.2105
           99%            72.6780
           95%            66.2221
           90%            62.6678
           75% Q3         56.7280
           50% Median     50.0649
           25% Q1         43.4462
           10%            37.1139
           5%             33.5454
           1%             26.9189
           0% Min         13.6971
```

```
10000 Obs Sample from a Normal Distribution          3
   with Mean=50 and Standard Deviation=10

    Frequency

        |              *
    800 +              ***
        |              ****
        |             ******
        |            *******
    600 +            *******
        |           **********
        |          ***********
        |          ***********
    400 +          ************
        |         *************
        |        ***************
        |       *****************
    200 +       ******************
        |      *******************
        |    **********************
        |  ***************************
        --------------------------------
         2    3    4    5    6    7    8
         0    0    0    0    0    0    0

         3    2    1    M    1    2    3
         *    *    *    e    *    *    *
         S    S    S    a    S    S    S
         t    t    t    n    t    t    t
         d    d    d         d    d    d

                   X Midpoint
```

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.

This 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                  1
              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                  2
              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 thesampling 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. This is so 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 belllike, 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 noextrobs=0 normal
                mu0=1;
   var mean;
```

```
run;
```

```
            1000 Sample Means with 10 Obs per Sample           1
             Drawn from an Exponential Distribution

        Frequency

        300 +
            |
            |
            |
            |
        250 +
            |
            |
            |
            |
        200 +
            |
            |
            |
            |
        150 +
            |                 *
            |               * *   *
            |               * * * *
            |               * * * *
        100 +               * * * *
            |               * * * * *
            |             * * * * * *
            |             * * * * * *
            |           * * * * * * *   *
         50 +           * * * * * * * * * *
            |           * * * * * * * * * *
            |           * * * * * * * * * * *
            |         * * * * * * * * * * * *
            |       * * * * * * * * * * * * * * * *
            --------------------------------------------
              0         0         1         1         2
              .         .         .         .         .
              0         5         0         5         0
              5         5         5         5         5

                            Mean Midpoint
```

```
            1000 Sample Means with 10 Obs per Sample           2
             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           1
             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           2
             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 $\mathrm{H}_0 :\ \mu = 0$.

The other two hypotheses, called *alternative* hypotheses, are that the students are underweight on the average, $\mathrm{H}_1: \mu < 0$, and that the students are overweight on the average, $\mathrm{H}_2: \mu > 0$.

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 $\mathrm{H}_1: \mu < 0$.
- If $\bar{x} = 0$, then decide on $\mathrm{H}_0: \mu = 0$.
- If $\bar{x} > 0$, then decide on $\mathrm{H}_2: \mu > 0$.

The trouble with this approach is that there may 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 may be nearly a 50 percent 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 percent. 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 may 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 a roughly 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 percent of the values from a normal distribution are within two standard deviations of the mean, so about 95 percent 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 $\mathrm{H}_1: \mu < 0$.
- If $-8 \leq \bar{x} \leq 8$, then reserve judgment.
- If $\bar{x} > 8$, then decide on $\mathrm{H}_2: \mu > 0$.

If H_0 is true, then in about 95 percent 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 percent of the samples you will make an error; in 2.5 percent of the samples you will incorrectly choose H_1, and in 2.5 percent you will incorrectly choose H_2.

The price 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 percent 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 percent, while –12 and 12 would give a type I error rate of about 0.3 percent.

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 $\mathrm{H}_2 : \ \mu > 0$.

For this one-tailed test, the type I error rate is 2.5 percent, 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 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 percent, so the power for detecting the alternative hypothesis H_2 is about 16 percent. If the population mean were 8, then the power for H_2 would be 50 percent, whereas a population mean of 12 would yield a power of about 84 percent.

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 $\mathrm{H}_1 : \ \mu < 0$.
- □ If $-2.3 \leq t \leq 2.3$, then reserve judgment.
- □ If $t > 2.3$, then decide on $\mathrm{H}_0 : \ \mu > 0$.

The value 2.3 was obtained from a table of Student's t distribution to give a type I error rate of 5 percent 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 percent level of significance and 8 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                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 percent significance level and 8 degrees of freedom). Thus, at a 5 percent significance level you must reserve judgment. If you had elected to use a 10 percent 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 direction(s) 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 percent significance level but not at the 5 percent 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

Descriptions of 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 data 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

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.

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  1992 Allison 31.65 19
Monroe  1992 Barry   23.76 16
Monroe  1992 Candace 21.11  5
Monroe  1992 Danny    6.89 23
Monroe  1992 Edward  53.76 31
Monroe  1992 Fiona   48.55 13
Monroe  1992 Gert    24.00 16
Monroe  1992 Harold  27.55 17
Monroe  1992 Ima     15.98  9
Monroe  1992 Jack    20.00 23
Monroe  1992 Katie   22.11  2
Monroe  1992 Lisa    18.34 17
Monroe  1992 Tonya   55.16 40
Monroe  1992 Max     26.77 34
Monroe  1992 Ned     28.43 22
Monroe  1992 Opal    32.66 14
Monroe  1993 Patsy   18.33 18
Monroe  1993 Quentin 16.89 15
```

```
Monroe  1993 Randall 12.98 17
Monroe  1993 Sam     15.88  5
Monroe  1993 Tyra    21.88 23
Monroe  1993 Myrtle  47.33 26
Monroe  1993 Frank   41.11 22
Monroe  1993 Cameron 65.44 14
Monroe  1993 Vern    17.89 11
Monroe  1993 Wendell 23.00 10
Monroe  1993 Bob     26.88  6
Monroe  1993 Leah    28.99 23
Monroe  1994 Becky   30.33 26
Monroe  1994 Sally   35.75 27
Monroe  1994 Edgar   27.11 12
Monroe  1994 Dawson  17.24 16
Monroe  1994 Lou      5.12 16
Monroe  1994 Damien  18.74 17
Monroe  1994 Mona    27.43  7
Monroe  1994 Della   56.78 15
Monroe  1994 Monique 29.88 19
Monroe  1994 Carl    31.12 25
Monroe  1994 Reba    35.16 22
Monroe  1994 Dax     27.65 23
Monroe  1994 Gary    23.11 15
Monroe  1994 Suzie   26.65 11
Monroe  1994 Benito  47.44 18
Monroe  1994 Thomas  21.99 23
Monroe  1994 Annie   24.99 27
Monroe  1994 Paul    27.98 22
Monroe  1994 Alex    24.00 16
Monroe  1994 Lauren  15.00 17
Monroe  1994 Julia   12.98 15
Monroe  1994 Keith   11.89 19
Monroe  1994 Jackie  26.88 22
Monroe  1994 Pablo   13.98 28
Monroe  1994 L.T.    56.87 33
Monroe  1994 Willard 78.65 24
Monroe  1994 Kathy   32.88 11
Monroe  1994 Abby    35.88 10
Kennedy 1992 Arturo  34.98 14
Kennedy 1992 Grace   27.55 25
Kennedy 1992 Winston 23.88 22
Kennedy 1992 Vince   12.88 21
Kennedy 1992 Claude  15.62  5
Kennedy 1992 Mary    28.99 34
Kennedy 1992 Abner   25.89 22
Kennedy 1992 Jay     35.89 35
Kennedy 1992 Alicia  28.77 26
Kennedy 1992 Freddy  29.00 27
Kennedy 1992 Eloise  31.67 25
Kennedy 1992 Jenny   43.89 22
Kennedy 1992 Thelma  52.63 21
Kennedy 1992 Tina    19.67 21
Kennedy 1992 Eric    24.89 12
Kennedy 1993 Bubba   37.88 12
```

```
Kennedy 1993 G.L.    25.89 21
Kennedy 1993 Bert    28.89 21
Kennedy 1993 Clay    26.44 21
Kennedy 1993 Leeann  27.17 17
Kennedy 1993 Georgia 38.90 11
Kennedy 1993 Bill    42.23 25
Kennedy 1993 Holly   18.67 27
Kennedy 1993 Benny   19.09 25
Kennedy 1993 Cammie  28.77 28
Kennedy 1993 Amy     27.08 31
Kennedy 1993 Doris   22.22 24
Kennedy 1993 Robbie  19.80 24
Kennedy 1993 Ted     27.07 25
Kennedy 1993 Sarah   24.44 12
Kennedy 1993 Megan   28.89 11
Kennedy 1993 Jeff    31.11 12
Kennedy 1993 Taz     30.55 11
Kennedy 1993 George  27.56 11
Kennedy 1993 Heather 38.67 15
Kennedy 1994 Nancy   29.90 26
Kennedy 1994 Rusty   30.55 28
Kennedy 1994 Mimi    37.67 22
Kennedy 1994 J.C.    23.33 27
Kennedy 1994 Clark   27.90 25
Kennedy 1994 Rudy    27.78 23
Kennedy 1994 Samuel  34.44 18
Kennedy 1994 Forrest 28.89 26
Kennedy 1994 Luther  72.22 24
Kennedy 1994 Trey     6.78 18
Kennedy 1994 Albert  23.33 19
Kennedy 1994 Che-Min 26.66 33
Kennedy 1994 Preston 32.22 23
Kennedy 1994 Larry   40.00 26
Kennedy 1994 Anton   35.99 28
Kennedy 1994 Sid     27.45 25
Kennedy 1994 Will    28.88 21
Kennedy 1994 Morty   34.44 25
;
```

CUSTOMER_RESPONSE

```
data customer_response;
   input Customer Factor1-Factor4 Source1-Source3
        Quality1-Quality3;
   datalines;
1 . . 1 1 1 1 . 1 . .
2 1 1 . 1 1 1 . 1 1 .
3 . . 1 1 1 1 . . . .
4 1 1 . 1 . 1 . . . 1
5 . 1 . 1 1 . . . . 1
6 . 1 . 1 1 . . . . .
7 . 1 . 1 1 . . 1 . .
```

```
8 1 . . 1 1 1 . 1 1 .
9 1 1 . 1 1 . . . . 1
10 1 . . 1 1 1 . 1 1 .
11 1 1 1 1 . 1 . 1 1 1
12 1 1 . 1 1 1 . . . .
13 1 1 . 1 . 1 . 1 1 .
14 1 1 . 1 1 1 . . . .
15 1 1 . 1 . 1 . 1 1 1
16 1 . . 1 1 . . 1 . .
17 1 1 . 1 1 1 . . 1 .
18 1 1 . 1 1 1 1 . . 1
19 . 1 . 1 1 1 1 . 1 .
20 1 . . 1 1 1 . 1 1 1
21 . . . 1 1 1 . 1 . .
22 . . . 1 1 1 . 1 1 .
23 1 . . 1 . . . . . 1
24 . 1 . 1 1 . . 1 . 1
25 1 1 . 1 1 . . . 1 1
26 1 1 . 1 1 . . 1 . .
27 1 . . 1 1 . . . 1 .
28 1 1 . 1 . . . 1 1 1
29 1 . . 1 1 1 . 1 . 1
30 1 . 1 1 1 . . 1 1 .
31 . . . 1 1 . . 1 1 .
32 1 1 1 1 1 . . 1 1 1
33 1 . . 1 1 . . 1 . 1
34 . . 1 1 . . . 1 1 .
35 1 1 1 1 1 . 1 1 . .
36 1 1 1 1 . 1 . 1 . .
37 1 1 . 1 . . . 1 . .
38 . . . 1 1 1 . 1 . .
39 1 1 . 1 1 . . 1 . 1
40 1 . . 1 . . 1 1 . 1
41 1 . . 1 1 1 1 1 . 1
42 1 1 1 1 . . 1 1 . .
43 1 . . 1 1 1 . 1 . .
44 1 . 1 1 . 1 . 1 . 1
45 . . . 1 . . 1 . . 1
46 . . . 1 1 . . . 1 .
47 1 1 . 1 . . 1 1 . .
48 1 . 1 1 1 . 1 1 . .
49 . . 1 1 1 1 . 1 . 1
50 . 1 . 1 1 . . 1 1 .
51 1 . 1 1 1 1 . . . .
52 1 1 1 1 1 1 . 1 . .
53 . 1 1 1 . 1 . 1 1 1
54 1 . . 1 1 . . 1 1 .
55 1 1 . 1 1 1 . 1 . .
56 1 . . 1 1 . . 1 1 .
57 1 1 . 1 1 . 1 . . 1
58 . 1 . 1 . 1 . . 1 1
59 1 1 1 1 . . 1 1 1 .
60 . 1 1 1 1 1 . . 1 1
61 1 1 1 1 1 1 . 1 . .
```

```
62 1 1 . 1 1 . . 1 1 .
63 . . . 1 . . . 1 1 1
64 1 . . 1 1 1 . 1 . .
65 1 . . 1 1 1 . 1 . .
66 1 . . 1 1 1 1 1 1 .
67 1 1 . 1 1 1 . 1 1 .
68 1 1 . 1 1 1 . 1 1 .
69 1 1 . 1 1 . 1 . . .
70 . . . 1 1 1 . 1 . .
71 1 . . 1 1 . 1 . . 1
72 1 . 1 1 1 1 . . 1 .
73 1 1 . 1 . 1 . 1 1 .
74 1 1 1 1 1 1 . 1 . .
75 . 1 . 1 1 1 . . 1 .
76 1 1 . 1 1 1 . 1 1 1
77 . . . 1 1 1 . . . .
78 1 1 1 1 1 1 . 1 1 .
79 1 . . 1 1 1 . 1 1 .
80 1 1 1 1 1 . 1 1 . 1
81 1 1 . 1 1 1 1 1 1 .
82 . . . 1 1 1 1 . . .
83 1 1 . 1 1 1 . 1 1 .
84 1 . . 1 1 . . 1 1 .
85 . . . 1 . 1 . 1 . .
86 1 . . 1 1 1 . 1 1 1
87 1 1 . 1 1 1 . 1 . .
88 . . . 1 . 1 . . . .
89 1 . . 1 . 1 . . 1 1
90 1 1 . 1 1 1 . 1 . 1
91 . . . 1 1 . . . 1 .
92 1 . . 1 1 1 . 1 1 .
93 1 . . 1 1 . . 1 1 .
94 1 . . 1 1 1 1 1 . .
95 1 . . 1 . 1 1 1 1 .
96 1 . 1 1 1 1 . . 1 .
97 1 1 . 1 1 . . . 1 .
98 1 . 1 1 1 1 1 1 . .
99 1 1 . 1 1 1 1 1 1 .
100 1 . 1 1 1 . . . 1 1
101 1 . 1 1 1 1 . . . .
102 1 . . 1 1 . 1 1 . .
103 1 1 . 1 1 1 . 1 . .
104 . . . 1 1 1 . 1 1 1
105 1 . 1 1 1 . . 1 . 1
106 1 1 1 1 1 1 1 1 1 1
107 1 1 1 1 . . . 1 . 1
108 1 . . 1 . 1 1 1 . .
109 . 1 . 1 1 . . 1 1 .
110 1 . . 1 . . . . . .
111 1 . . 1 1 1 . 1 1 .
112 1 1 . 1 1 1 . . . 1
113 1 1 . 1 1 . 1 1 1 .
114 1 1 . 1 1 . . . . .
115 1 1 . 1 1 . . 1 . .
```

```
116 . 1 . 1 1 1 1 1 . .
117 . 1 . 1 1 1 . . . .
118 . 1 1 1 1 . . 1 1 .
119 . . . 1 . . . 1 . .
120 1 1 . 1 . . . . 1 .
;
```

DJIA

```
data djia;
      input Year @7 HighDate date7. High @24 LowDate date7. Low;
      format highdate lowdate date7.;
      datalines;
1954  31DEC54  404.39  11JAN54  279.87
1955  30DEC55  488.40  17JAN55  388.20
1956  06APR56  521.05  23JAN56  462.35
1957  12JUL57  520.77  22OCT57  419.79
1958  31DEC58  583.65  25FEB58  436.89
1959  31DEC59  679.36  09FEB59  574.46
1960  05JAN60  685.47  25OCT60  568.05
1961  13DEC61  734.91  03JAN61  610.25
1962  03JAN62  726.01  26JUN62  535.76
1963  18DEC63  767.21  02JAN63  646.79
1964  18NOV64  891.71  02JAN64  768.08
1965  31DEC65  969.26  28JUN65  840.59
1966  09FEB66  995.15  07OCT66  744.32
1967  25SEP67  943.08  03JAN67  786.41
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
;
```

EDUCATION

```
data education;
   input State $14. +1 Code $ DropoutRate Expenditures MathScore Region $;
   label dropoutrate='Dropout Percentage - 1989'
          expenditures='Expenditure Per Pupil - 1989'
            mathscore='8th Grade Math Exam - 1990';
   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     15SEP48      07JUN75    203/781-1255
1653    Alexander  Susan     Bridgeport   CT
F       ME2        35108     18OCT52      12AUG78    203/675-7715
1400    Apple      Troy      New York     NY
M       ME1        29769     08NOV55      19OCT78    212/586-0808
1350    Arthur     Barbara   New York     NY
F       FA3        32886     03SEP53      01AUG78    718/383-1549
1401    Avery      Jerry     Paterson     NJ
M       TA3        38822     16DEC38      20NOV73    201/732-8787
1499    Barefoot   Joseph    Princeton    NJ
M       ME3        43025     29APR42      10JUN68    201/812-5665
1101    Baucom     Walter    New York     NY
M       SCP        18723     09JUN50      04OCT78    212/586-8060
1333    Blair      Justin    Stamford     CT
M       PT2        88606     02APR49      13FEB69    203/781-1777
1402    Blalock    Ralph     New York     NY
M       TA2        32615     20JAN51      05DEC78    718/384-2849
1479    Bostic     Marie     New York     NY
F       TA3        38785     25DEC56      08OCT77    718/384-8816
1403    Bowden     Earl      Bridgeport   CT
M       ME1        28072     31JAN57      24DEC79    203/675-3434
1739    Boyce      Jonathan  New York     NY
M       PT1        66517     28DEC52      30JAN79    212/587-1247
1658    Bradley    Jeremy    New York     NY
M       SCP        17943     11APR55      03MAR80    212/587-3622
1428    Brady      Christine Stamford     CT
F       PT1        68767     07APR58      19NOV79    203/781-1212
1782    Brown      Jason     Stamford     CT
M       ME2        35345     07DEC58      25FEB80    203/781-0019
1244    Bryant     Leonard   New York     NY
M       ME2        36925     03SEP51      20JAN76    718/383-3334
1383    Burnette   Thomas    New York     NY
M       BCK        25823     28JAN56      23OCT80    718/384-3569
1574    Cahill     Marshall  New York     NY
M       FA2        28572     30APR48      23DEC80    718/383-2338
1789    Caraway    Davis     New York     NY
M       SCP        18326     28JAN45      14APR66    212/587-9000
1404    Carter     Donald    New York     NY
M       PT2        91376     27FEB41      04JAN68    718/384-2946
1437    Carter     Dorothy   Bridgeport   CT
F       A3         33104     23SEP48      03SEP72    203/675-4117
1639    Carter     Karen     Stamford     CT
```

```
F       A3          40260      29JUN45       31JAN72     203/781-8839
1269    Caston      Franklin   Stamford      CT
M       NA1         41690      06MAY60       01DEC80     203/781-3335
1065    Chapman     Neil       New York      NY
M       ME2         35090      29JAN32       10JAN75     718/384-5618
1876    Chin        Jack       New York      NY
M       TA3         39675      23MAY46       30APR73     212/588-5634
1037    Chow        Jane       Stamford      CT
F       TA1         28558      13APR52       16SEP80     203/781-8868
1129    Cook        Brenda     New York      NY
F       ME2         34929      11DEC49       20AUG79     718/383-2313
1988    Cooper      Anthony    New York      NY
M       FA3         32217      03DEC47       21SEP72     212/587-1228
1405    Davidson    Jason      Paterson      NJ
M       SCP         18056      08MAR54       29JAN80     201/732-2323
1430    Dean        Sandra     Bridgeport    CT
F       TA2         32925      03MAR50       30APR75     203/675-1647
1983    Dean        Sharon     New York      NY
F       FA3         33419      03MAR50       30APR75     718/384-1647
1134    Delgado     Maria      Stamford      CT
F       TA2         33462      08MAR57       24DEC76     203/781-1528
1118    Dennis      Roger      New York      NY
M       PT3         111379     19JAN32       21DEC68     718/383-1122
1438    Donaldson   Karen      Stamford      CT
F       TA3         39223      18MAR53       21NOV75     203/781-2229
1125    Dunlap      Donna      New York      NY
F       FA2         28888      11NOV56       14DEC75     718/383-2094
1475    Eaton       Alicia     New York      NY
F       FA2         27787      18DEC49       16JUL78     718/383-2828
1117    Edgerton    Joshua     New York      NY
M       TA3         39771      08JUN51       16AUG80     212/588-1239
1935    Fernandez   Katrina    Bridgeport    CT
F       NA2         51081      31MAR42       19OCT69     203/675-2962
1124    Fields      Diana      White Plains NY
F       FA1         23177      13JUL46       04OCT78     914/455-2998
1422    Fletcher    Marie      Princeton     NJ
F       FA1         22454      07JUN52       09APR79     201/812-0902
1616    Flowers     Annette    New York      NY
F       TA2         34137      04MAR58       07JUN81     718/384-3329
1406    Foster      Gerald     Bridgeport    CT
M       ME2         35185      11MAR49       20FEB75     203/675-6363
1120    Garcia      Jack       New York      NY
M       ME1         28619      14SEP60       10OCT81     718/384-4930
1094    Gomez       Alan       Bridgeport    CT
M       FA1         22268      05APR58       20APR79     203/675-7181
1389    Gordon      Levi       New York      NY
M       BCK         25028      18JUL47       21AUG78     718/384-9326
1905    Graham      Alvin      New York      NY
M       PT1         65111      19APR60       01JUN80     212/586-8815
1407    Grant       Daniel     Mt. Vernon    NY
M       PT1         68096      26MAR57       21MAR78     914/468-1616
1114    Green       Janice     New York      NY
F       TA2         32928      21SEP57       30JUN75     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
;
```

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     01MAR94  LGA   LAX   1-10 Minutes     Domestic                8
202     01MAR94  LGA   ORD   No Delay         Domestic               -5
219     01MAR94  LGA   LON   11+ Minutes      International          18
622     01MAR94  LGA   FRA   No Delay         International          -5
132     01MAR94  LGA   YYZ   11+ Minutes      International          14
271     01MAR94  LGA   PAR   1-10 Minutes     International           5
302     01MAR94  LGA   WAS   No Delay         Domestic               -2
114     02MAR94  LGA   LAX   No Delay         Domestic                0
202     02MAR94  LGA   ORD   1-10 Minutes     Domestic                5
219     02MAR94  LGA   LON   11+ Minutes      International          18
622     02MAR94  LGA   FRA   No Delay         International           0
132     02MAR94  LGA   YYZ   1-10 Minutes     International           5
271     02MAR94  LGA   PAR   1-10 Minutes     International           4
302     02MAR94  LGA   WAS   No Delay         Domestic                0
114     03MAR94  LGA   LAX   No Delay         Domestic               -1
202     03MAR94  LGA   ORD   No Delay         Domestic               -1
219     03MAR94  LGA   LON   1-10 Minutes     International           4
622     03MAR94  LGA   FRA   No Delay         International          -2
132     03MAR94  LGA   YYZ   1-10 Minutes     International           6
271     03MAR94  LGA   PAR   1-10 Minutes     International           2
302     03MAR94  LGA   WAS   1-10 Minutes     Domestic                5
114     04MAR94  LGA   LAX   11+ Minutes      Domestic               15
202     04MAR94  LGA   ORD   No Delay         Domestic               -5
219     04MAR94  LGA   LON   1-10 Minutes     International           3
622     04MAR94  LGA   FRA   11+ Minutes      International          30
132     04MAR94  LGA   YYZ   No Delay         International          -5
271     04MAR94  LGA   PAR   1-10 Minutes     International           5
```

```
302     04MAR94  LGA   WAS   1-10 Minutes     Domestic             7
114     05MAR94  LGA   LAX   No Delay         Domestic            -2
202     05MAR94  LGA   ORD   1-10 Minutes     Domestic             2
219     05MAR94  LGA   LON   1-10 Minutes     International        3
622     05MAR94  LGA   FRA   No Delay         International       -6
132     05MAR94  LGA   YYZ   1-10 Minutes     International        3
271     05MAR94  LGA   PAR   1-10 Minutes     International        5
114     06MAR94  LGA   LAX   No Delay         Domestic            -1
202     06MAR94  LGA   ORD   No Delay         Domestic            -3
219     06MAR94  LGA   LON   11+ Minutes      International       27
132     06MAR94  LGA   YYZ   1-10 Minutes     International        7
302     06MAR94  LGA   WAS   1-10 Minutes     Domestic             1
114     07MAR94  LGA   LAX   No Delay         Domestic            -1
202     07MAR94  LGA   ORD   No Delay         Domestic            -2
219     07MAR94  LGA   LON   11+ Minutes      International       15
622     07MAR94  LGA   FRA   11+ Minutes      International       21
132     07MAR94  LGA   YYZ   No Delay         International       -2
271     07MAR94  LGA   PAR   1-10 Minutes     International        4
302     07MAR94  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     01MAR94  LON        198
622     01MAR94  FRA        207
132     01MAR94  YYZ        115
271     01MAR94  PAR        138
219     02MAR94  LON        147
622     02MAR94  FRA        176
132     02MAR94  YYZ        106
271     02MAR94  PAR        172
219     03MAR94  LON        197
622     03MAR94  FRA        180
132     03MAR94  YYZ         75
271     03MAR94  PAR        147
219     04MAR94  LON        232
622     04MAR94  FRA        137
132     04MAR94  YYZ        117
271     04MAR94  PAR        146
219     05MAR94  LON        160
622     05MAR94  FRA        185
132     05MAR94  YYZ        157
271     05MAR94  PAR        177
219     06MAR94  LON        163
132     06MAR94  YYZ        150
219     07MAR94  LON        241
622     07MAR94  FRA        210
132     07MAR94  YYZ        164
271     07MAR94  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     01MAR94    7:10  LGA   LAX       2475       172       210
202     01MAR94   10:43  LGA   ORD        740       151       210
219     01MAR94    9:31  LGA   LON       3442       198       250
622     01MAR94   12:19  LGA   FRA       3857       207       250
132     01MAR94   15:35  LGA   YYZ        366       115       178
271     01MAR94   13:17  LGA   PAR       3635       138       250
302     01MAR94   20:22  LGA   WAS        229       105       180
114     02MAR94    7:10  LGA   LAX       2475       119       210
202     02MAR94   10:43  LGA   ORD        740       120       210
219     02MAR94    9:31  LGA   LON       3442       147       250
622     02MAR94   12:19  LGA   FRA       3857       176       250
132     02MAR94   15:35  LGA   YYZ        366       106       178
302     02MAR94   20:22  LGA   WAS        229        78       180
271     02MAR94   13:17  LGA   PAR       3635       104       250
114     03MAR94    7:10  LGA   LAX       2475       197       210
202     03MAR94   10:43  LGA   ORD        740       118       210
219     03MAR94    9:31  LGA   LON       3442       197       250
622     03MAR94   12:19  LGA   FRA       3857       180       250
132     03MAR94   15:35  LGA   YYZ        366        75       178
271     03MAR94   13:17  LGA   PAR       3635       147       250
302     03MAR94   20:22  LGA   WAS        229       123       180
114     04MAR94    7:10  LGA   LAX       2475       178       210
202     04MAR94   10:43  LGA   ORD        740       148       210
219     04MAR94    9:31  LGA   LON       3442       232       250
622     04MAR94   12:19  LGA   FRA       3857       137       250
132     04MAR94   15:35  LGA   YYZ        366       117       178
271     04MAR94   13:17  LGA   PAR       3635       146       250
302     04MAR94   20:22  LGA   WAS        229       115       180
114     05MAR94    7:10  LGA   LAX       2475       117       210
202     05MAR94   10:43  LGA   ORD        740       104       210
219     05MAR94    9:31  LGA   LON       3442       160       250
622     05MAR94   12:19  LGA   FRA       3857       185       250
```

```
132     05MAR94   15:35  LGA   YYZ        366       157       178
271     05MAR94   13:17  LGA   PAR       3635       177       250
114     06MAR94    7:10  LGA   LAX       2475       128       210
202     06MAR94   10:43  LGA   ORD        740       115       210
219     06MAR94    9:31  LGA   LON       3442       163       250
132     06MAR94   15:35  LGA   YYZ        366       150       178
302     06MAR94   20:22  LGA   WAS        229        66       180
114     07MAR94    7:10  LGA   LAX       2475       160       210
202     07MAR94   10:43  LGA   ORD        740       175       210
219     07MAR94    9:31  LGA   LON       3442       241       250
622     07MAR94   12:19  LGA   FRA       3857       210       250
132     07MAR94   15:35  LGA   YYZ        366       164       178
271     07MAR94   13:17  LGA   PAR       3635       155       250
302     07MAR94   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 on page 1129 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     01MAR94  YYZ   1739
132     01MAR94  YYZ   1478
132     01MAR94  YYZ   1130
132     01MAR94  YYZ   1390
132     01MAR94  YYZ   1983
132     01MAR94  YYZ   1111
219     01MAR94  LON   1407
219     01MAR94  LON   1777
219     01MAR94  LON   1103
219     01MAR94  LON   1125
219     01MAR94  LON   1350
219     01MAR94  LON   1332
271     01MAR94  PAR   1439
271     01MAR94  PAR   1442
271     01MAR94  PAR   1132
271     01MAR94  PAR   1411
271     01MAR94  PAR   1988
271     01MAR94  PAR   1443
622     01MAR94  FRA   1545
622     01MAR94  FRA   1890
622     01MAR94  FRA   1116
622     01MAR94  FRA   1221
622     01MAR94  FRA   1433
622     01MAR94  FRA   1352
132     02MAR94  YYZ   1556
132     02MAR94  YYZ   1478
132     02MAR94  YYZ   1113
132     02MAR94  YYZ   1411
132     02MAR94  YYZ   1574
132     02MAR94  YYZ   1111
219     02MAR94  LON   1407
219     02MAR94  LON   1118
219     02MAR94  LON   1132
219     02MAR94  LON   1135
```

```
219     02MAR94  LON   1441
219     02MAR94  LON   1332
271     02MAR94  PAR   1739
271     02MAR94  PAR   1442
271     02MAR94  PAR   1103
271     02MAR94  PAR   1413
271     02MAR94  PAR   1115
271     02MAR94  PAR   1443
622     02MAR94  FRA   1439
622     02MAR94  FRA   1890
622     02MAR94  FRA   1124
622     02MAR94  FRA   1368
622     02MAR94  FRA   1477
622     02MAR94  FRA   1352
132     03MAR94  YYZ   1739
132     03MAR94  YYZ   1928
132     03MAR94  YYZ   1425
132     03MAR94  YYZ   1135
132     03MAR94  YYZ   1437
132     03MAR94  YYZ   1111
219     03MAR94  LON   1428
219     03MAR94  LON   1442
219     03MAR94  LON   1130
219     03MAR94  LON   1411
219     03MAR94  LON   1115
219     03MAR94  LON   1332
271     03MAR94  PAR   1905
271     03MAR94  PAR   1118
271     03MAR94  PAR   1970
271     03MAR94  PAR   1125
271     03MAR94  PAR   1983
271     03MAR94  PAR   1443
622     03MAR94  FRA   1545
622     03MAR94  FRA   1830
622     03MAR94  FRA   1414
622     03MAR94  FRA   1368
622     03MAR94  FRA   1431
622     03MAR94  FRA   1352
132     04MAR94  YYZ   1428
132     04MAR94  YYZ   1118
132     04MAR94  YYZ   1103
132     04MAR94  YYZ   1390
132     04MAR94  YYZ   1350
132     04MAR94  YYZ   1111
219     04MAR94  LON   1739
219     04MAR94  LON   1478
219     04MAR94  LON   1130
219     04MAR94  LON   1125
219     04MAR94  LON   1983
219     04MAR94  LON   1332
271     04MAR94  PAR   1407
271     04MAR94  PAR   1410
271     04MAR94  PAR   1094
271     04MAR94  PAR   1411
```

```
271     04MAR94  PAR   1115
271     04MAR94  PAR   1443
622     04MAR94  FRA   1545
622     04MAR94  FRA   1890
622     04MAR94  FRA   1116
622     04MAR94  FRA   1221
622     04MAR94  FRA   1433
622     04MAR94  FRA   1352
132     05MAR94  YYZ   1556
132     05MAR94  YYZ   1890
132     05MAR94  YYZ   1113
132     05MAR94  YYZ   1475
132     05MAR94  YYZ   1431
132     05MAR94  YYZ   1111
219     05MAR94  LON   1428
219     05MAR94  LON   1442
219     05MAR94  LON   1422
219     05MAR94  LON   1413
219     05MAR94  LON   1574
219     05MAR94  LON   1332
271     05MAR94  PAR   1739
271     05MAR94  PAR   1928
271     05MAR94  PAR   1103
271     05MAR94  PAR   1477
271     05MAR94  PAR   1433
271     05MAR94  PAR   1443
622     05MAR94  FRA   1545
622     05MAR94  FRA   1830
622     05MAR94  FRA   1970
622     05MAR94  FRA   1441
622     05MAR94  FRA   1350
622     05MAR94  FRA   1352
132     06MAR94  YYZ   1333
132     06MAR94  YYZ   1890
132     06MAR94  YYZ   1414
132     06MAR94  YYZ   1475
132     06MAR94  YYZ   1437
132     06MAR94  YYZ   1111
219     06MAR94  LON   1106
219     06MAR94  LON   1118
219     06MAR94  LON   1425
219     06MAR94  LON   1434
219     06MAR94  LON   1555
219     06MAR94  LON   1332
132     07MAR94  YYZ   1407
132     07MAR94  YYZ   1118
132     07MAR94  YYZ   1094
132     07MAR94  YYZ   1555
132     07MAR94  YYZ   1350
132     07MAR94  YYZ   1111
219     07MAR94  LON   1905
219     07MAR94  LON   1478
219     07MAR94  LON   1124
219     07MAR94  LON   1434
```

```
219     07MAR94  LON   1983
219     07MAR94  LON   1332
271     07MAR94  PAR   1410
271     07MAR94  PAR   1777
271     07MAR94  PAR   1103
271     07MAR94  PAR   1574
271     07MAR94  PAR   1115
271     07MAR94  PAR   1443
622     07MAR94  FRA   1107
622     07MAR94  FRA   1890
622     07MAR94  FRA   1425
622     07MAR94  FRA   1475
622     07MAR94  FRA   1433
622     07MAR94  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.SUPERV

```
data proclib.superv;
   input supid $4. +8 state $2. +5  jobcat  $2.;
   label supid='Supervisor Id' jobcat='Job Category';
   datalines;
1677        CT     BC
1834        NY     BC
1431        CT     FA
1433        NJ     FA
1983        NY     FA
1385        CT     ME
1420        NJ     ME
1882        NY     ME
1935        CT     NA
1417        NJ     NA
1352        NY     NA
1106        CT     PT
1442        NJ     PT
1118        NY     PT
1405        NJ     SC
1564        NY     SC
1639        CT     TA
1401        NJ     TA
1126        NY     TA
;
```

RADIO

This DATA step uses an INFILE statement to read data that is stored in an external file.

```
data radio;
   infile 'input-file' missover;
   input /(time1-time7) ($1. +1);
   listener=_n_;
run;
```

Here is the data that is stored in the external file:

```
967 32 f 5 3 5
7 5 5 5 7 0 0 0 8 7 0 0 8 0
781 30 f 2 3 5
5 0 0 0 5 0 0 0 4 7 5 0 0 0
859 39 f 1 0 5
1 0 0 0 1 0 0 0 0 0 0 0 0 0
859 40 f 6 1 5
7 5 0 5 7 0 0 0 0 0 0 5 0 0
467 37 m 2 3 1
1 5 5 5 5 4 4 8 8 0 0 0 0 0
220 35 f 3 1 7
7 0 0 0 7 0 0 0 7 0 0 0 0 0
833 42 m 2 2 4
7 0 0 0 7 5 4 7 4 0 1 4 4 0
967 39 f .5 1 7
7 0 0 0 7 7 0 0 0 0 0 0 8 0
677 28 m .5 .5 7
7 0 0 0 0 0 0 0 0 0 0 0 0 0
833 28 f 3 4 1
1 0 0 0 0 1 1 1 1 0 0 0 1 1
677 24 f 3 1 2
2 0 0 0 0 0 0 2 0 8 8 0 0 0
688 32 m 5 2 4
5 5 0 4 8 0 0 5 0 8 0 0 0 0
542 38 f 6 8 5
5 0 0 5 5 5 0 5 5 5 5 5 5 0
677 27 m 6 1 1
1 1 0 4 4 0 0 1 4 0 0 0 0 0
779 37 f 2.5 4 7
7 0 0 0 7 7 0 7 7 4 4 7 8 0
362 31 f 1 2 2
8 0 0 0 8 0 0 0 0 0 8 8 0 0
859 29 m 10 3 4
4 4 0 2 2 0 0 4 0 0 0 4 4 0
467 24 m 5 8 1
7 1 1 1 7 1 1 0 1 7 1 1 1 1
851 34 m 1 2 8
0 0 0 0 8 0 0 0 4 0 0 0 8 0
859 23 f 1 1 8
8 0 0 0 8 0 0 0 0 0 0 0 0 8
781 34 f 9 3 1
2 1 0 1 4 4 4 0 1 1 1 1 4 4
851 40 f 2 4 5
5 0 0 0 5 0 0 5 0 0 5 5 0 0
783 34 m 3 2 4
7 0 0 0 7 4 4 0 0 4 4 0 0 0
848 29 f 4 1.5 7
7 4 4 1 7 0 0 0 7 0 0 7 0 0
851 28 f 1 2 2
2 0 2 0 2 0 0 0 0 2 2 2 0 0
856 42 f 1.5 1 2
2 0 0 0 0 0 0 2 0 0 0 0 0 0
859 29 m .5 .5 5
```

```
5 0 0 0 1 0 0 0 0 0 8 8 5 0
833 29 m 1 3 2
2 0 0 0 2 2 0 0 4 2 0 2 0 0
859 23 f 10 3 1
1 5 0 8 8 1 4 0 1 1 1 1 1 4
781 37 f .5 2 7
7 0 0 0 1 0 0 0 1 7 0 1 0 0
833 31 f 5 4 1
1 0 0 0 1 0 0 0 4 0 4 0 0 0
942 23 f 4 2 1
1 0 0 0 1 0 1 0 1 1 0 0 0 0
848 33 f 5 4 1
1 1 0 1 1 0 0 0 1 1 1 0 0 0
222 33 f 2 0 1
1 0 0 0 1 0 0 0 0 0 0 0 0 0
851 45 f .5 1 8
8 0 0 0 8 0 0 0 0 0 8 0 0 0
848 27 f 2 4 1
1 0 0 0 1 1 0 0 4 1 1 1 1 1
781 38 m 2 2 1
5 0 0 0 1 0 0 0 0 0 1 1 0 0
222 27 f 3 1 2
2 0 2 0 2 2 0 0 2 0 0 0 0 0
467 34 f 2 2 1
1 0 0 0 0 1 0 1 0 0 0 0 1 0
833 27 f 8 8 1
7 0 1 0 7 4 0 0 1 1 1 4 1 0
677 49 f 1.5 0 8
8 0 8 0 8 0 0 0 0 0 0 0 0 0
849 43 m 1 4 1
1 0 0 0 4 0 0 0 4 0 1 0 0 0
467 28 m 2 1 7
7 0 0 0 7 0 0 7 0 0 1 0 0 0
732 29 f 1 0 2
2 0 0 0 2 0 0 0 0 0 0 0 0 0
851 31 m 2 2 2
2 5 0 6 0 0 8 0 2 2 8 2 0 0
779 42 f 8 2 2
7 2 0 2 7 0 0 0 0 0 0 0 2 0
493 40 m 1 3 3
3 0 0 0 5 3 0 5 5 0 0 0 1 1
859 30 m 1 0 7
7 0 0 0 7 0 0 0 0 0 0 0 0 0
833 36 m 4 2 5
7 5 0 5 0 5 0 0 7 0 0 0 5 0
467 30 f 1 4 1
0 0 0 0 1 0 6 0 0 1 1 1 0 6
859 32 f 3 5 2
2 2 2 2 2 2 6 6 2 2 2 2 2 6
851 43 f 8 1 5
7 5 5 5 0 0 0 4 0 0 0 0 0 0
848 29 f 3 5 1
7 0 0 0 7 1 0 0 1 1 1 1 1 0
833 25 f 2 4 5
```

```
7 0 0 0 5 7 0 0 7 5 0 0 5 0
783 33 f 8 3 8
8 0 8 0 7 0 0 0 8 0 5 4 0 5
222 26 f 10 2 1
1 1 0 1 1 0 0 0 3 1 1 0 0 0
222 23 f 3 2 2
2 2 2 2 7 0 0 2 2 0 0 0 0 0
859 50 f 1 5 4
7 0 0 0 7 0 0 5 4 4 4 7 0 0
833 26 f 3 2 1
1 0 0 1 1 0 0 5 5 0 1 0 0 0
467 29 m 7 2 1
1 1 1 1 1 0 0 1 1 1 0 0 0 0
859 35 m .5 2 2
7 0 0 0 2 0 0 7 5 0 0 4 0 0
833 33 f 3 3 6
7 0 0 0 6 8 0 8 0 0 0 8 6 0
221 36 f .5 1 5
0 7 0 0 0 7 0 0 7 0 0 7 7 0
220 32 f 2 4 5
5 0 5 0 5 5 5 0 5 5 5 5 5 5
684 19 f 2 4 2
0 2 0 2 0 0 0 0 0 2 2 0 0 0
493 55 f 1 0 5
5 0 0 5 0 0 0 0 7 0 0 0 0 0
221 27 m 1 1 7
7 0 0 0 0 0 0 0 5 0 0 0 5 0
684 19 f 0 .5 1
7 0 0 0 0 1 1 0 0 0 0 0 1 1
493 38 f .5 .5 5
0 8 0 0 5 0 0 0 5 0 0 0 0 0
221 26 f .5 2 1
0 1 0 0 0 1 0 0 5 5 5 1 0 0
684 18 m 1 .5 1
0 2 0 0 0 0 1 0 0 0 0 1 1 0
684 19 m 1 1 1
0 0 0 1 1 0 0 0 0 0 1 0 0 0
221 29 m .5 .5 5
0 0 0 0 0 5 5 0 0 0 0 0 5 5
683 18 f 2 4 8
0 0 0 0 8 0 0 0 8 8 8 0 0 0
966 23 f 1 2 1
1 5 5 5 1 0 0 0 0 1 0 0 1 0
493 25 f 3 5 7
7 0 0 0 7 2 0 0 7 0 2 7 7 0
683 18 f .5 .5 2
1 0 0 0 0 0 5 0 0 1 0 0 0 1
382 21 f 3 1 8
0 8 0 0 5 8 8 0 0 8 8 0 0 0
683 18 f 4 6 2
2 0 0 0 2 2 2 0 2 0 2 2 2 0
684 19 m .5 2 1
0 0 0 0 1 1 0 0 0 1 1 1 1 5
684 19 m 1.5 3.5 2
```

```
2 0 0 0 2 0 0 0 0 0 2 5 0 0
221 23 f 1 5 1
7 5 1 5 1 3 1 7 5 1 5 1 3 1
684 18 f 2 3 1
2 0 0 1 1 1 1 7 2 0 1 1 1 1
683 19 f 3 5 2
2 0 0 2 0 6 1 0 1 1 2 2 6 1
683 19 f 3 5 1
2 0 0 2 0 6 1 0 1 1 2 0 2 1
221 35 m 3 5 5
7 5 0 1 7 0 0 5 5 5 0 0 0 0
221 43 f 1 4 5
1 0 0 0 5 0 0 5 5 0 0 0 0 0
493 32 f 2 1 6
0 0 0 6 0 0 0 0 0 0 0 0 4 0
221 24 f 4 5 2
2 0 5 0 0 2 4 4 4 5 0 0 2 2
684 19 f 2 3 2
0 5 5 2 5 0 1 0 5 5 2 2 2 2
221 19 f 3 3 8
0 1 1 8 8 8 4 0 5 4 1 8 8 4
221 29 m 1 1 5
5 5 5 5 5 5 5 5 5 5 5 5 5 5
221 21 m 1 1 1
1 0 0 0 0 0 5 1 0 0 0 0 0 5
683 20 f 1 2 2
0 0 0 0 2 0 0 0 2 0 0 0 0 0
493 54 f 1 1 5
7 0 0 5 0 0 0 0 0 0 5 0 0 0
493 45 m 4 6 5
7 0 0 0 7 5 0 0 5 5 5 5 5 5
850 44 m 2.5 1.5 7
7 0 7 0 4 7 5 0 5 4 3 0 0 4
220 33 m 5 3 5
1 5 0 5 1 0 0 0 0 0 0 0 5 5
684 20 f 1.5 3 1
1 0 0 0 1 0 1 0 1 0 0 1 1 0
966 63 m 3 5 3
5 4 7 5 4 5 0 5 0 0 5 5 4 0
683 21 f 4 6 1
0 1 0 1 1 1 1 0 1 1 1 1 1 1
493 23 f 5 2 5
7 5 0 4 0 0 0 0 1 1 1 1 1 0
493 32 f 8 8 5
7 5 0 0 7 0 5 5 5 0 0 7 5 5
942 33 f 7 2 5
0 5 5 4 7 0 0 0 0 0 0 7 8 0
493 34 f .5 1 5
5 0 0 0 5 0 0 0 0 0 6 0 0 0
382 40 f 2 2 5
5 0 0 0 5 0 0 5 0 0 5 0 0 0
362 27 f 0 3 8
0 0 0 0 0 0 0 0 0 0 0 0 8 0
542 36 f 3 3 7
```

```
7 0 0 0 7 1 0 0 0 7 1 1 0 0
966 39 f 3 6 5
7 0 0 0 7 5 0 0 7 0 5 0 5 0
849 32 m 1 .5 7
7 0 0 0 5 0 0 0 7 4 4 5 7 0
677 52 f 3 2 3
7 0 0 0 0 7 0 0 0 7 0 0 3 0
222 25 m 2 4 1
1 0 0 0 1 0 0 0 1 0 1 0 0 0
732 42 f 3 2 7
7 0 0 0 1 7 5 5 7 0 0 3 4 0
467 26 f 4 4 1
7 0 1 0 7 1 0 0 7 7 4 7 0 0
467 38 m 2.5 0 1
1 0 0 0 1 0 0 0 0 0 0 0 0 0
382 37 f 1.5 .5 7
7 0 0 0 7 0 0 0 3 0 0 0 3 0
856 45 f 3 3 7
7 0 0 0 7 5 0 0 7 7 4 0 0 0
677 33 m 3 2 7
7 0 0 4 7 0 0 0 7 0 0 0 0 0
490 27 f .5 1 2
2 0 0 0 2 0 0 0 2 0 2 0 0 0
362 27 f 1.5 2 2
2 0 0 0 1 0 4 0 1 0 0 0 4 4
783 25 f 2 1 1
1 0 0 0 1 7 0 0 0 0 1 1 1 0
546 30 f 8 3 1
1 1 1 1 1 0 0 1 0 5 5 0 0 0
677 30 f 2 0 1
1 0 0 0 0 1 0 0 0 0 0 0 0 1
221 35 f 2 2 1
1 0 0 0 1 0 1 0 1 1 1 0 0 0
966 32 f 6 1 7
7 1 1 1 7 4 0 1 7 1 8 8 4 0
222 28 f 1 5 4
7 0 0 0 4 0 0 4 4 4 4 0 0 0
467 29 f 5 3 4
4 5 5 5 1 4 4 5 1 1 1 1 4 4
467 32 m 3 4 1
1 0 1 0 4 0 0 0 4 0 0 0 1 0
966 30 m 1.5 1 7
7 0 0 0 7 5 0 7 0 0 0 0 5 0
967 38 m 14 4 7
7 7 7 7 7 0 4 8 0 0 0 0 4 0
490 28 m 8 1 1
7 1 1 1 1 0 0 7 0 0 8 0 0 0
833 30 f .5 1 6
6 0 0 0 6 0 0 0 0 6 0 0 6 0
851 40 m 1 0 7
7 5 5 5 7 0 0 0 0 0 0 0 0 0
859 27 f 2 5 2
6 0 0 0 2 0 0 0 0 0 0 2 2 2
851 22 f 3 5 2
```

```
7 0 2 0 2 2 0 0 2 0 8 0 2 0
967 38 f 1 1.5 7
7 0 0 0 7 5 0 7 4 0 0 7 5 0
856 34 f 1.5 1 1
0 1 0 0 0 1 0 0 4 0 0 0 0 0
222 33 m .1 .1 7
7 0 0 0 7 0 0 0 0 0 7 0 0 0
856 22 m .50 .25 1
0 1 0 0 1 0 0 0 0 0 0 0 0 0
677 30 f 2 2 4
1 0 4 0 4 0 0 0 4 0 0 0 0 0
859 25 m 2 3 7
0 0 0 0 0 7 0 0 7 0 2 0 0 1
833 35 m 2 6 7
7 0 0 0 7 1 1 0 4 7 4 7 1 1
677 35 m 10 4 1
1 1 1 1 1 8 6 8 1 0 0 8 8 8
848 29 f 5 3 8
8 0 0 0 8 8 0 0 0 8 8 8 0 0
688 26 m 3 1 1
1 1 7 1 1 7 0 0 0 8 8 0 0 0
490 41 m 2 2 5
5 0 0 0 0 0 5 5 0 0 0 0 0 5
493 35 m 4 4 7
7 5 0 5 7 0 0 7 7 7 7 0 0 0
677 27 m 15 11 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
848 27 f 3 5 1
1 1 0 0 1 1 0 0 1 1 1 1 0 0
362 30 f 1 0 1
1 0 0 0 7 5 0 0 0 0 0 0 0 0
783 29 f 1 1 4
4 0 0 0 4 0 0 0 4 0 0 0 4 0
467 39 f .5 2 4
7 0 4 0 4 4 0 0 4 4 4 4 4 4
677 27 m 2 2 7
7 0 0 0 7 0 0 7 7 0 0 7 0 0
221 23 f 2.5 1 1
1 0 0 0 1 0 0 0 0 0 0 0 0 0
677 29 f 1 1 7
0 0 0 0 7 0 0 0 7 0 0 0 0 0
783 32 m 1 2 5
4 5 5 5 4 2 0 0 0 0 3 2 2 0
833 25 f 1 0 1
1 1 0 0 0 0 0 0 0 0 0 0 0 0
859 24 f 7 3 7
1 0 0 0 1 0 0 0 0 1 0 0 1 0
677 29 m 2 2 8
0 8 8 0 8 0 0 0 8 8 8 0 0 0
688 31 m 8 2 5
7 5 5 5 5 7 0 0 7 7 0 0 0 0
856 31 m 9 4 1
1 1 1 1 1 0 0 0 0 0 0 0 1 0
856 44 f 1 0 6
```

```
6 0 0 0 6 0 0 0 0 0 0 0 0 0
677 37 f 3 3 1
0 0 1 0 0 0 0 0 4 4 0 0 0 0
859 27 m 2 .5 2
2 2 2 2 2 2 2 2 0 0 0 0 0 2
781 30 f 10 4 2
2 0 0 0 2 0 2 0 0 0 0 0 0 2
362 27 m 12 4 3
3 1 1 1 1 3 3 3 0 0 0 0 3 0
362 33 f 2 4 1
1 0 0 0 7 0 0 7 1 1 1 1 1 0
222 26 f 8 1 1
1 1 1 1 0 0 0 1 0 0 0 0 0 0
779 37 f 6 3 1
1 1 1 1 1 0 0 1 1 0 0 0 1 0
467 32 f 1 1 2
2 0 0 0 0 0 0 0 2 0 0 2 0 0
859 23 m 1 1 1
1 0 0 0 1 1 0 1 0 0 0 0 1 1
781 33 f 1 .5 6
6 0 0 0 6 0 0 0 0 0 0 0 0 0
779 28 m 5 2 1
1 1 1 1 1 0 0 0 0 7 7 1 1 0
677 28 m 3 1 5
7 5 5 5 5 6 0 0 6 6 6 6 6 0
677 25 f 9 2 5
1 5 5 5 5 1 1 0 1 1 1 1 1 1
848 30 f 6 2 8
8 0 0 0 2 7 0 0 0 0 2 0 2 0
546 36 f 4 6 4
7 0 0 0 4 4 0 5 5 5 5 2 4 4
222 30 f 2 3 2
2 2 0 0 2 0 0 0 2 0 2 2 0 0
383 32 m 4 1 2
2 0 0 0 2 0 0 2 0 0 0 0 0 0
851 43 f 8 1 6
4 6 0 6 4 0 0 0 0 0 0 0 0 0
222 27 f 1 3 1
1 1 0 1 1 1 0 0 1 0 0 0 4 0
833 22 f 1.5 2 1
1 0 0 0 1 1 0 0 1 1 1 0 0 0
467 29 f 2 1 8
8 0 8 0 8 0 0 0 0 0 8 0 0 0
856 28 f 2 3 1
1 0 0 0 1 0 0 0 1 0 0 1 0 0
580 31 f 2.5 2.5 6
6 6 6 6 6 6 6 6 1 1 1 1 6 6
688 39 f 8 8 3
3 3 3 3 3 3 3 3 3 3 3 3 3 3
677 37 f 1.5 .5 1
6 1 1 1 6 6 0 0 1 1 6 6 6 0
859 38 m 3 6 3
7 0 0 0 7 3 0 0 3 0 3 0 0 0
677 25 f 7 1 1
```

```
0 1 1 1 2 0 0 0 1 2 1 1 1 0
848 36 f 7 1 1
0 1 0 1 1 0 0 0 0 0 0 1 1 0
781 31 f 2 4 1
1 0 0 0 1 1 0 1 1 1 1 1 0 0
781 40 f 2 2 8
8 0 0 8 8 0 0 0 0 0 8 8 0 0
677 25 f 3 5 1
1 6 1 6 6 3 0 0 2 2 1 1 1 1
779 33 f 3 2 1
1 0 1 0 0 0 1 0 1 0 0 0 1 0
677 25 m 7 1.5 1
1 1 0 1 1 0 0 0 0 0 1 0 0 0
362 35 f .5 0 1
1 0 0 0 1 0 0 0 0 0 0 0 0 0
677 41 f 6 2 7
7 7 0 7 7 0 0 0 0 0 8 0 0 0
677 24 m 5 1 5
1 5 0 5 0 0 0 0 1 0 0 0 0 0
833 29 f .5 0 6
6 0 0 0 6 0 0 0 0 0 0 0 0 0
362 30 f 1 1 1
1 0 0 0 1 0 0 0 1 0 0 0 0 0
850 26 f 6 12 6
6 0 0 0 2 2 2 6 6 6 0 0 6 6
467 25 f 2 3 1
1 0 0 6 1 1 0 0 0 0 1 1 1 1
967 29 f 1 2 7
7 0 0 0 7 0 0 7 7 0 0 0 0 0
833 31 f 1 1 7
7 0 7 0 7 3 0 0 3 3 0 0 0 0
859 40 f 7 1 5
1 5 0 5 5 1 0 0 1 0 0 0 0 0
848 31 m 1 2 1
1 0 0 0 1 1 0 0 4 4 1 4 0 0
222 32 f 2 3 3
3 0 0 0 0 7 0 0 3 0 8 0 0 0
783 33 f 2 0 4
7 0 0 0 7 0 0 0 4 0 4 0 0 0
856 28 f 8 4 2
0 2 0 2 2 0 0 0 2 0 2 0 4 0
781 30 f 3 5 1
1 1 1 1 1 1 0 0 1 1 1 1 1 0
850 25 f 6 3 1
7 5 0 5 7 1 0 0 7 0 1 0 1 0
580 33 f 2.5 4 2
2 0 0 0 2 0 0 0 0 0 8 8 0 0
677 38 f 3 3 1
1 0 0 0 1 0 1 1 1 0 1 0 0 4
677 26 f 2 2 1
1 0 1 0 1 0 0 0 1 1 1 0 0 0
467 52 f 3 2 2
2 6 6 6 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 7 0 0 0 0
490 35 f 1 1 7
7 0 0 0 7 0 0 0 0 0 0 0 8 0
222 27 f 3 2 3
8 0 0 0 3 8 0 3 3 0 0 0 0 0
382 36 m 3 2 4
7 0 5 4 7 4 4 0 7 7 4 7 0 4
859 37 f 1 1 2
7 0 0 0 0 2 0 2 2 0 0 0 0 2
856 29 f 3 1 1
1 0 0 0 1 1 1 1 0 0 1 1 0 1
542 32 m 3 3 7
7 0 0 0 0 7 7 7 0 0 0 0 7 7
783 31 m 1 1 1
1 0 0 0 1 0 0 0 1 1 1 0 0 0
833 35 m 1 1 1
```

```
5 4 1 5 1 0 0 1 1 0 0 0 0 0
782 38 m 30 8 5
7 5 5 5 5 0 0 4 4 4 4 4 0 0
222 33 m 3 3 1
1 1 1 1 1 1 1 1 4 1 1 1 1 1
467 24 f 2 4 1
0 0 1 0 1 0 0 0 1 1 1 0 0 0
467 34 f 1 1 1
1 0 0 0 1 0 0 1 1 0 0 0 0 0
781 53 f 2 1 5
5 0 0 0 5 5 0 0 0 0 5 5 5 0
222 30 m 2 5 3
6 3 3 3 6 0 0 0 3 3 3 3 0 0
688 26 f 2 2 1
1 0 0 0 1 0 0 0 1 0 1 1 0 0
222 29 m 8 5 1
1 6 0 6 1 0 0 1 1 1 1 0 0 0
783 33 m 1 2 7
7 0 0 0 7 0 0 0 7 0 0 0 7 0
781 39 m 1.5 2.5 2
2 0 2 0 2 0 0 0 2 2 2 0 0 0
850 22 f 2 1 1
1 0 0 0 1 1 1 0 5 0 0 1 0 0
493 36 f 1 0 5
0 0 0 0 7 0 0 0 0 0 0 0 0 0
967 46 f 2 4 7
7 5 0 5 7 0 0 0 4 7 4 0 0 0
856 41 m 2 2 4
7 4 0 0 7 4 0 4 0 0 0 7 0 0
546 25 m 5 5 8
8 8 0 0 0 0 0 0 0 0 0 0 0 0
222 27 f 4 4 3
2 2 2 3 7 7 0 2 2 2 3 3 3 0
688 23 m 9 3 3
3 3 3 3 3 7 0 0 3 0 0 0 0 0
849 26 m .5 .5 8
8 0 0 0 8 0 0 0 0 8 0 0 0 0
783 29 f 3 3 1
1 0 0 0 4 0 0 4 1 0 1 0 0 0
856 34 f 1.5 2 1
7 0 0 0 7 0 0 7 4 0 0 7 0 0
966 33 m 3 5 4
7 0 0 0 7 4 5 0 7 0 0 7 4 4
493 34 f 2 5 1
1 0 0 0 1 0 0 0 7 0 1 1 8 0
467 29 m 2 4 2
2 0 0 0 2 0 0 2 2 2 2 2 2 2
677 28 f 1 4 1
1 1 1 1 1 0 0 0 1 0 1 0 0 0
781 27 m 2 2 1
1 0 1 0 4 2 4 0 2 2 1 0 1 4
467 24 m 4 4 1
7 1 0 1 1 1 0 7 1 0 0 0 0 0
859 26 m 5 5 1
```

```
1 1 1 1 1 1 1 1 1 1 1 1 1 1
848 27 m 7 2 5
7 5 0 5 4 5 0 0 0 7 4 4 0 4
677 25 f 1 2 8
8 0 0 0 0 5 0 0 8 0 0 0 2 0
222 26 f 3.5 0 2
2 0 0 0 2 0 0 0 0 0 0 0 0 0
833 32 m 1 2 1
1 0 0 0 1 0 0 0 5 0 1 0 0 0
781 28 m 2 .5 7
7 0 0 0 7 0 0 0 4 0 0 0 0 0
783 28 f 1 1 1
1 0 0 0 1 0 0 0 0 0 1 1 0 0
222 28 f 5 5 2
2 6 6 2 2 0 0 0 2 2 0 0 2 2
851 33 m 4 5 3
1 0 0 0 7 3 0 3 3 3 3 3 7 5
859 39 m 2 1 1
1 0 0 0 1 0 0 0 0 0 0 1 0 0
848 45 m 2 2 7
7 0 0 0 7 0 0 0 7 0 0 0 0 0
467 37 m 2 2 7
7 0 0 0 0 7 0 0 0 7 0 0 7 0
859 32 m .25 .25 1
1 0 0 0 0 0 0 0 1 0 0 0 0 0
```

APPENDIX 4

Recommended Reading

Recommended Reading

Here is the recommended reading list for this title:

- *The Little SAS Book: A Primer, Third Edition*
- *Output Delivery System: The Basics*
- *PROC TABULATE by Example*
- *SAS Guide to Report Writing: Examples*
- *SAS Language Reference: Concepts*
- *SAS Language Reference: Dictionary*
- *SAS Output Delivery System: User's Guide*
- *SAS Programming by Example*
- *SAS SQL Procedure User's Guide*
- *Step-by-Step Programming with Base SAS Software*

For a complete list of SAS publications, see the current *SAS Publishing Catalog*. To order the most current publications or to receive a free copy of the catalog, contact a SAS representative at

SAS Publishing Sales
SAS Campus Drive
Cary, NC 27513
Telephone: (800) 727-3228*
Fax: (919) 677-8166
E-mail: **sasbook@sas.com**
Web address: **support.sas.com/pubs**
* For other SAS Institute business, call (919) 677-8000.

Customers outside the United States should contact their local SAS office.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

V

W

X

Your Turn

If you have comments or suggestions about *Base SAS 9.1 Procedures Guide*, please send them to us on a photocopy of this page, or send us electronic mail.

For comments about this book, please return the photocopy to

SAS Publishing
SAS Campus Drive
Cary, NC 27513
email: `yourturn@sas.com`

Send suggestions about the software, please return the photocopy to

SAS Institute Inc.
Technical Support Division
SAS Campus Drive
Cary, NC 27513
email: `suggest@sas.com`

SAS Publishing

Base SAS® 9.1 Procedures Guide

Volume 3

CORR, FREQ, and UNIVARIATE Procedures

The Power to Know®

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2004. *Base SAS® 9.1 Procedures Guide*. Cary, NC: SAS Institute Inc.

Base SAS® 9.1 Procedures Guide

ISBN 1-59047-204-7

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

1st printing, January 2004

Contents

Chapter 1
The CORR Procedure

Chapter Contents

Chapter 1
The CORR Procedure

Overview

The CORR procedure computes Pearson correlation coefficients, three nonparametric measures of association, and the probabilities associated with these statistics. The correlation statistics include

- Pearson product-moment correlation
- Spearman rank-order correlation
- Kendall's tau-b coefficient
- Hoeffding's measure of dependence, D
- Pearson, Spearman, and Kendall partial correlation

Pearson product-moment correlation is a parametric measure of a linear relationship between two variables. For nonparametric measures of association, Spearman rank-order correlation uses the ranks of the data values and Kendall's tau-b uses the number of concordances and discordances in paired observations. Hoeffding's measure of dependence is another nonparametric measure of association that detects more general departures from independence. A partial correlation provides a measure of the correlation between two variables after controlling the effects of other variables.

With only one set of analysis variables specified, the default correlation analysis includes descriptive statistics for each analysis variable and Pearson correlation statistics for these variables. You can also compute Cronbach's coefficient alpha for estimating reliability.

With two sets of analysis variables specified, the default correlation analysis includes descriptive statistics for each analysis variable and Pearson correlation statistics between these two sets of variables.

You can save the correlation statistics in a SAS data set for use with other statistical and reporting procedures.

For a Pearson or Spearman correlation, the Fisher's z transformation can be used to derive its confidence limits and a p-value under a specified null hypothesis $H_0\colon \rho = \rho_0$. Either a one-sided or a two-sided alternative is used for these statistics.

Experimental ODS graphics are now available with the CORR procedure, including scatter plots and a scatter plot matrix of the analysis variables. For more information, see the "ODS Graphics" section on page 31.

Getting Started

The following statements create the data set Fitness, which has been altered to contain some missing values:

```
*----------------- Data on Physical Fitness -----------------*
| These measurements were made on men involved in a physical |
| fitness course at N.C. State University.                   |
| The variables are Age (years), Weight (kg),                |
| Runtime (time to run 1.5 miles in minutes), and            |
| Oxygen (oxygen intake, ml per kg body weight per minute)   |
| Certain values were changed to missing for the analysis.   |
*------------------------------------------------------------*;
data Fitness;
   input Age Weight Oxygen RunTime @@;
   datalines;
44 89.47 44.609 11.37    40 75.07 45.313 10.07
44 85.84 54.297  8.65    42 68.15 59.571  8.17
38 89.02 49.874   .      47 77.45 44.811 11.63
40 75.98 45.681 11.95    43 81.19 49.091 10.85
44 81.42 39.442 13.08    38 81.87 60.055  8.63
44 73.03 50.541 10.13    45 87.66 37.388 14.03
45 66.45 44.754 11.12    47 79.15 47.273 10.60
54 83.12 51.855 10.33    49 81.42 49.156  8.95
51 69.63 40.836 10.95    51 77.91 46.672 10.00
48 91.63 46.774 10.25    49 73.37   .    10.08
57 73.37 39.407 12.63    54 79.38 46.080 11.17
52 76.32 45.441  9.63    50 70.87 54.625  8.92
51 67.25 45.118 11.08    54 91.63 39.203 12.88
51 73.71 45.790 10.47    57 59.08 50.545  9.93
49 76.32   .      .      48 61.24 47.920 11.50
52 82.78 47.467 10.50
;
```

The following statements invoke the CORR procedure and request a correlation analysis:

```
ods html;
ods graphics on;

proc corr data=Fitness plots;
run;

ods graphics off;
ods html close;
```

This graphical display is requested by specifying the experimental ODS GRAPHICS statement and the experimental PLOTS option. For general information about ODS graphics, refer to Chapter 15, “Statistical Graphics Using ODS” (*SAS/STAT User’s Guide*). For specific information about the graphics available in the CORR procedure, see the section “ODS Graphics” on page 31.

The CORR Procedure

4 Variables: Age Weight Oxygen RunTime

Simple Statistics

Variable	N	Mean	Std Dev	Sum	Minimum	Maximum
Age	31	47.67742	5.21144	1478	38.00000	57.00000
Weight	31	77.44452	8.32857	2401	59.08000	91.63000
Oxygen	29	47.22721	5.47718	1370	37.38800	60.05500
RunTime	29	10.67414	1.39194	309.55000	8.17000	14.03000

Figure 1.1. Univariate Statistics

By default, all numeric variables not listed in other statements are used in the analysis. Observations with nonmissing values for each variable are used to derive the univariate statistics for that variable.

Pearson Correlation Coefficients
Prob > |r| under H0: Rho=0
Number of Observations

	Age	Weight	Oxygen	RunTime
Age	1.00000	-0.23354	-0.31474	0.14478
		0.2061	0.0963	0.4536
	31	31	29	29
Weight	-0.23354	1.00000	-0.15358	0.20072
	0.2061		0.4264	0.2965
	31	31	29	29
Oxygen	-0.31474	-0.15358	1.00000	-0.86843
	0.0963	0.4264		<.0001
	29	29	29	28
RunTime	0.14478	0.20072	-0.86843	1.00000
	0.4536	0.2965	<.0001	
	29	29	28	29

Figure 1.2. Pearson Correlation Coefficients

By default, Pearson correlation statistics are computed from observations with nonmissing values for each pair of analysis variables. With missing values in the analysis, the "Pearson Correlation Coefficients" table shown in Figure 1.2 displays the correlation, the p-value under the null hypothesis of zero correlation, and the number of nonmissing observations for each pair of variables.

The table displays a correlation of -0.86843 between Runtime and Oxygen, which is significant with a p-value less than 0.0001. That is, there exists an inverse linear relationship between these two variables. As Runtime (time to run 1.5 miles in minutes) increases, Oxygen (oxygen intake, ml per kg body weight per minute) decreases.

The experimental PLOTS option displays a symmetric matrix plot for the analysis variables. This inverse linear relationship between these two variables, Oxygen and Runtime, is also shown in Figure 1.3.

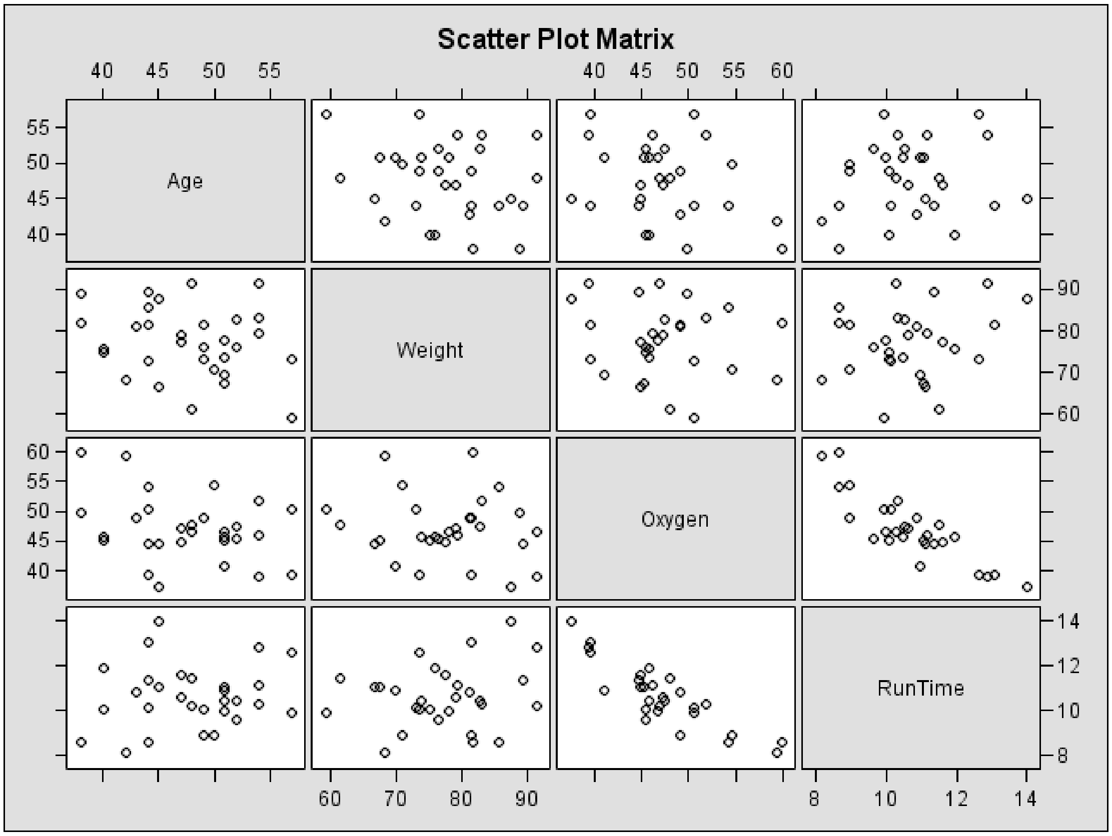


Figure 1.3. Symmetric Matrix Plot (Experimental)

Syntax

The following statements are available in PROC CORR.

> **PROC CORR** < *options* > ;
>   **BY** *variables* ;
>   **FREQ** *variable* ;
>   **PARTIAL** *variables* ;
>   **VAR** *variables* ;
>   **WEIGHT** *variable* ;
>   **WITH** *variables* ;

The BY statement specifies groups in which separate correlation analyses are performed.

The FREQ statement specifies the variable that represents the frequency of occurrence for other values in the observation.

The PARTIAL statement identifies controlling variables to compute Pearson, Spearman, or Kendall partial-correlation coefficients.

The VAR statement lists the numeric variables to be analyzed and their order in the correlation matrix. If you omit the VAR statement, all numeric variables not listed in other statements are used.

The WEIGHT statement identifies the variable whose values weight each observation to compute Pearson product-moment correlation.

The WITH statement lists the numeric variables with which correlations are to be computed.

The PROC CORR statement is the only required statement for the CORR procedure. The rest of this section provides detailed syntax information for each of these statements, beginning with the PROC CORR statement. The remaining statements are in alphabetical order.

PROC CORR Statement

PROC CORR < *options* > **;**

The following table summarizes the options available in the PROC CORR statement.

Table 1.1. Summary of PROC CORR Options

Tasks	Options
Specify data sets	
Input data set	DATA=
Output data set with Hoeffding's D statistics	OUTH=
Output data set with Kendall correlation statistics	OUTK=
Output data set with Pearson correlation statistics	OUTP=
Output data set with Spearman correlation statistics	OUTS=
Control statistical analysis	
Exclude observations with nonpositive weight values from the analysis	EXCLNPWGT
Exclude observations with missing analysis values from the analysis	NOMISS
Request Hoeffding's measure of dependence, D	HOEFFDING
Request Kendall's tau-b	KENDALL
Request Pearson product-moment correlation	PEARSON
Request Spearman rank-order correlation	SPEARMAN
Request Pearson correlation statistics using Fisher's z transformation	FISHER PEARSON
Request Spearman rank-order correlation statistics using Fisher's z transformation	FISHER SPEARMAN
Control Pearson correlation statistics	
Compute Cronbach's coefficient alpha	ALPHA
Compute covariances	COV
Compute corrected sums of squares and crossproducts	CSSCP

Table 1.1. (continued)

Tasks	Options
Compute correlation statistics based on Fisher's z transformation	FISHER
Exclude missing values	NOMISS
Specify singularity criterion	SINGULAR=
Compute sums of squares and crossproducts	SSCP
Specify the divisor for variance calculations	VARDEF=
Control printed output	
Display a specified number of ordered correlation coefficients	BEST=
Suppress Pearson correlations	NOCORR
Suppress all printed output	NOPRINT
Suppress p-values	NOPROB
Suppress descriptive statistics	NOSIMPLE
Display ordered correlation coefficients	RANK

The following options (listed in alphabetical order) can be used in the PROC CORR statement:

ALPHA

calculates and prints Cronbach's coefficient alpha. PROC CORR computes separate coefficients using raw and standardized values (scaling the variables to a unit variance of 1). For each VAR statement variable, PROC CORR computes the correlation between the variable and the total of the remaining variables. It also computes Cronbach's coefficient alpha using only the remaining variables.

If a WITH statement is specified, the ALPHA option is invalid. When you specify the ALPHA option, the Pearson correlations will also be displayed. If you specify the OUTP= option, the output data set also contains observations with Cronbach's coefficient alpha. If you use the PARTIAL statement, PROC CORR calculates Cronbach's coefficient alpha for partialled variables. See the section "Partial Correlation" on page 18.

BEST=*n*

prints the n highest correlation coefficients for each variable, $n \geq 1$. Correlations are ordered from highest to lowest in absolute value. Otherwise, PROC CORR prints correlations in a rectangular table using the variable names as row and column labels.

If you specify the HOEFFDING option, PROC CORR displays the D statistics in order from highest to lowest.

COV

displays the variance and covariance matrix. When you specify the COV option, the Pearson correlations will also be displayed. If you specify the OUTP= option, the output data set also contains the covariance matrix with the corresponding _TYPE_ variable value 'COV.' If you use the PARTIAL statement, PROC CORR computes a partial covariance matrix.

CSSCP

displays a table of the corrected sums of squares and crossproducts. When you specify the CSSCP option, the Pearson correlations will also be displayed. If you specify the OUTP= option, the output data set also contains a CSSCP matrix with the corresponding _TYPE_ variable value 'CSSCP.' If you use a PARTIAL statement, PROC CORR prints both an unpartial and a partial CSSCP matrix, and the output data set contains a partial CSSCP matrix.

DATA=*SAS-data-set*

names the SAS data set to be analyzed by PROC CORR. By default, the procedure uses the most recently created SAS data set.

EXCLNPWGT

excludes observations with nonpositive weight values from the analysis. By default, PROC CORR treats observations with negative weights like those with zero weights and counts them in the total number of observations.

FISHER < *(fisher-options)* >

requests confidence limits and p-values under a specified null hypothesis, $H_0\colon \rho = \rho_0$, for correlation coefficients using Fisher's z transformation. These correlations include the Pearson correlations and Spearman correlations.

The following *fisher-options* are available:

ALPHA= α

specifies the level of the confidence limits for the correlation, $100(1-\alpha)\%$. The value of the ALPHA= option must be between 0 and 1, and the default is ALPHA=0.05.

BIASADJ= YES | NO

specifies whether or not the bias adjustment is used in constructing confidence limits. The BIASADJ=YES option also produces a new correlation estimate using the bias adjustment. By default, BIASADJ=YES.

RHO0= ρ_0

specifies the value ρ_0 in the null hypothesis $H_0\colon \rho = \rho_0$, where $-1 < \rho_0 < 1$. By default, RHO0=0.

TYPE= LOWER | UPPER | TWOSIDED

specifies the type of confidence limits. The TYPE=LOWER option requests a lower confidence limit from the lower alternative $H_1\colon \rho < \rho_0$, the TYPE=UPPER option requests an upper confidence limit from the upper alternative $H_1\colon \rho > \rho_0$, and the default TYPE=TWOSIDED option requests two-sided confidence limits from the two-sided alternative $H_1\colon \rho \neq \rho_0$.

HOEFFDING

requests a table of Hoeffding's D statistics. This D statistic is 30 times larger than the usual definition and scales the range between -0.5 and 1 so that large positive values indicate dependence. The HOEFFDING option is invalid if a WEIGHT or PARTIAL statement is used.

KENDALL
requests a table of Kendall's tau-b coefficients based on the number of concordant and discordant pairs of observations. Kendall's tau-b ranges from −1 to 1.

The KENDALL option is invalid if a WEIGHT statement is used. If you use a PARTIAL statement, probability values for Kendall's partial tau-b are not available.

NOCORR
suppresses displaying of Pearson correlations. If you specify the OUTP= option, the data set type remains CORR. To change the data set type to COV, CSSCP, or SSCP, use the TYPE= data set option.

NOMISS
excludes observations with missing values from the analysis. Otherwise, PROC CORR computes correlation statistics using all of the nonmissing pairs of variables. Using the NOMISS option is computationally more efficient.

NOPRINT
suppresses all displayed output. Use NOPRINT if you want to create an output data set only.

NOPROB
suppresses displaying the probabilities associated with each correlation coefficient.

NOSIMPLE
suppresses printing simple descriptive statistics for each variable. However, if you request an output data set, the output data set still contains simple descriptive statistics for the variables.

OUTH=*output-data-set*
creates an output data set containing Hoeffding's D statistics. The contents of the output data set are similar to the OUTP= data set. When you specify the OUTH= option, the Hoeffding's D statistics will be displayed, and the Pearson correlations will be displayed only if the PEARSON, ALPHA, COV, CSSCP, SSCP, or OUT= option is also specified.

OUTK=*output-data-set*
creates an output data set containing Kendall correlation statistics. The contents of the output data set are similar to those of the OUTP= data set. When you specify the OUTK= option, the Kendall correlation statistics will be displayed, and the Pearson correlations will be displayed only if the PEARSON, ALPHA, COV, CSSCP, SSCP, or OUT= option is also specified.

OUTP=*output-data-set*
OUT=*output-data-set*
creates an output data set containing Pearson correlation statistics. This data set also includes means, standard deviations, and the number of observations. The value of the _TYPE_ variable is 'CORR.' When you specify the OUTP= option, the Pearson correlations will also be displayed. If you specify the ALPHA option, the output data set also contains six observations with Cronbach's coefficient alpha.

OUTS=*SAS-data-set*

creates an output data set containing Spearman correlation coefficients. The contents of the output data set are similar to the OUTP= data set. When you specify the OUTS= option, the Spearman correlation coefficients will be displayed, and the Pearson correlations will be displayed only if the PEARSON, ALPHA, COV, CSSCP, SSCP, or OUT= option is also specified.

PEARSON

requests a table of Pearson product-moment correlations. If you do not specify the HOEFFDING, KENDALL, SPEARMAN, OUTH=, OUTK=, or OUTS= option, the CORR procedure produces Pearson product-moment correlations by default. Otherwise, you must specify the PEARSON, ALPHA, COV, CSSCP, SSCP, or OUT= option for Pearson correlations. The correlations range from -1 to 1.

RANK

displays the ordered correlation coefficients for each variable. Correlations are ordered from highest to lowest in absolute value. If you specify the HOEFFDING option, the D statistics are displayed in order from highest to lowest.

SINGULAR=*p*

specifies the criterion for determining the singularity of a variable if you use a PARTIAL statement. A variable is considered singular if its corresponding diagonal element after Cholesky decomposition has a value less than *p* times the original unpartialled value of that variable. The default value is 1E$-$8. The range of ρ is between 0 and 1.

SPEARMAN

requests a table of Spearman correlation coefficients based on the ranks of the variables. The correlations range from -1 to 1. If you specify a WEIGHT statement, the SPEARMAN option is invalid.

SSCP

displays a table the sums of squares and crossproducts. When you specify the SSCP option, the Pearson correlations will also be displayed. If you specify the OUTP= option, the output data set contains a SSCP matrix and the corresponding _TYPE_ variable value is 'SSCP.' If you use a PARTIAL statement, the unpartial SSCP matrix is displayed, and the output data set does not contain an SSCP matrix.

VARDEF=*d*

specifies the variance divisor in the calculation of variances and covariances. The following table shows the possible values for the value d and associated divisors, where *k* is the number of PARTIAL statement variables. The default is VARDEF=DF.

Table 1.2. Possible Values for VARDEF=

Value	Divisor	Formula
DF	degrees of freedom	$n-k-1$
N	number of observations	n
WDF	sum of weights minus one	$(\Sigma w_i)-k-1$
WEIGHT|WGT	sum of weights	Σw_i

The variance is computed as

$$\frac{1}{d}\sum_i (x_i - \bar{x})^2$$

where $\bar{x}$ is the sample mean.

If a WEIGHT statement is used, the variance is computed as

$$\frac{1}{d}\sum_i w_i(x_i - \bar{x}_w)^2$$

where w_i is the weight for the ith observation and $\bar{x}_w$ is the weighted mean.

If you use the WEIGHT statement and VARDEF=DF, the variance is an estimate of s^2, where the variance of the ith observation is $V(x_i) = s^2/w_i$. This yields an estimate of the variance of an observation with unit weight.

If you use the WEIGHT statement and VARDEF=WGT, the computed variance is asymptotically an estimate of $s^2/\bar{w}$, where $\bar{w}$ is the average weight (for large *n*). This yields an asymptotic estimate of the variance of an observation with average weight.

BY Statement

BY *variables* **;**

You can specify a BY statement with PROC CORR to obtain separate analyses on observations in groups defined by the BY variables. If a BY statement appears, the procedure expects the input data set to be sorted in order of the BY variables.

If your input data set is not sorted in ascending order, use one of the following alternatives:

- Sort the data using the SORT procedure with a similar BY statement.
- Specify the BY statement option NOTSORTED or DESCENDING in the BY statement for the CORR procedure. The NOTSORTED option does not mean that the data are unsorted but rather that the data are arranged in groups (according to values of the BY variables) and that these groups are not necessarily in alphabetical or increasing numeric order.
- Create an index on the BY variables using the DATASETS procedure.

For more information on the BY statement, refer to the discussion in *SAS Language Reference: Concepts*. For more information on the DATASETS procedure, refer to the discussion in the *SAS Procedures Guide*.

FREQ Statement

FREQ *variable* **;**

The FREQ statement lists 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 the FREQ variable. If n is not an integer, SAS truncates it. If n is less than 1 or is missing, the observation is excluded from the analysis. The sum of the frequency variable represents the total number of observations.

The effects of the FREQ and WEIGHT statements are similar except when calculating degrees of freedom.

PARTIAL Statement

PARTIAL *variables* **;**

The PARTIAL statement lists variables to use in the calculation of partial correlation statistics. Only the Pearson partial correlation, Spearman partial rank-order correlation, and Kendall's partial tau-b can be computed. It is not valid with the HOEFFDING option. When you use the PARTIAL statement, observations with missing values are excluded.

With a PARTIAL statement, PROC CORR also displays the partial variance and standard deviation for each analysis variable if the PEARSON option is specified.

VAR Statement

VAR *variables* **;**

The VAR statement lists variables for which to compute correlation coefficients. If the VAR statement is not specified, PROC CORR computes correlations for all numeric variables not listed in other statements.

WEIGHT Statement

WEIGHT *variable* **;**

The WEIGHT statement lists weights to use in the calculation of Pearson weighted product-moment correlation. The HOEFFDING, KENDALL, and SPEARMAN options are not valid with the WEIGHT statement.

The observations with missing weights are excluded from the analysis. By default, for observations with nonpositive weights, weights are set to zero and the observations are included in the analysis. You can use the EXCLNPWGT option to exclude observations with negative or zero weights from the analysis.

Note that most SAS/STAT procedures, such as PROC GLM, exclude negative and zero weights by default. If you use the WEIGHT statement, consider which value of the VARDEF= option is appropriate. See the discussion of the VARDEF= option for more information.

WITH Statement

WITH *variables***;**

The WITH statement lists variables with which correlations of the VAR statement variables are to be computed. The WITH statement requests correlations of the form $r(X_i, Y_j)$, where $X_1, \ldots, X_m$ are analysis variables specified in the VAR statement, and $Y_1, \ldots, Y_n$ are variables specified in the WITH statement. The correlation matrix has a rectangular structure of the form

$$\begin{bmatrix} r(Y_1, X_1) & \cdots & r(Y_1, X_m) \\ \vdots & \ddots & \vdots \\ r(Y_n, X_1) & \cdots & r(Y_n, X_m) \end{bmatrix}$$

For example, the statements

```
proc corr;
   var x1 x2;
   with y1 y2 y3;
run;
```

produce correlations for the following combinations:

$$\begin{bmatrix} r(Y1, X1) & r(Y1, X2) \\ r(Y2, X1) & r(Y2, X2) \\ r(Y3, X1) & r(Y3, X2) \end{bmatrix}$$

Details

Pearson Product-Moment Correlation

The Pearson product-moment correlation is a parametric measure of association for two variables. It measures both the strength and direction of a linear relationship. If one variable X is an exact linear function of another variable Y, a positive relationship exists if the correlation is 1 and a negative relationship exists if the correlation is -1. If there is no linear predictability between the two variables, the correlation is 0. If the two variables are normal with a correlation 0, the two variables are independent. However, correlation does not imply causality because, in some cases, an underlying causal relationship may not exist.

The following scatter plot matrix displays the relationship between two numeric random variables under various situations.

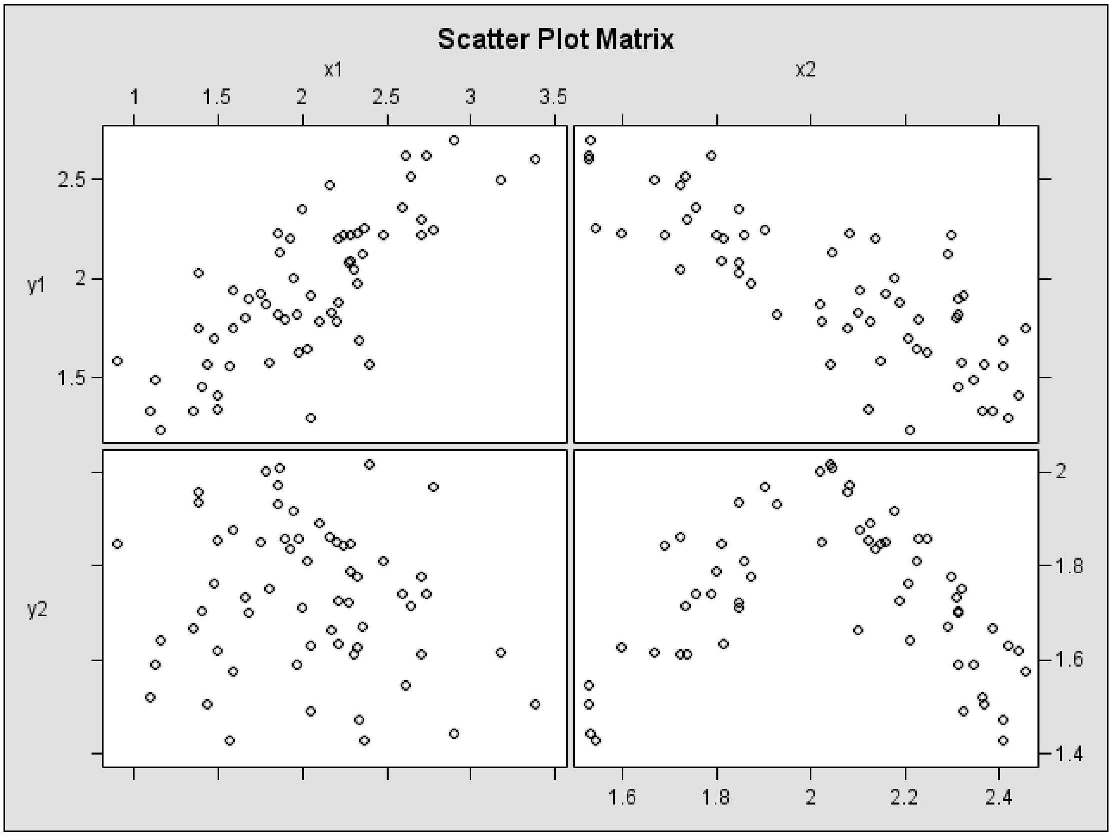


Figure 1.4. Correlations between Two Variables

The scatter plot matrix shows a positive correlation between variables Y1 and X1, a negative correlation between Y1 and X2, and no clear correlation between Y2 and X1. The plot also shows no clear linear correlation between Y2 and X2, even though Y2 is dependent on X2.

The formula for the population Pearson product-moment correlation, denoted ρ_{xy}, is

$$\rho_{xy} = \frac{\mathrm{Cov}(x,y)}{\sqrt{V(x)V(y)}} = \frac{\mathrm{E}(\,(x-\mathrm{E}(x))(y-\mathrm{E}(y))\,)}{\sqrt{\mathrm{E}(x-\mathrm{E}(x))^2\,\mathrm{E}(y-\mathrm{E}(y))^2}}$$

The sample correlation, such as a Pearson product-moment correlation or weighted product-moment correlation, estimates the population correlation. The formula for the sample Pearson product-moment correlation is

$$r_{xy} = \frac{\sum_i(\,(x_i-\bar{x})(y_i-\bar{y})\,)}{\sqrt{\sum_i(x_i-\bar{x})^2\,\sum_i(y_i-\bar{y})^2}}$$

where $\bar{x}$ is the sample mean of x and $\bar{y}$ is the sample mean of y. The formula for a weighted Pearson product-moment correlation is

$$r_{xy} = \frac{\sum_i w_i(x_i-\bar{x}_w)(y_i-\bar{y}_w)}{\sqrt{\sum_i w_i(x_i-\bar{x}_w)^2\,\sum_i w_i(y_i-\bar{y}_w)^2}}$$

where w_i is the weight, $\bar{x}_w$ is the weighted mean of x, and $\bar{y}_w$ is the weighted mean of y.

Probability Values

Probability values for the Pearson correlation are computed by treating

$$t = (n-2)^{1/2} \left(\frac{r^2}{1-r^2} \right)^{1/2}$$

as coming from a *t* distribution with $(n-2)$ degrees of freedom, where r is the sample correlation.

Spearman Rank-Order Correlation

Spearman rank-order correlation is a nonparametric measure of association based on the ranks of the data values. The formula is

$$\theta = \frac{\sum_i ((R_i - \bar{R})(S_i - \bar{S}))}{\sqrt{\sum_i (R_i - \bar{R})^2 \sum (S_i - \bar{S})^2}}$$

where R_i is the rank of x_i, S_i is the rank of y_i, $\bar{R}$ is the mean of the R_i values, and $\bar{S}$ is the mean of the S_i values.

PROC CORR computes the Spearman correlation by ranking the data and using the ranks in the Pearson product-moment correlation formula. In case of ties, the averaged ranks are used.

Probability Values

Probability values for the Spearman correlation are computed by treating

$$t = (n-2)^{1/2} \left(\frac{r^2}{1-r^2} \right)^{1/2}$$

as coming from a *t* distribution with $(n-2)$ degrees of freedom, where r is the sample Spearman correlation.

Kendall's Tau-b Correlation Coefficient

Kendall's tau-b is a nonparametric measure of association based on the number of concordances and discordances in paired observations. Concordance occurs when paired observations vary together, and discordance occurs when paired observations vary differently. The formula for Kendall's tau-b is

$$\tau = \frac{\sum_{i<j} (\mathrm{sgn}(x_i - x_j)\mathrm{sgn}(y_i - y_j))}{\sqrt{(T_0 - T_1)(T_0 - T_2)}}$$

where $T_0 = n(n-1)/2$, $T_1 = \sum_k t_k(t_k-1)/2$, and $T_2 = \sum_l u_l(u_l-1)/2$. The t_k is the number of tied x values in the kth group of tied x values, u_l is the number of tied y values in the lth group of tied y values, n is the number of observations, and $\mathrm{sgn}(z)$ is defined as

$$\mathrm{sgn}(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z = 0 \\ -1 & \text{if } z < 0 \end{cases}$$

PROC CORR computes Kendall's tau-b by ranking the data and using a method similar to Knight (1966). The data are double sorted by ranking observations according to values of the first variable and reranking the observations according to values of the second variable. PROC CORR computes Kendall's tau-b from the number of interchanges of the first variable and corrects for tied pairs (pairs of observations with equal values of X or equal values of Y).

Probability Values

Probability values for Kendall's tau-b are computed by treating

$$\frac{s}{\sqrt{V(s)}}$$

as coming from a standard normal distribution where

$$s = \sum_{i<j} (\mathrm{sgn}(x_i - x_j)\mathrm{sgn}(y_i - y_j))$$

and $V(s)$, the variance of s, is computed as

$$V(s) = \frac{v_0 - v_t - v_u}{18} + \frac{v_1}{2n(n-1)} + \frac{v_2}{9n(n-1)(n-2)}$$

where

$$v_0 = n(n-1)(2n+5)$$

$$v_t = \textstyle\sum_k t_k(t_k-1)(2t_k+5)$$

$$v_u = \textstyle\sum_l u_l(u_l-1)(2u_l+5)$$

$$v_1 = \left(\textstyle\sum_k t_k(t_k-1)\right)\left(\sum u_i(u_l-1)\right)$$

$$v_2 = \left(\textstyle\sum_l t_i(t_k-1)(t_k-2)\right)\left(\sum u_l(u_l-1)(u_l-2)\right)$$

The sums are over tied groups of values where t_i is the number of tied x values and u_i is the number of tied y values (Noether 1967). The sampling distribution of Kendall's partial tau-b is unknown; therefore, the probability values are not available.

Hoeffding Dependence Coefficient

Hoeffding's measure of dependence, D, is a nonparametric measure of association that detects more general departures from independence. The statistic approximates a weighted sum over observations of chi-square statistics for two-by-two classification tables (Hoeffding 1948). Each set of (x, y) values are cut points for the classification. The formula for Hoeffding's D is

$$D = 30\,\frac{(n-2)(n-3)D_1 + D_2 - 2(n-2)D_3}{n(n-1)(n-2)(n-3)(n-4)}$$

where $D_1 = \sum_i (Q_i - 1)(Q_i - 2)$, $D_2 = \sum_i (R_i - 1)(R_i - 2)(S_i - 1)(S_i - 2)$, and $D_3 = \sum_i (R_i - 2)(S_i - 2)(Q_i - 1)$. R_i is the rank of x_i, S_i is the rank of y_i, and Q_i (also called the bivariate rank) is 1 plus the number of points with both x and y values less than the ith point.

A point that is tied on only the x value or y value contributes 1/2 to Q_i if the other value is less than the corresponding value for the ith point.

A point that is tied on both x and y contributes 1/4 to Q_i. PROC CORR obtains the Q_i values by first ranking the data. The data are then double sorted by ranking observations according to values of the first variable and reranking the observations according to values of the second variable. Hoeffding's D statistic is computed using the number of interchanges of the first variable. When no ties occur among data set observations, the D statistic values are between -0.5 and 1, with 1 indicating complete dependence. However, when ties occur, the D statistic may result in a smaller value. That is, for a pair of variables with identical values, the Hoeffding's D statistic may be less than 1. With a large number of ties in a small data set, the D statistic may be less than -0.5. For more information about Hoeffding's D, refer to Hollander and Wolfe (1973, p. 228).

Probability Values

The probability values for Hoeffding's D statistic are computed using the asymptotic distribution computed by Blum, Kiefer, and Rosenblatt (1961). The formula is

$$\frac{(n-1)\pi^4}{60} D + \frac{\pi^4}{72}$$

which comes from the asymptotic distribution. If the sample size is less than 10, refer to the tables for the distribution of D in Hollander and Wolfe (1973).

Partial Correlation

A partial correlation measures the strength of a relationship between two variables, while controlling the effect of other variables. The Pearson partial correlation between two variables, after controlling for variables in the PARTIAL statement, is equivalent to the Pearson correlation between the residuals of the two variables after regression on the controlling variables.

Let $\mathbf{y} = (y_1, y_2, \ldots, y_v)$ be the set of variables to correlate and $\mathbf{z} = (z_1, z_2, \ldots, z_p)$ be the set of controlling variables. The population Pearson partial correlation between the ith and the jth variables of $\mathbf{y}$ given $\mathbf{z}$ is the correlation between errors $(y_i - \mathrm{E}(y_i))$ and $(y_j - \mathrm{E}(y_j))$, where

$$\mathrm{E}(y_i) = \alpha_i + \mathbf{z}\boldsymbol{\beta}_i \quad \text{and} \quad \mathrm{E}(y_j) = \alpha_j + \mathbf{z}\boldsymbol{\beta}_j$$

are the regression models for variables y_i and y_j given the set of controlling variables $\mathbf{z}$, respectively.

For a given sample of observations, a sample Pearson partial correlation between y_i and y_j given $\mathbf{z}$ is derived from the residuals $y_i - \hat{y}_i$ and $y_j - \hat{y}_j$, where

$$\hat{y}_i = \hat{\alpha}_i + \mathbf{z}\hat{\boldsymbol{\beta}}_i \quad \text{and} \quad \hat{y}_j = \hat{\alpha}_j + \mathbf{z}\hat{\boldsymbol{\beta}}_j$$

are fitted values from regression models for variables y_i and y_j given $\mathbf{z}$.

The partial corrected sums of squares and crossproducts (CSSCP) of $\mathbf{y}$ given $\mathbf{z}$ are the corrected sums of squares and crossproducts of the residuals $\mathbf{y} - \hat{\boldsymbol{y}}$. Using these partial corrected sums of squares and crossproducts, you can calculate the partial partial covariances and partial correlations.

PROC CORR derives the partial corrected sums of squares and crossproducts matrix by applying the Cholesky decomposition algorithm to the CSSCP matrix. For Pearson partial correlations, let S be the partitioned CSSCP matrix between two sets of variables, $\mathbf{z}$ and $\mathbf{y}$:

$$\mathbf{S} \;=\; \begin{bmatrix} \mathbf{S}_{zz} & \mathbf{S}_{zy} \\ \mathbf{S}'_{zy} & \mathbf{S}_{yy} \end{bmatrix}$$

PROC CORR calculates $S_{yy.z}$, the partial CSSCP matrix of $\mathbf{y}$ after controlling for $\mathbf{z}$, by applying the Cholesky decomposition algorithm sequentially on the rows associated with $\mathbf{z}$, the variables being partialled out.

After applying the Cholesky decomposition algorithm to each row associated with variables $\mathbf{z}$, PROC CORR checks all higher numbered diagonal elements associated with $\mathbf{z}$ for singularity. A variable is considered singular if the value of the corresponding diagonal element is less than ε times the original unpartialled corrected sum of squares of that variable. You can specify the singularity criterion ε using the SINGULAR= option. For Pearson partial correlations, a controlling variable $\mathbf{z}$ is considered singular if the R^2 for predicting this variable from the variables that are already partialled out exceeds $1 - \varepsilon$. When this happens, PROC CORR excludes the variable from the analysis. Similarly, a variable is considered singular if the R^2 for predicting this variable from the controlling variables exceeds $1 - \varepsilon$. When this happens, its associated diagonal element and all higher numbered elements in this row or column are set to zero.

After the Cholesky decomposition algorithm is performed on all rows associated with $\mathbf{z}$, the resulting matrix has the form

$$T = \begin{bmatrix} \mathbf{T}_{zz} & \mathbf{T}_{zy} \\ 0 & \mathbf{S}_{yy.z} \end{bmatrix}$$

where T_{zz} is an upper triangular matrix with $T'_{zz}T_{zz} = S'_{zz}$, $T'_{zz}T_{zy} = S'_{zy}$, and $S_{yy.z} = S_{yy} - T'_{zy}T_{zy}$.

If S_{zz} is positive definite, then $T_{zy} = {T'_{zz}}^{-1}S'_{zy}$ and the partial CSSCP matrix $S_{yy.z}$ is identical to the matrix derived from the formula

$$S_{yy.z} = S_{yy} - S'_{zy}S'^{-1}_{zz}S_{zy}$$

The partial variance-covariance matrix is calculated with the variance divisor (VARDEF= option). PROC CORR then uses the standard Pearson correlation formula on the partial variance-covariance matrix to calculate the Pearson partial correlation matrix.

When a correlation matrix is positive definite, the resulting partial correlation between variables x and y after adjusting for a single variable z is identical to that obtained from the first-order partial correlation formula

$$r_{xy.z} = \frac{r_{xy} - r_{xz}r_{yz}}{\sqrt{(1 - r^2_{xz})(1 - r^2_{yz})}}$$

where r_{xy}, r_{xz}, and r_{yz} are the appropriate correlations.

The formula for higher-order partial correlations is a straightforward extension of the preceding first-order formula. For example, when the correlation matrix is positive definite, the partial correlation between x and y controlling for both z_1 and z_2 is identical to the second-order partial correlation formula

$$r_{xy.z_1z_2} = \frac{r_{xy.z_1} - r_{xz_2.z_1}r_{yz_2.z_1}}{\sqrt{(1 - r^2_{xz_2.z_1})(1 - r^2_{yz_2.z_1})}}$$

where $r_{xy.z_1}$, $r_{xz_2.z_1}$, and $r_{yz_2.z_1}$ are first-order partial correlations among variables x, y, and z_2 given z_1.

To derive the corresponding Spearman partial rank-order correlations and Kendall partial tau-b correlations, PROC CORR applies the Cholesky decomposition algorithm to the Spearman rank-order correlation matrix and Kendall's tau-b correlation matrix and uses the correlation formula. That is, the Spearman partial correlation is equivalent to the Pearson correlation between the residuals of the linear regression of the ranks of the two variables on the ranks of the partialled variables. Thus, if a PARTIAL statement is specified with the CORR=SPEARMAN option, the residuals of the ranks of the two variables are displayed in the plot. The partial tau-b correlations range from –1 to 1. However, the sampling distribution of this partial tau-b is unknown; therefore, the probability values are not available.

Probability Values

Probability values for the Pearson and Spearman partial correlations are computed by treating

$$\frac{(n-k-2)^{1/2} r}{(1-r^2)^{1/2}}$$

as coming from a *t* distribution with $(n-k-2)$ degrees of freedom, where r is the partial correlation and k is the number of variables being partialled out.

Fisher's z Transformation

For a sample correlation r using a sample from a bivariate normal distribution with correlation $\rho = 0$, the statistic

$$t_r = (n-2)^{1/2} \left(\frac{r^2}{1-r^2} \right)^{1/2}$$

has a Student-t distribution with ($n-2$) degrees of freedom.

With the monotone transformation of the correlation r (Fisher 1921)

$$z_r = \tanh^{-1}(r) = \frac{1}{2} \log \left(\frac{1+r}{1-r} \right)$$

the statistic z has an approximate normal distribution with mean and variance

$$E(z_r) = \zeta + \frac{\rho}{2(n-1)}$$

$$V(z_r) = \frac{1}{n-3}$$

where $\zeta = \tanh^{-1}(\rho)$.

For the transformed z_r, the approximate variance $V(z_r) = 1/(n-3)$ is independent of the correlation ρ. Furthermore, even the distribution of z_r is not strictly normal; it tends to normality rapidly as the sample size increases for any values of ρ (Fisher 1970, pp. 200 –201).

For the null hypothesis H_0: $\rho = \rho_0$, the p-values are computed by treating

$$z_r - \zeta_0 - \frac{\rho_0}{2(n-1)}$$

as a normal random variable with mean zero and variance $1/(n-3)$, where $\zeta_0 = \tanh^{-1}(\rho_0)$ (Fisher 1970, p. 207; Anderson 1984, p. 123).

Note that the bias adjustment, $\rho_0/(2(n-1))$, is always used when computing p-values under the null hypothesis H_0: $\rho = \rho_0$ in the CORR procedure.

The ALPHA= option in the FISHER option specifies the value α for the confidence level $1 - \alpha$, the RHO0= option specifies the value ρ_0 in the hypothesis H_0: $\rho = \rho_0$, and the BIASADJ= option specifies whether the bias adjustment is to be used for the confidence limits.

The TYPE= option specifies the type of confidence limits. The TYPE=TWOSIDED option requests two-sided confidence limits and a p-value under the hypothesis H_0: $\rho = \rho_0$. For a one-sided confidence limit, the TYPE=LOWER option requests a lower confidence limit and a p-value under the hypothesis H_0: $\rho <= \rho_0$, and the TYPE=UPPER option requests an upper confidence limit and a p-value under the hypothesis H_0: $\rho >= \rho_0$.

Confidence Limits for the Correlation

The confidence limits for the correlation ρ are derived through the confidence limits for the parameter ζ, with or without the bias adjustment.

Without a bias adjustment, confidence limits for ζ are computed by treating

$$z_r - \zeta$$

as having a normal distribution with mean zero and variance $1/(n-3)$.

That is, the two-sided confidence limits for ζ are computed as

$$\zeta_l = z_r - z_{(1-\alpha/2)} \sqrt{\frac{1}{n-3}}$$

$$\zeta_u = z_r + z_{(1-\alpha/2)} \sqrt{\frac{1}{n-3}}$$

where $z_{(1-\alpha/2)}$ is the $100(1-\alpha/2)$ percentage point of the standard normal distribution.

With a bias adjustment, confidence limits for ζ are computed by treating

$$z_r - \zeta - \mathrm{bias}(r)$$

as having a normal distribution with mean zero and variance $1/(n-3)$, where the bias adjustment function (Keeping 1962, p. 308) is

$$\mathrm{bias}(r_r) = \frac{r}{2(n-1)}$$

That is, the two-sided confidence limits for ζ are computed as

$$\zeta_l = z_r - \mathrm{bias}(r) - z_{(1-\alpha/2)} \sqrt{\frac{1}{n-3}}$$

$$\zeta_u = z_r - \text{bias}(r) + z_{(1-\alpha/2)} \sqrt{\frac{1}{n-3}}$$

These computed confidence limits of ζ_l and ζ_u are then transformed back to derive the confidence limits for the correlation ρ:

$$r_l = \tanh(\zeta_l) = \frac{\exp(2\zeta_l) - 1}{\exp(2\zeta_l) + 1}$$

$$r_u = \tanh(\zeta_u) = \frac{\exp(2\zeta_u) - 1}{\exp(2\zeta_u) + 1}$$

Note that with a bias adjustment, the CORR procedure also displays the following correlation estimate:

$$r_{adj} = \tanh(z_r - \text{bias}(r))$$

Applications of Fisher's z Transformation

Fisher (1970, p. 199) describes the following practical applications of the z transformation:

- Testing whether a population correlation is equal to a given value
- Testing for equality of two population correlations
- Combining correlation estimates from different samples

To test if a population correlation ρ_1 from a sample of n_1 observations with sample correlation r_1 is equal to a given ρ_0, first apply the z transformation to r_1 and ρ_0: $z_1 = \tanh^{-1}(r_1)$ and $\zeta_0 = \tanh^{-1}(\rho_0)$.

The p-value is then computed by treating

$$z_1 - \zeta_0 - \frac{\rho_0}{2(n_1 - 1)}$$

as a normal random variable with mean zero and variance $1/(n_1 - 3)$.

Assume that sample correlations r_1 and r_2 are computed from two independent samples of n_1 and n_2 observations, respectively. To test whether the two corresponding population correlations, ρ_1 and ρ_2, are equal, first apply the z transformation to the two sample correlations: $z_1 = \tanh^{-1}(r_1)$ and $z_2 = \tanh^{-1}(r_2)$.

The p-value is derived under the null hypothesis of equal correlation. That is, the difference $z_1 - z_2$ is distributed as a normal random variable with mean zero and variance $1/(n_1 - 3) + 1/(n_2 - 3)$.

Assuming further that the two samples are from populations with identical correlation, a combined correlation estimate can be computed. The weighted average of the corresponding z values is

$$\bar{z} = \frac{(n_1 - 3)z_1 + (n_2 - 3)z_2}{n_1 + n_2 - 6}$$

where the weights are inversely proportional to their variances.

Thus, a combined correlation estimate is $\bar{r} = \tanh(\bar{z})$ and $V(\bar{z}) = 1/(n_1 + n_2 - 6)$. See Example 1.4 for further illustrations of these applications.

Note that this approach can be extended to include more than two samples.

Cronbach's Coefficient Alpha

Analyzing latent constructs such as job satisfaction, motor ability, sensory recognition, or customer satisfaction requires instruments to accurately measure the constructs. Interrelated items may be summed to obtain an overall score for each participant. Cronbach's coefficient alpha estimates the reliability of this type of scale by determining the internal consistency of the test or the average correlation of items within the test (Cronbach 1951).

When a value is recorded, the observed value contains some degree of measurement error. Two sets of measurements on the same variable for the same individual may not have identical values. However, repeated measurements for a series of individuals will show some consistency. Reliability measures internal consistency from one set of measurements to another. The observed value Y is divided into two components, a true value T and a measurement error E. The measurement error is assumed to be independent of the true value, that is,

$$Y = T + E \qquad\qquad \text{Cov}(T, E) = 0$$

The reliability coefficient of a measurement test is defined as the squared correlation between the observed value Y and the true value T, that is,

$$r^2(Y,T) = \frac{\text{Cov}(Y,T)^2}{V(Y)V(T)} = \frac{V(T)^2}{V(Y)V(T)} = \frac{V(T)}{V(Y)}$$

which is the proportion of the observed variance due to true differences among individuals in the sample. If Y is the sum of several observed variables measuring the same feature, you can estimate $V(T)$. Cronbach's coefficient alpha, based on a lower bound for $V(T)$, is an estimate of the reliability coefficient.

Suppose p variables are used with $Y_j = T_j + E_j$ for $j = 1, 2, \ldots, p$, where Y_j is the observed value, T_j is the true value, and E_j is the measurement error. The measurement errors (E_j) are independent of the true values (T_j) and are also independent of each other. Let $Y_0 = \sum_j Y_j$ be the total observed score and $T_0 = \sum_j T_j$ be the total true score. Because

$$(p-1)\sum_j V(T_j) \geq \sum_{i \neq j} \text{Cov}(T_i, T_j)$$

a lower bound for $V(T_0)$ is given by

$$\frac{p}{p-1}\sum_{i \neq j} \text{Cov}(T_i, T_j)$$

With $\text{Cov}(Y_i, Y_j) = \text{Cov}(T_i, T_j)$ for $i \neq j$, a lower bound for the reliability coefficient, $V(T_0)/V(Y_0)$, is then given by the Cronbach's coefficient alpha:

$$\alpha = \left(\frac{p}{p-1}\right)\frac{\sum_{i \neq j} \text{Cov}(Y_i, Y_j)}{V(Y_0)} = \left(\frac{p}{p-1}\right)\left(1 - \frac{\sum_j V(Y_j)}{V(Y_0)}\right)$$

If the variances of the items vary widely, you can standardize the items to a standard deviation of 1 before computing the coefficient alpha. If the variables are dichotomous (0,1), the coefficient alpha is equivalent to the Kuder-Richardson 20 (KR-20) reliability measure.

When the correlation between each pair of variables is 1, the coefficient alpha has a maximum value of 1. With negative correlations between some variables, the coefficient alpha can have a value less than zero. The larger the overall alpha coefficient, the more likely that items contribute to a reliable scale. Nunnally and Bernstein (1994) suggests 0.70 as an acceptable reliability coefficient; smaller reliability coefficients are seen as inadequate. However, this varies by discipline.

To determine how each item reflects the reliability of the scale, you calculate a coefficient alpha after deleting each variable independently from the scale. The Cronbach's coefficient alpha from all variables except the kth variable is given by

$$\alpha_k = \left(\frac{p-1}{p-2}\right)\left(1 - \frac{\sum_{i \neq k} V(Y_i)}{V(\sum_{i \neq k} Y_i)}\right)$$

If the reliability coefficient increases after an item is deleted from the scale, you can assume that the item is not correlated highly with other items in the scale. Conversely, if the reliability coefficient decreases, you can assume that the item is highly correlated with other items in the scale. Refer to *SAS Communications*, Fourth Quarter 1994, for more information on how to interpret Cronbach's coefficient alpha.

Listwise deletion of observations with missing values is necessary to correctly calculate Cronbach's coefficient alpha. PROC CORR does not automatically use listwise deletion if you specify the ALPHA option. Therefore, you should use the NOMISS option if the data set contains missing values. Otherwise, PROC CORR prints a warning message indicating the need to use the NOMISS option with the ALPHA option.

Missing Values

PROC CORR excludes observations with missing values in the WEIGHT and FREQ variables. By default, PROC CORR uses *pairwise deletion* when observations contain missing values. PROC CORR includes all nonmissing pairs of values for each pair of variables in the statistical computations. Therefore, the correlation statistics may be based on different numbers of observations.

If you specify the NOMISS option, PROC CORR uses *listwise deletion* when a value of the VAR or WITH statement variable is missing. PROC CORR excludes all observations with missing values from the analysis. Therefore, the number of observations for each pair of variables is identical.

The PARTIAL statement always excludes the observations with missing values by automatically invoking the NOMISS option. With the NOMISS option, the data are processed more efficiently because fewer resources are needed. Also, the resulting correlation matrix is nonnegative definite.

In contrast, if the data set contains missing values for the analysis variables and the NOMISS option is not specified, the resulting correlation matrix may not be nonnegative definite. This leads to several statistical difficulties if you use the correlations as input to regression or other statistical procedures.

Output Tables

By default, PROC CORR prints a report that includes descriptive statistics and correlation statistics for each variable. The descriptive statistics include the number of observations with nonmissing values, the mean, the standard deviation, the minimum, and the maximum.

If a nonparametric measure of association is requested, the descriptive statistics include the median. Otherwise, the sample sum is included. If a Pearson partial correlation is requested, the descriptive statistics also include the partial variance and partial standard deviation.

If variable labels are available, PROC CORR labels the variables. If you specify the CSSCP, SSCP, or COV option, the appropriate sum-of-squares and crossproducts and covariance matrix appears at the top of the correlation report. If the data set contains missing values, PROC CORR prints additional statistics for each pair of variables. These statistics, calculated from the observations with nonmissing row and column variable values, may include

- SSCP('W','V'), uncorrected sum-of-squares and crossproducts
- USS('W'), uncorrected sum-of-squares for the row variable
- USS('V'), uncorrected sum-of-squares for the column variable
- CSSCP('W','V'), corrected sum-of-squares and crossproducts
- CSS('W'), corrected sum-of-squares for the row variable
- CSS('V'), corrected sum-of-squares for the column variable

- COV('W','V'), covariance
- VAR('W'), variance for the row variable
- VAR('V'), variance for the column variable
- DF('W','V'), divisor for calculating covariance and variances

For each pair of variables, PROC CORR prints the correlation coefficients, the number of observations used to calculate the coefficient, and the p-value.

If you specify the ALPHA option, PROC CORR prints Cronbach's coefficient alpha, the correlation between the variable and the total of the remaining variables, and Cronbach's coefficient alpha using the remaining variables for the raw variables and the standardized variables.

Output Data Sets

If you specify the OUTP=, OUTS=, OUTK=, or OUTH= option, PROC CORR creates an output data set containing statistics for Pearson correlation, Spearman correlation, Kendall's tau-b, or Hoeffding's D, respectively. By default, the output data set is a special data set type (TYPE=CORR) that many SAS/STAT procedures recognize, including PROC REG and PROC FACTOR. When you specify the NOCORR option and the COV, CSSCP, or SSCP option, use the TYPE= data set option to change the data set type to COV, CSSCP, or SSCP.

The output data set includes the following variables:

- BY variables, which identify the BY group when using a BY statement
- _TYPE_ variable, which identifies the type of observation
- _NAME_ variable, which identifies the variable that corresponds to a given row of the correlation matrix
- INTERCEPT variable, which identifies variable sums when specifying the SSCP option
- VAR variables, which identify the variables listed in the VAR statement

You can use a combination of the _TYPE_ and _NAME_ variables to identify the contents of an observation. The _NAME_ variable indicates which row of the correlation matrix the observation corresponds to. The values of the _TYPE_ variable are

- SSCP, uncorrected sums of squares and crossproducts
- CSSCP, corrected sums of squares and crossproducts
- COV, covariances
- MEAN, mean of each variable
- STD, standard deviation of each variable
- N, number of nonmissing observations for each variable
- SUMWGT, sum of the weights for each variable when using a WEIGHT statement
- CORR, correlation statistics for each variable.

If you specify the SSCP option, the OUTP= data set includes an additional observation that contains intercept values. If you specify the ALPHA option, the OUTP= data set also includes observations with the following _TYPE_ values:

- RAWALPHA, Cronbach's coefficient alpha for raw variables
- STDALPHA, Cronbach's coefficient alpha for standardized variables
- RAWALDEL, Cronbach's coefficient alpha for raw variables after deleting one variable
- STDALDEL, Cronbach's coefficient alpha for standardized variables after deleting one variable
- RAWCTDEL, the correlation between a raw variable and the total of the remaining raw variables
- STDCTDEL, the correlation between a standardized variable and the total of the remaining standardized variables

If you use a PARTIAL statement, the statistics are calculated after the variables are partialled. If PROC CORR computes Pearson correlation statistics, MEAN equals zero and STD equals the partial standard deviation associated with the partial variance for the OUTP=, OUTK=, and OUTS= data sets. Otherwise, PROC CORR assigns missing values to MEAN and STD.

Determining Computer Resources

The only factor limiting the number of variables that you can analyze is the amount of available memory. The computer resources that PROC CORR requires depend on which statements and options you specify. To determine the computer resources, define the following variables as follows:

$$\begin{array}{lcl} L & = & \text{number of observations in the data set} \\ C & = & \text{number of correlation types } (C = 1, 2, 3, 4) \\ V & = & \text{number of VAR statement variables} \\ W & = & \text{number of WITH statement variables} \\ P & = & \text{number of PARTIAL statement variables} \end{array}$$

Furthermore, define the following variables:

$$\begin{array}{lcl} T & = & V + W + P \\ K & = & \begin{cases} V * W & \text{when } W > 0 \\ V * (V+1)/2 & \text{when } W = 0 \end{cases} \\ L & = & \begin{cases} K & \text{when } P = 0 \\ T * (T+1)/2 & \text{when } P > 0 \end{cases} \end{array}$$

For small N and large K, the CPU time varies as K for all types of correlations. For large N, the CPU time depends on the type of correlation:

$$\begin{array}{cl} K*N & \text{with PEARSON (default)} \\ T*N*\log N & \text{with SPEARMAN} \\ K*N*\log N & \text{with HOEFFDING or KENDALL} \end{array}$$

You can reduce CPU time by specifying NOMISS. With NOMISS, processing is much faster when most observations do not contain missing values. The options and statements you use in the procedure require different amounts of storage to process the data. For Pearson correlations, the amount of temporary storage needed (in bytes) is

$$M = 40T + 16L + 56K + 56T$$

The NOMISS option decreases the amount of temporary storage by $56K$ bytes, the FISHER option increases the storage by $24K$ bytes, the PARTIAL statement increases the storage by $12T$ bytes, and the ALPHA option increases the storage by $32V + 16$ bytes.

The following example uses a PARTIAL statement, which excludes missing values.

```
proc corr;
   var x1 x2;
   with y1 y2 y3;
   partial z1;
run;
```

Therefore, using $40T + 16L + 56T + 12T$, the minimum temporary storage equals 984 bytes ($T = 2 + 3 + 1$ and $L = T(T + 1)/2$).

Using the SPEARMAN, KENDALL, or HOEFFDING option requires additional temporary storage for each observation. For the most time-efficient processing, the amount of temporary storage (in bytes) is

$$M = 40T + 8K + 8L*C + 12T*N + 28N + QS + QP + QK$$

where

$$QS = \begin{cases} 0 & \text{with NOSIMPLE} \\ 68T & \text{otherwise} \end{cases}$$
$$QP = \begin{cases} 56K & \text{with PEARSON and without NOMISS} \\ 0 & \text{otherwise} \end{cases}$$
$$QK = \begin{cases} 32N & \text{with KENDALL or HOEFFDING} \\ 0 & \text{otherwise} \end{cases}$$

The following example requests Kendall's tau-b coefficients:

```
proc corr kendall;
   var x1 x2 x3;
run;
```

Therefore, the minimum temporary storage in bytes is

$$\begin{aligned} M &= 40*3+8*6+8*6*1+12*3N+28N+3*68+32N \\ &= 420+96N \end{aligned}$$

where N is the number of observations.

If M bytes are not available, PROC CORR must process the data multiple times to compute all the statistics. This reduces the minimum temporary storage you need by $12(T-2)N$ bytes. When this occurs, PROC CORR prints a note suggesting a larger memory region.

ODS Table Names

PROC CORR assigns a name to each table 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.

Table 1.3. ODS Tables Produced with the PROC CORR Statement

ODS Name	Description	Option
Cov	Covariances	COV
CronbachAlpha	Coefficient alpha	ALPHA
CronbachAlphaDel	Coefficient alpha with deleted variable	ALPHA
Csscp	Corrected sums of squares and crossproducts	CSSCP
FisherPearsonCorr	Pearson correlation statistics using Fisher's z Transformation	FISHER
FisherSpearmanCorr	Spearman correlation statistics using Fisher's z Transformation	FISHER SPEARMAN
HoeffdingCorr	Hoeffding's D statistics	HOEFFDING
KendallCorr	Kendall's tau-b coefficients	KENDALL
PearsonCorr	Pearson correlations	PEARSON
SimpleStats	Simple descriptive statistics	
SpearmanCorr	Spearman correlations	SPEARMAN
Sscp	Sums of squares and crossproducts	SSCP
VarInformation	Variable information	

Table 1.4. ODS Tables Produced with the PARTIAL Statement

ODS Name	Description	Option
FisherPearsonPartialCorr	Pearson Partial Correlation Statistics Using Fisher's z Transformation	FISHER
FisherSpearmanPartialCorr	Spearman Partial Correlation Statistics Using Fisher's z Transformation	FISHER SPEARMAN

Table 1.4. (continued)

ODS Name	Description	Option
PartialCsscp	Partial corrected sums of squares and crossproduct	CSSCP
PartialCov	Partial covariances	COV
PartialKendallCorr	Partial Kendall tau-b coefficients	KENDALL
PartialPearsonCorr	Partial Pearson correlations	
PartialSpearmanCorr	Partial Spearman correlations	SPEARMAN

ODS Graphics (Experimental)

This section describes the use of ODS for creating graphics with the CORR procedure. These graphics are experimental in this release, meaning that both the graphical results and the syntax for specifying them are subject to change in a future release.

To request these graphs, you must specify the ODS GRAPHICS statement in addition to the following options in the PROC CORR statement. For more information on the ODS GRAPHICS statement, refer to Chapter 15, "Statistical Graphics Using ODS" (*SAS/STAT User's Guide*).

PLOTS
PLOTS = MATRIX < *(matrix-options)* >
PLOTS = SCATTER < *(scatter-options)* >
PLOTS = (MATRIX < *(matrix-options)* > **SCATTER** < *(scatter-options)* > **)**

requests a scatter plot matrix for all variables, scatter plots for all pairs of variables, or both. If only the option keyword PLOTS is specified, the PLOTS=MATRIX option is used. When you specify the PLOTS option, the Pearson correlations will also be displayed.

You can specify the following with the PLOTS= *option*:

MATRIX < *(matrix-options)* >

requests a scatter plot matrix for all variables. That is, the procedure displays a symmetric matrix plot with variables in the VAR list if a WITH statement is not specified. Otherwise, the procedure displays a rectangular matrix plot with the WITH variables appear down the side and the VAR variables appear across the top.

The available *matrix-options* are:

NMAXVAR=*n*

specifies the maximum number of variables in the VAR list to be displayed in the matrix plot, where $n \geq 0$. If you specify NMAXVAR=0, then the total number of variables in the VAR list is used and no restriction occurs. By default, NMAXVAR=5.

NMAXWITH=*n*

specifies the maximum number of variables in the WITH list to be displayed in the matrix plot, where $n \geq 0$. If you specify NMAXWITH=0, then the total number of variables in the WITH list is used and no restriction occurs. By default, NMAXWITH=5.

SCATTER < *(scatter-options)* >

requests a scatter plot for each pair of variables. That is, the procedure displays a scatter plot for each pair of distinct variables from the VAR list if a WITH statement is not specified. Otherwise, the procedure displays a scatter plot for each pair of variables, one from the WITH list and the other from the VAR list.

The available *scatter-options* are:

ALPHA=*numbers*

specifies the α values for the confidence or prediction ellipses to be displayed in the scatter plots, where $0 < \alpha < 1$. For each α value specified, a $(1 - \alpha)$ confidence or prediction ellipse is created. By default, $\alpha = 0.05$.

ELLIPSE=PREDICTION | MEAN | NONE

requests prediction ellipses for new observations (ELLIPSE=PREDICTION), confidence ellipses for the mean (ELLIPSE=MEAN), or no ellipses (ELLIPSE=NONE) to be created in the scatter plots. By default, ELLIPSE=PREDICTION.

NOINSET

suppresses the default inset of summary information for the scatter plot. The inset table is displayed next to the scatter plot and contains statistics such as number of observations (NObs), correlation, and p-value (Prob >|r|).

NOLEGEND

suppresses the default legend for overlaid prediction or confidence ellipses. The legend table is displayed next to the scatter plot and identifies each ellipse displayed in the plot.

NMAXVAR=*n*

specifies the maximum number of variables in the VAR list to be displayed in the plots, where $n \geq 0$. If you specify NMAXVAR=0, then the total number of variables in the VAR list is used and no restriction occurs. By default, NMAXVAR=5.

NMAXWITH=*n*

specifies the maximum number of variables in the WITH list to be displayed in the plots, where $n \geq 0$. If you specify NMAXWITH=0, then the total number of variables in the WITH list is used and no restriction occurs. By default, NMAXWITH=5.

When the relationship between two variables is nonlinear or when outliers are present, the correlation coefficient may incorrectly estimate the strength of the relationship. Plotting the data enables you to verify the linear relationship and to identify the potential outliers.

The partial correlation between two variables, after controlling for variables in the PARTIAL statement, is the correlation between the residuals of the linear regression of the two variables on the partialled variables. Thus, if a PARTIAL statement is also specified, the residuals of the analysis variables are displayed in the scatter plot matrix and scatter plots.

Confidence and Prediction Ellipses

The CORR procedure optionally provides two types of ellipses for each pair of variables in a scatter plot. One is a confidence ellipse for the population mean, and the other is a prediction ellipse for a new observation. Both assume a bivariate normal distribution.

Let $\bar{\mathbf{Z}}$ and $\mathbf{S}$ be the sample mean and sample covariance matrix of a random sample of size n from a bivariate normal distribution with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$. The variable $\bar{\mathbf{Z}} - \boldsymbol{\mu}$ is distributed as a bivariate normal variate with mean zero and covariance $(1/n)\boldsymbol{\Sigma}$, and it is independent of $\mathbf{S}$. Using Hotelling's T^2 statistic, which is defined as

$$T^2 = n(\bar{\mathbf{Z}} - \boldsymbol{\mu})'\mathbf{S}^{-1}(\bar{\mathbf{Z}} - \boldsymbol{\mu})$$

a $100(1-\alpha)\%$ confidence ellipse for $\boldsymbol{\mu}$ is computed from the equation

$$\frac{n}{n-1}(\bar{\mathbf{Z}} - \boldsymbol{\mu})'\mathbf{S}^{-1}(\bar{\mathbf{Z}} - \boldsymbol{\mu}) = \frac{2}{n-2}F_{2,n-2}(1-\alpha)$$

where $F_{2,n-2}(1-\alpha)$ is the $(1-\alpha)$ critical value of an F distribution with degrees of freedom 2 and $n-2$.

A prediction ellipse is a region for predicting a new observation in the population. It also approximates a region containing a specified percentage of the population.

Denote a new observation as the bivariate random variable $\mathbf{Z}_{\text{new}}$. The variable

$$\mathbf{Z}_{\text{new}} - \bar{\mathbf{Z}} = (\mathbf{Z}_{\text{new}} - \boldsymbol{\mu}) - (\bar{\mathbf{Z}} - \boldsymbol{\mu})$$

is distributed as a bivariate normal variate with mean zero (the zero vector) and covariance $(1+1/n)\boldsymbol{\Sigma}$, and it is independent of $\mathbf{S}$. A $100(1-\alpha)\%$ prediction ellipse is then given by the equation

$$\frac{n}{n-1}(\bar{\mathbf{Z}} - \boldsymbol{\mu})'\mathbf{S}^{-1}(\bar{\mathbf{Z}} - \boldsymbol{\mu}) = \frac{2(n+1)}{n-2}F_{2,n-2}(1-\alpha)$$

The family of ellipses generated by different critical values of the F distribution has a common center (the sample mean) and common major and minor axis directions.

The shape of an ellipse depends on the aspect ratio of the plot. The ellipse indicates the correlation between the two variables if the variables are standardized (by dividing the variables by their respective standard deviations). In this situation, the ratio between the major and minor axis lengths is

$$\sqrt{\frac{1+|r|}{1-|r|}}$$

In particular, if $r = 0$, the ratio is 1, which corresponds to a circular confidence contour and indicates that the variables are uncorrelated. A larger value of the ratio indicates a larger positive or negative correlation between the variables.

ODS Graph Names

The CORR procedure assigns a name to each graph it creates using ODS. You can use these names to reference the graphs when using ODS. The names are listed in Table 1.5.

To request these graphs you must specify the ODS GRAPHICS statement in addition to the options and statements indicated in Table 1.5. For more information on the ODS GRAPHICS statement, refer to Chapter 15, "Statistical Graphics Using ODS" (*SAS/STAT User's Guide*).

Table 1.5. ODS Graphics Produced by PROC CORR

ODS Graph Name	Plot Description	Option	Statement
ScatterPlot	Scatter plot	PLOTS=SCATTER	
RecMatrixPlot	Rectangular scatter plot matrix	PLOTS=MATRIX	WITH
SymMatrixPlot	Symmetric scatter plot matrix	PLOTS=MATRIX	(omit WITH)

Examples

Example 1.1. Computing Four Measures of Association

This example produces a correlation analysis with descriptive statistics and four measures of association: the Pearson product-moment correlation, the Spearman rank-order correlation, Kendall's tau-b coefficients, and Hoeffding's measure of dependence, D.

The Fitness data set created in the "Getting Started" section beginning on page 4 contains measurements from a study of physical fitness of 31 participants. The following statements request all four measures of association for the variables Weight, Oxygen, and Runtime.

```
ods html;
ods graphics on;

title 'Measures of Association for a Physical Fitness Study';
proc corr data=Fitness pearson spearman kendall hoeffding
          plots;
   var Weight Oxygen RunTime;
run;

ods graphics off;
ods html close;
```

Note that Pearson correlations are computed by default only if all three nonparametric correlations (SPEARMAN, KENDALL, and HOEFFDING) are not specified. Otherwise, you need to specify the PEARSON option explicitly to compute Pearson correlations.

By default, observations with nonmissing values for each variable are used to derive the univariate statistics for that variable. When nonparametric measures of association are specified, the procedure displays the median instead of the sum as an additional descriptive measure.

Output 1.1.1. Simple Statistics

```
             Measures of Association for a Physical Fitness Study

                              The CORR Procedure

                   3  Variables:    Weight    Oxygen    RunTime


                              Simple Statistics

Variable          N        Mean     Std Dev      Median     Minimum     Maximum

Weight           31    77.44452     8.32857    77.45000    59.08000    91.63000
Oxygen           29    47.22721     5.47718    46.67200    37.38800    60.05500
RunTime          29    10.67414     1.39194    10.50000     8.17000    14.03000
```

Output 1.1.2. Pearson Correlation Coefficients

```
             Measures of Association for a Physical Fitness Study

                      Pearson Correlation Coefficients
                         Prob > |r| under H0: Rho=0
                           Number of Observations

                               Weight          Oxygen         RunTime

              Weight          1.00000        -0.15358         0.20072
                                               0.4264          0.2965
                                   31              29              29

              Oxygen         -0.15358         1.00000        -0.86843
                               0.4264                          <.0001
                                   29              29              28

              RunTime         0.20072        -0.86843         1.00000
                               0.2965          <.0001
                                   29              28              29
```

The Pearson correlation is a parametric measure of association for two continuous random variables. When there is missing data, the number of observations used to calculate the correlation can vary.

In Output 1.1.2, the Pearson correlation between Runtime and Oxygen is -0.86843, which is significant with a p-value less than 0.0001. This indicates a strong negative linear relationship between these two variables. As Runtime increases, Oxygen decreases linearly.

The Spearman rank-order correlation is a nonparametric measure of association based on the ranks of the data values. The "Spearman Correlation Coefficients" table shown in Output 1.1.3 displays results similar to those of the "Pearson Correlation Coefficients" table.

Output 1.1.3. Spearman Correlation Coefficients

```
                Measures of Association for a Physical Fitness Study

                        Spearman Correlation Coefficients
                           Prob > |r| under H0: Rho=0
                             Number of Observations

                            Weight        Oxygen       RunTime

          Weight           1.00000      -0.06824       0.13749
                                          0.7250        0.4769
                                31            29            29

          Oxygen          -0.06824       1.00000      -0.80131
                            0.7250                      <.0001
                                29            29            28

          RunTime          0.13749      -0.80131       1.00000
                            0.4769        <.0001
                                29            28            29
```

Output 1.1.4. Kendall's Tau-b Correlation Coefficients

```
                Measures of Association for a Physical Fitness Study

                      Kendall Tau b Correlation Coefficients
                           Prob > |r| under H0: Rho=0
                             Number of Observations

                            Weight        Oxygen       RunTime

          Weight           1.00000      -0.00988       0.06675
                                          0.9402        0.6123
                                31            29            29

          Oxygen          -0.00988       1.00000      -0.62434
                            0.9402                      <.0001
                                29            29            28

          RunTime          0.06675      -0.62434       1.00000
                            0.6123        <.0001
                                29            28            29
```

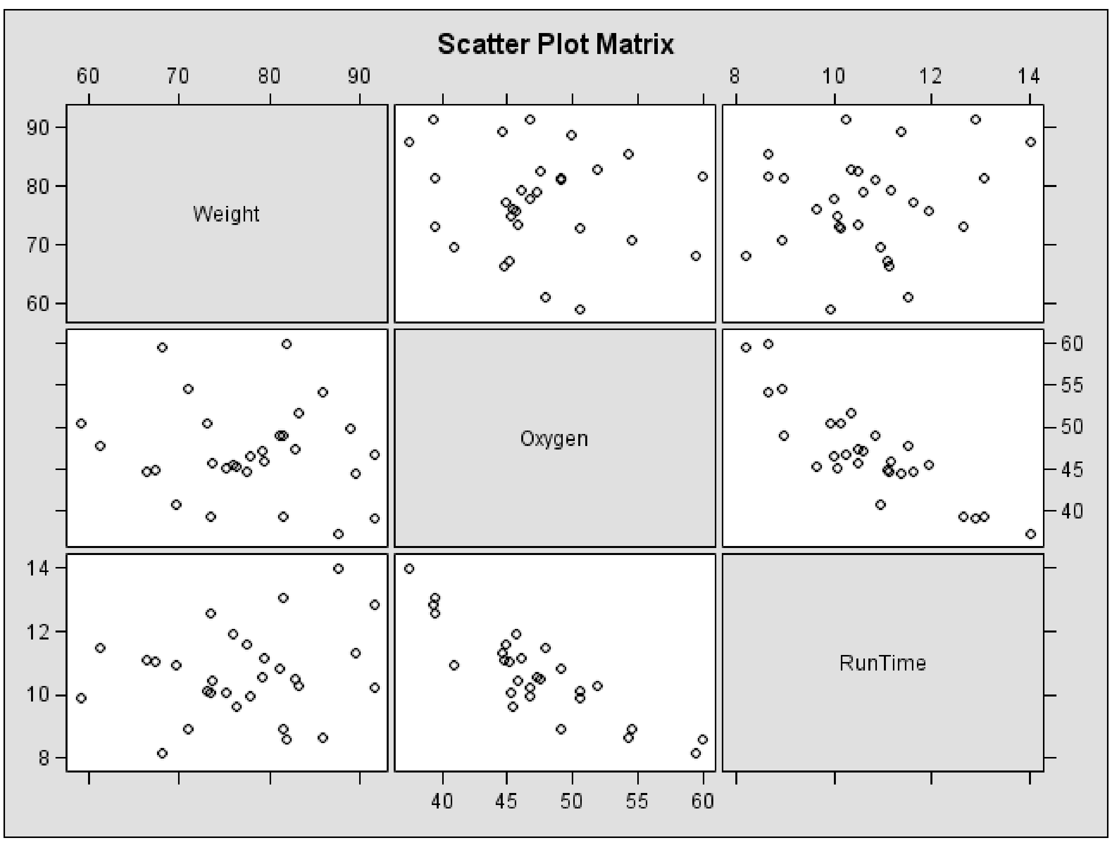
Kendall's tau-b is a nonparametric measure of association based on the number of concordances and discordances in paired observations. The "Kendall Tau-b Correlation Coefficients" table shown in Output 1.1.4 displays results similar to those of the "Pearson Correlation Coefficients" table in Output 1.1.2.

Hoeffding's measure of dependence, D, is a nonparametric measure of association that detects more general departures from independence. Without ties in the variables, the values of the D statistic can vary between -0.5 and 1, with 1 indicating complete dependence. Otherwise, the D statistic can result in a smaller value. Since ties occur in the variable Weight, the D statistic for the Weight variable is less than 1, as shown in the "Hoeffding Dependence Coefficients" table in Output 1.1.5.

Output 1.1.5. Hoeffding's Dependence Coefficients

```
               Measures of Association for a Physical Fitness Study

                       Hoeffding Dependence Coefficients
                             Prob > D under H0: D=0
                             Number of Observations

                              Weight         Oxygen        RunTime

                Weight       0.97690       -0.00497       -0.02355
                              <.0001         0.5101         1.0000
                                  31             29             29

                Oxygen      -0.00497        1.00000        0.23449
                              0.5101                        <.0001
                                  29             29             28

                RunTime     -0.02355        0.23449        1.00000
                              1.0000         <.0001
                                  29             28             29
```

Output 1.1.6. Symmetric Scatter Plot Matrix (Experimental)



The experimental PLOTS option requests a symmetric scatter plot for the analysis variables listed in the VAR statement. The strong negative linear relationship between Oxygen and Runtime is evident in Output 1.1.6.

This display is requested by specifying both the ODS GRAPHICS statement and the PLOTS option. For general information about ODS graphics, refer to Chapter 15, “Statistical Graphics Using ODS” (*SAS/STAT User’s Guide*). For specific information about the graphics available in the CORR procedure, see the section “ODS Graphics” on page 31.

Example 1.2. Computing Correlations between Two Sets of Variables

The following statements create a data set which contains measurements for four iris parts from Fisher’s iris data (1936): sepal length, sepal width, petal length, and petal width. Each observation represents one specimen.

```
*------------------- Data on Iris Setosa --------------------*
| The data set contains 50 iris specimens from the species   |
| Iris Setosa with the following four measurements:          |
| SepalLength (sepal length)                                 |
| SepalWidth  (sepal width)                                  |
| PetalLength (petal length)                                 |
| PetalWidth  (petal width)                                  |
| Certain values were changed to missing for the analysis.   |
*------------------------------------------------------------*;
data Setosa;
  input SepalLength SepalWidth PetalLength PetalWidth @@;
  label sepallength='Sepal Length in mm.'
        sepalwidth='Sepal Width in mm.'
        petallength='Petal Length in mm.'
        petalwidth='Petal Width in mm.';
  datalines;
50 33 14 02  46 34 14 03  46 36 .  02
51 33 17 05  55 35 13 02  48 31 16 02
52 34 14 02  49 36 14 01  44 32 13 02
50 35 16 06  44 30 13 02  47 32 16 02
48 30 14 03  51 38 16 02  48 34 19 02
50 30 16 02  50 32 12 02  43 30 11 .
58 40 12 02  51 38 19 04  49 30 14 02
51 35 14 02  50 34 16 04  46 32 14 02
57 44 15 04  50 36 14 02  54 34 15 04
52 41 15 .   55 42 14 02  49 31 15 02
54 39 17 04  50 34 15 02  44 29 14 02
47 32 13 02  46 31 15 02  51 34 15 02
50 35 13 03  49 31 15 01  54 37 15 02
54 39 13 04  51 35 14 03  48 34 16 02
48 30 14 01  45 23 13 03  57 38 17 03
51 38 15 03  54 34 17 02  51 37 15 04
52 35 15 02  53 37 15 02
;
```

The following statements request a correlation analysis between two sets of variables, the sepal measurements and the petal measurements.

```
ods html;
ods graphics on;

title 'Fisher (1936) Iris Setosa Data';
proc corr data=Setosa sscp cov plots;
   var  sepallength sepalwidth;
   with petallength petalwidth;
run;

ods graphics off;
ods html close;
```

The CORR procedure displays univariate statistics for variables in the VAR and WITH statements.

Output 1.2.1. Simple Statistics

```
                        Fisher (1936) Iris Setosa Data

                              The CORR Procedure

                  2 With Variables:    PetalLength PetalWidth
                  2      Variables:    SepalLength SepalWidth


                              Simple Statistics

       Variable             N          Mean       Std Dev           Sum

       PetalLength         49      14.71429       1.62019     721.00000
       PetalWidth          48       2.52083       1.03121     121.00000
       SepalLength         50      50.06000       3.52490          2503
       SepalWidth          50      34.28000       3.79064          1714

                              Simple Statistics

         Variable          Minimum       Maximum    Label

         PetalLength      11.00000      19.00000    Petal Length in mm.
         PetalWidth        1.00000       6.00000    Petal Width in mm.
         SepalLength      43.00000      58.00000    Sepal Length in mm.
         SepalWidth       23.00000      44.00000    Sepal Width in mm.
```

When the WITH statement is specified together with the VAR statement, the CORR procedure produces rectangular matrices for statistics such as covariances and correlations. The matrix rows correspond to the WITH variables (PetalLength and PetalWidth) while the matrix columns correspond to the VAR variables (SepalLength and SepalWidth). The CORR procedure uses the WITH variable labels to label the matrix rows.

The SSCP option requests a table of the uncorrected sum-of-squares and crossproducts matrix, and the COV option requests a table of the covariance matrix. The SSCP and COV options also produce a table of the Pearson correlations.

The sum-of-squares and crossproducts statistics for each pair of variables are computed by using observations with nonmissing row and column variable values. The "Sums of Squares and Crossproducts" table shown in Output 1.2.2 displays the crossproduct, sum of squares for the row variable, and sum of squares for the column variable for each pair of variables.

Output 1.2.2. Sum-of-squares and Crossproducts

```
                     Fisher (1936) Iris Setosa Data

                  Sums of Squares and Crossproducts
                    SSCP / Row Var SS / Col Var SS

                               SepalLength        SepalWidth

          PetalLength          36214.00000       24756.00000
          Petal Length in mm.  10735.00000       10735.00000
                               123793.0000        58164.0000

          PetalWidth            6113.00000        4191.00000
          Petal Width in mm.     355.00000         355.00000
                               121356.0000        56879.0000
```

The variances are computed by using observations with nonmissing row and column variable values. The "Variances and Covariances" table shown in Output 1.2.3 displays the covariance, variance for the row variable, variance for the column variable, and the associated degrees of freedom for each pair of variables.

Output 1.2.3. Variances and Covariances

```
                     Fisher (1936) Iris Setosa Data

                       Variances and Covariances
           Covariance / Row Var Variance / Col Var Variance / DF

                               SepalLength        SepalWidth

          PetalLength          1.270833333       1.363095238
          Petal Length in mm.  2.625000000       2.625000000
                               12.33333333       14.60544218
                                        48                48

          PetalWidth           0.911347518       1.048315603
          Petal Width in mm.   1.063386525       1.063386525
                               11.80141844       13.62721631
                                        47                47
```

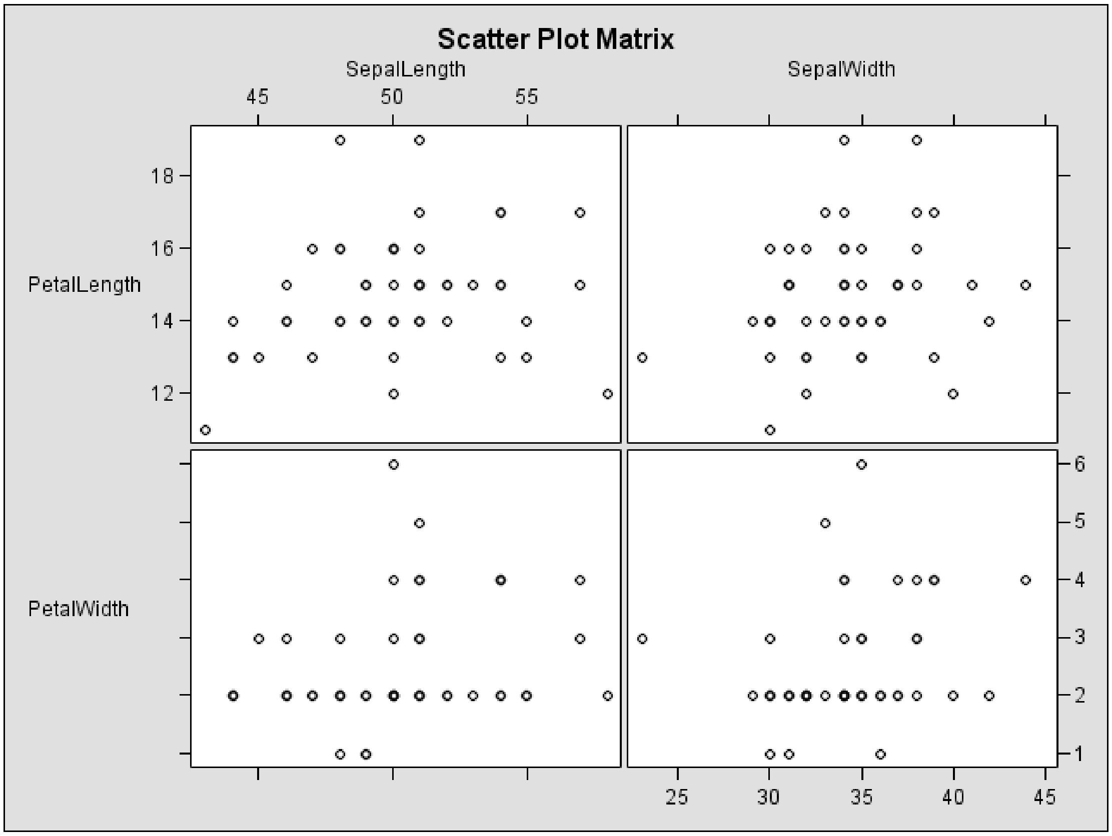
Output 1.2.4. Pearson Correlation Coefficients

```
                    Fisher (1936) Iris Setosa Data

                  Pearson Correlation Coefficients
                     Prob > |r| under H0: Rho=0
                       Number of Observations

                                   Sepal         Sepal
                                  Length         Width

        PetalLength              0.22335       0.22014
        Petal Length in mm.       0.1229        0.1285
                                      49            49

        PetalWidth               0.25726       0.27539
        Petal Width in mm.        0.0775        0.0582
                                      48            48
```

When there are missing values in the analysis variables, the “Pearson Correlation Coefficients” table shown in Output 1.2.4 displays the correlation, the p-value under the null hypothesis of zero correlation, and the number of observations for each pair of variables. Only the correlation between PetalWidth and SepalLength and the correlation between PetalWidth and SepalWidth are slightly positive.

The experimental PLOTS option displays a rectangular scatter plot matrix for the two sets of variables. The VAR variables SepalLength and SepalWidth are listed across the top of the matrix, and the WITH variables PetalLength and PetalWidth are listed down the side of the matrix. As measured in Output 1.2.4, the plot for PetalWidth and SepalLength and the plot for PetalWidth and SepalWidth show slight positive correlations.

Output 1.2.5. Rectangular Matrix Plot (Experimental)



This display is requested by specifying both the ODS GRAPHICS statement and the PLOTS option. For general information about ODS graphics, refer to Chapter 15, "Statistical Graphics Using ODS" (*SAS/STAT User's Guide*). For specific information about the graphics available in the CORR procedure, see the section "ODS Graphics" on page 31.

Example 1.3. Analysis Using Fisher's z Transformation

The following statements request Pearson correlation statistics using Fisher's z transformation for the data set Fitness.

```
proc corr data=Fitness nosimple fisher;
   var weight oxygen runtime;
run;
```

The NOSIMPLE option suppresses the table of descriptive statistics. The "Pearson Correlation Coefficients" table is displayed by default.

Output 1.3.1. Sample Correlations

```
                     Fisher (1936) Iris Setosa Data

                           The CORR Procedure

                    Pearson Correlation Coefficients
                       Prob > |r| under H0: Rho=0
                         Number of Observations

                         Weight          Oxygen        RunTime

          Weight        1.00000        -0.15358        0.20072
                                         0.4264         0.2965
                             31              29             29

          Oxygen       -0.15358         1.00000       -0.86843
                         0.4264                         <.0001
                             29              29             28

          RunTime       0.20072        -0.86843        1.00000
                         0.2965          <.0001
                             29              28             29
```

The FISHER option requests correlation statistics using Fisher's z transformation, which are shown in Output 1.3.2.

Output 1.3.2. Correlation Statistics Using Fisher's z Transformation

```
          Pearson Correlation Statistics (Fisher's z Transformation)

          With                      Sample                      Bias  Correlation
Variable  Variable          N  Correlation  Fisher's z  Adjustment     Estimate

Weight    Oxygen           29     -0.15358    -0.15480    -0.00274     -0.15090
Weight    RunTime          29      0.20072     0.20348     0.00358      0.19727
Oxygen    RunTime          28     -0.86843    -1.32665    -0.01608     -0.86442

          Pearson Correlation Statistics (Fisher's z Transformation)

                   With                                     p Value for
         Variable  Variable     95% Confidence Limits          H0:Rho=0

         Weight    Oxygen       -0.490289      0.228229          0.4299
         Weight    RunTime      -0.182422      0.525765          0.2995
         Oxygen    RunTime      -0.935728     -0.725221          <.0001
```

See the section "Fisher's z Transformation" on page 21 for details on Fisher's z transformation.

The following statements request one-sided hypothesis tests and confidence limits for the correlation using Fisher's z transformation.

```
proc corr data=Fitness nosimple nocorr fisher (type=lower);
   var weight oxygen runtime;
run;
```

The NOSIMPLE option suppresses the "Simple Statistics" table, and the NOCORR option suppresses the "Pearson Correlation Coefficients" table.

Output 1.3.3. One-sided Correlation Analysis Using Fisher's z Transformation

The CORR Procedure

Pearson Correlation Statistics (Fisher's z Transformation)

Variable	With Variable	N	Sample Correlation	Fisher's z	Bias Adjustment	Correlation Estimate
Weight	Oxygen	29	-0.15358	-0.15480	-0.00274	-0.15090
Weight	RunTime	29	0.20072	0.20348	0.00358	0.19727
Oxygen	RunTime	28	-0.86843	-1.32665	-0.01608	-0.86442

Pearson Correlation Statistics (Fisher's z Transformation)

Variable	With Variable	Lower 95% CL	p Value for H0:Rho<=0
Weight	Oxygen	-0.441943	0.7850
Weight	RunTime	-0.122077	0.1497
Oxygen	RunTime	-0.927408	1.0000

The TYPE=LOWER option requests a lower confidence limit and a p-value for the test of the one-sided hypothesis $H_0\colon \rho \leq 0$ against the alternative hypothesis $H_1\colon \rho > 0$. Here Fisher's z, the bias adjustment, and the estimate of the correlation are the same as for the two-sided alternative. However, because TYPE=LOWER is specified, only a lower confidence limit is computed for each correlation, and one-sided p-values are computed.

Example 1.4. Applications of Fisher's z Transformation

This example illustrates some applications of Fisher's z transformation. For details, see the section "Fisher's z Transformation" on page 21.

The following statements simulate independent samples of variables X and Y from a bivariate normal distribution. The first batch of 150 observations is sampled using a known correlation of 0.3, the second batch of 150 observations is sampled using a known correlation of 0.25, and the third batch of 100 observations is sampled using a known correlation of 0.3.

```
data Sim (drop=i);
do i=1 to 400;
  X = rannor(135791);
  Batch = 1 + (i>150) + (i>300);
  if Batch = 1 then Y = 0.3*X + 0.9*rannor(246791);
  if Batch = 2 then Y = 0.25*X + sqrt(.8375)*rannor(246791);
  if Batch = 3 then Y = 0.3*X + 0.9*rannor(246791);
  output;
end;
run;
```

This data set will be used to illustrate the following applications of Fisher's z transformation:

- Testing whether a population correlation is equal to a given value
- Testing for equality of two population correlations
- Combining correlation estimates from different samples

See the section "Fisher's z Transformation" on page 21.

Testing Whether a Population Correlation Is Equal to a Given Value ρ_0

You can use the following statements to test the null hypothesis $H_0\colon \rho = 0.5$ against a two-sided alternative $H_1\colon \rho \neq 0.5$.

```
ods select FisherPearsonCorr;
title 'Analysis for Batch 1';
proc corr data=Sim (where=(Batch=1)) fisher(rho0=.5);
   var X Y;
run;
```

The test is requested with the option FISHER(RHO0=0.5). The results, which are based on Fisher's transformation, are shown in Output 1.4.1.

Output 1.4.1. Fisher's Test for $H_0\colon \rho = \rho_0$

```
                          Analysis for Batch 1

                          The CORR Procedure

          Pearson Correlation Statistics (Fisher's z Transformation)

         With                     Sample                     Bias  Correlation
Variable  Variable          N  Correlation  Fisher's z  Adjustment     Estimate

X         Y               150      0.22081     0.22451   0.0007410      0.22011

         Pearson Correlation Statistics (Fisher's z Transformation)

            With                                     ------H0:Rho=Rho0-----
   Variable  Variable     95% Confidence Limits           Rho0      p Value

   X         Y             0.062034      0.367409      0.50000      <.0001
```

The null hypothesis is rejected since the p-value is less than 0.0001.

Testing for Equality of Two Population Correlations

You can use the following statements to test for equality of two population correlations, ρ_1 and ρ_2. Here, the null hypothesis $H_0\colon \rho_1 = \rho_2$ is tested against the alternative $H_1\colon \rho_1 \neq \rho_2$.

```
ods select FisherPearsonCorr;
ods output FisherPearsonCorr=SimCorr;
title 'Testing Equality of Population Correlations';
proc corr data=Sim (where=(Batch=1 or Batch=2)) fisher;
   var X Y;
   by Batch;
run;
```

The ODS SELECT statement restricts the output from PROC CORR to the “FisherPearsonCorr” table, which is shown in Output 1.4.2; see the section “ODS Table Names” on page 30. The output data set SimCorr contains Fisher’s z statistics for both batches.

Output 1.4.2. Fisher’s Correlation Statistics

```
                  Testing Equality of Population Correlations

----------------------------------- Batch=1 -----------------------------------

                               The CORR Procedure

          Pearson Correlation Statistics (Fisher's z Transformation)

          With                    Sample                    Bias  Correlation
Variable  Variable         N  Correlation  Fisher's z  Adjustment     Estimate

X         Y              150      0.22081     0.22451   0.0007410      0.22011

          Pearson Correlation Statistics (Fisher's z Transformation)

                   With                                        p Value for
         Variable  Variable     95% Confidence Limits             H0:Rho=0

         X         Y             0.062034      0.367409             0.0065


                  Testing Equality of Population Correlations

----------------------------------- Batch=2 -----------------------------------

                               The CORR Procedure

          Pearson Correlation Statistics (Fisher's z Transformation)

          With                    Sample                    Bias  Correlation
Variable  Variable         N  Correlation  Fisher's z  Adjustment     Estimate

X         Y              150      0.33694     0.35064     0.00113      0.33594

          Pearson Correlation Statistics (Fisher's z Transformation)

                   With                                        p Value for
         Variable  Variable     95% Confidence Limits             H0:Rho=0

         X         Y             0.185676      0.470853             <.0001
```

The p-value for testing H_0 is derived by treating the difference $z_1 - z_2$ as a normal random variable with mean zero and variance $1/(n_1-3)+1/(n_2-3)$, where z_1 and z_2 are Fisher's z transformation of the sample correlations r_1 and r_2, respectively, and where n_1 and n_2 are the corresponding sample sizes.

The following statements compute the p-value shown in Output 1.4.3.

```
data SimTest (drop=Batch);
   merge SimCorr (where=(Batch=1) keep=Nobs ZVal Batch
                  rename=(Nobs=n1 ZVal=z1))
         SimCorr (where=(Batch=2) keep=Nobs ZVal Batch
                  rename=(Nobs=n2 ZVal=z2));
   variance = 1/(n1-3) + 1/(n2-3);
   z = (z1 - z2) / sqrt( variance );
   pval = probnorm(z);
   if (pval > 0.5) then pval = 1 - pval;
   pval = 2*pval;
run;

proc print data=SimTest noobs;
run;
```

Output 1.4.3. Test of Equality of Observed Correlations

n1	z1	n2	z2	variance	z	pval
150	0.22451	150	0.35064	0.013605	-1.08135	0.27954

In Output 1.4.3, the p-value of 0.2795 does not provide evidence to reject the null hypothesis that $\rho_1 = \rho_2$. The sample sizes $n_1 = 150$ and $n_2 = 150$ are not large enough to detect the difference $\rho_1 - \rho_2 = 0.05$ at a significance level of $\alpha = 0.05$.

Combining Correlation Estimates from Different Samples

Assume that sample correlations r_1 and r_2 are computed from two independent samples of n_1 and n_2 observations, respectively. A combined correlation estimate is given by $\bar{r} = \tanh(\bar{z})$, where $\bar{z}$ is the weighted average of the z-transformations of r_1 and r_2:

$$\bar{z} = \frac{(n_1 - 3)z_1 + (n_2 - 3)z_2}{n_1 + n_2 - 6}$$

The following statements compute a combined estimate of ρ using Batch 1 and Batch 3:

```
ods output FisherPearsonCorr=SimCorr2;
proc corr data=Sim (where=(Batch=1 or Batch=3)) fisher noprint;
   var X Y;
   by Batch;
run;

data SimComb (drop=Batch);
   merge SimCorr2 (where=(Batch=1) keep=Nobs ZVal Batch
                   rename=(Nobs=n1 ZVal=z1))
         SimCorr2 (where=(Batch=3) keep=Nobs ZVal Batch
                   rename=(Nobs=n2 ZVal=z2));
   z = ((n1-3)*z1 + (n2-3)*z2) / (n1+n2-6);
   corr = tanh(z);
   var = 1/(n1+n2-6);
   lcl = corr - probit(0.975)*sqrt(var);
   ucl = corr + probit(0.975)*sqrt(var);
run;

proc print data=SimComb noobs;
   var n1 z1 n2 z2 corr lcl ucl;
run;
```

Output 1.4.4 displays the combined estimate of ρ.

Output 1.4.4. Combined Correlation Estimate

n1	z1	n2	z2	corr	lcl	ucl
150	0.22451	100	0.23929	0.22640	0.10092	0.35187

Thus, a correlation estimate from the combined samples is $r = 0.23$. The 95% confidence interval displayed in Output 1.4.4 is $(0.10, 0.35)$ using the variance of the combined estimate. Note that this interval contains the population correlation 0.3. See the section "Applications of Fisher's z Transformation" on page 23.

Example 1.5. Computing Cronbach's Coefficient Alpha

The following statements create the data set Fish1 from the Fish data set used in Chapter 67, "The STEPDISC Procedure." The cubic root of the weight (Weight3) is computed as a one-dimensional measure of the size of a fish.

```
*------------------- Fish Measurement Data ----------------------*
| The data set contains 35 fish from the species Bream caught in |
| Finland's lake Laengelmavesi with the following measurements:  |
| Weight   (in grams)                                             |
| Length3  (length from the nose to the end of its tail, in cm)  |
| HtPct    (max height, as percentage of Length3)                |
| WidthPct (max width,  as percentage of Length3)                |
*----------------------------------------------------------------*;
```

```
data Fish1 (drop=HtPct WidthPct);
   title 'Fish Measurement Data';
   input Weight Length3 HtPct WidthPct @@;
   Weight3= Weight**(1/3);
   Height=HtPct*Length3/100;
   Width=WidthPct*Length3/100;
   datalines;
242.0 30.0 38.4 13.4     290.0 31.2 40.0 13.8
340.0 31.1 39.8 15.1     363.0 33.5 38.0 13.3
430.0 34.0 36.6 15.1     450.0 34.7 39.2 14.2
500.0 34.5 41.1 15.3     390.0 35.0 36.2 13.4
450.0 35.1 39.9 13.8     500.0 36.2 39.3 13.7
475.0 36.2 39.4 14.1     500.0 36.2 39.7 13.3
500.0 36.4 37.8 12.0         .  37.3 37.3 13.6
600.0 37.2 40.2 13.9     600.0 37.2 41.5 15.0
700.0 38.3 38.8 13.8     700.0 38.5 38.8 13.5
610.0 38.6 40.5 13.3     650.0 38.7 37.4 14.8
575.0 39.5 38.3 14.1     685.0 39.2 40.8 13.7
620.0 39.7 39.1 13.3     680.0 40.6 38.1 15.1
700.0 40.5 40.1 13.8     725.0 40.9 40.0 14.8
720.0 40.6 40.3 15.0     714.0 41.5 39.8 14.1
850.0 41.6 40.6 14.9    1000.0 42.6 44.5 15.5
920.0 44.1 40.9 14.3     955.0 44.0 41.1 14.3
925.0 45.3 41.4 14.9     975.0 45.9 40.6 14.7
950.0 46.5 37.9 13.7
;
```

The following statements request a correlation analysis and compute Cronbach's coefficient alpha for the variables Weight3, Length3, Height, and Width.

```
ods html;
ods graphics on;

title 'Fish Measurement Data';
proc corr data=fish1 nomiss alpha plots;
   var Weight3 Length3 Height Width;
 run;

ods graphics off;
ods html close;
```

The NOMISS option excludes observations with missing values, and the PLOTS option requests a symmetric scatter plot matrix for the analysis variables.

By default, the CORR procedure displays descriptive statistics for each variable, as shown in Output 1.5.1.

Output 1.5.1. Simple Statistics

```
                              Fish Measurement Data

                               The CORR Procedure

                4  Variables:    Weight3  Length3  Height   Width


                               Simple Statistics

Variable         N        Mean     Std Dev         Sum     Minimum     Maximum

Weight3         34     8.44751     0.97574   287.21524     6.23168    10.00000
Length3         34    38.38529     4.21628        1305    30.00000    46.50000
Height          34    15.22057     1.98159   517.49950    11.52000    18.95700
Width           34     5.43805     0.72967   184.89370     4.02000     6.74970
```

Since the NOMISS option is specified, the same set of 34 observations is used to compute the correlation for each pair of variables. The correlations are shown in Output 1.5.2.

Output 1.5.2. Pearson Correlation Coefficients

```
                              Fish Measurement Data

                     Pearson Correlation Coefficients, N = 34
                            Prob > |r| under H0: Rho=0

                 Weight3       Length3        Height         Width

Weight3          1.00000       0.96523       0.96261       0.92789
                                <.0001        <.0001        <.0001

Length3          0.96523       1.00000       0.95492       0.92171
                  <.0001                      <.0001        <.0001

Height           0.96261       0.95492       1.00000       0.92632
                  <.0001        <.0001                      <.0001

Width            0.92789       0.92171       0.92632       1.00000
                  <.0001        <.0001        <.0001
```

Since the data set contains only one species of fish, all the variables are highly correlated. This is evidenced in the scatter plot matrix for the analysis variables, which is shown in Output 1.7.3, created in Example 1.7.

Positive correlation is needed for the alpha coefficient because variables measure a common entity.

With the ALPHA option, the CORR procedure computes Cronbach's coefficient alpha, which is a lower bound for the reliability coefficient for the raw variables and the standardized variables.

Output 1.5.3. Cronbach's Coefficient Alpha

```
                    Fish Measurement Data

                  Cronbach Coefficient Alpha

                 Variables              Alpha
                 ----------------------------
                 Raw                 0.822134
                 Standardized        0.985145
```

Because the variances of some variables vary widely, you should use the standardized score to estimate reliability. The overall standardized Cronbach's coefficient alpha of 0.985145 provides an acceptable lower bound for the reliability coefficient. This is much greater than the suggested value of 0.70 given by Nunnally and Bernstein (1994).

Output 1.5.4. Cronbach's Coefficient Alpha with Deleted Variables

```
                          Fish Measurement Data

            Cronbach Coefficient Alpha with Deleted Variable

                    Raw Variables              Standardized Variables

Deleted       Correlation                      Correlation
Variable       with Total           Alpha       with Total           Alpha
-------------------------------------------------------------------------
Weight3          0.975379        0.783365         0.973464        0.977103
Length3          0.967602        0.881987         0.967177        0.978783
Height           0.964715        0.655098         0.968079        0.978542
Width            0.934635        0.824069         0.937599        0.986626
```

The standardized alpha coefficient provides information on how each variable reflects the reliability of the scale with standardized variables. If the standardized alpha decreases after removing a variable from the construct, then this variable is strongly correlated with other variables in the scale. On the other hand, if the standardized alpha increases after removing a variable from the construct, then removing this variable from the scale makes the construct more reliable. The "Cronbach Coefficient Alpha with Deleted Variables" table in Output 1.5.4 does not show significant increase or decrease for the standardized alpha coefficients. See the section "Cronbach's Coefficient Alpha" on page 24 for more information regarding constructs and Cronbach's alpha.

Example 1.6. Saving Correlations in an Output Data Set

The following statements compute Pearson correlations and covariances.

```
title 'Correlations for a Fitness and Exercise Study';
proc corr data=Fitness nomiss outp=CorrOutp;
   var weight oxygen runtime;
run;
```

The NOMISS option excludes observations with missing values of the VAR statement variables from the analysis. The NOSIMPLE option suppresses the display of descriptive statistics, and the OUTP= option creates an output data set named CorrOutp that contains the Pearson correlation statistics. Since the NOMISS option is specified, the same set of 28 observations is used to compute the correlation for each pair of variables.

Output 1.6.1. Pearson Correlation Coefficients

```
                  Correlations for a Fitness and Exercise Study

                              The CORR Procedure

                   Pearson Correlation Coefficients, N = 28
                          Prob > |r| under H0: Rho=0

                              Weight          Oxygen         RunTime

            Weight           1.00000        -0.18419         0.19505
                                              0.3481          0.3199

            Oxygen          -0.18419         1.00000        -0.86843
                              0.3481                          <.0001

            RunTime          0.19505        -0.86843         1.00000
                              0.3199          <.0001
```

The following statements display the output data set, which is shown in Output 1.6.2.

```
title 'Output Data Set from PROC CORR';
proc print data=CorrOutp noobs;
run;
```

Output 1.6.2. OUTP= Data Set with Pearson Correlations

```
                          Output Data Set from PROC CORR

             _TYPE_     _NAME_        Weight       Oxygen      RunTime

              MEAN                   77.2168      47.1327      10.6954
              STD                     8.4495       5.5535       1.4127
              N                      28.0000      28.0000      28.0000
              CORR      Weight        1.0000      -0.1842       0.1950
              CORR      Oxygen       -0.1842       1.0000      -0.8684
              CORR      RunTime       0.1950      -0.8684       1.0000
```

The output data set has the default type CORR and can be used as an input data set for regression or other statistical procedures. For example, the following statements request a regression analysis using CorrOutp, without reading the original data in the REG procedure:

```
title 'Input Type CORR Data Set from PROC REG';
proc reg data=CorrOutp;
   model runtime= weight oxygen;
run;
```

The preceding statements generate the same results as the following statements:

```
proc reg data=Fitness nomiss;
   model runtime= weight oxygen;
run;
```

Example 1.7. Creating Scatter Plots

The following statements request a correlation analysis and a scatter plot matrix for the variables in the data set Fish1, which was created in Example 1.5. This data set contains 35 observations, one of which contains a missing value for the variable Weight3.

```
ods html;
ods graphics on;

title 'Fish Measurement Data';
proc corr data=fish1 nomiss plots=matrix;
   var Height Width Length3 Weight3;
 run;

ods graphics off;
ods html close;
```

By default, the CORR procedure displays descriptive statistics for the VAR statement variables, which are shown in Output 1.7.1.

Output 1.7.1. Simple Statistics

```
                          Fish Measurement Data

                           The CORR Procedure

             4  Variables:    Height    Width    Length3  Weight3


                           Simple Statistics

Variable           N         Mean      Std Dev          Sum      Minimum      Maximum

Height            34     15.22057      1.98159    517.49950     11.52000     18.95700
Width             34      5.43805      0.72967    184.89370      4.02000      6.74970
Length3           34     38.38529      4.21628         1305     30.00000     46.50000
Weight3           34      8.44751      0.97574    287.21524      6.23168     10.00000
```

Since the NOMISS option is specified, the same set of 34 observations is used to compute the correlation for each pair of variables. The correlations are shown in Output 1.7.2.

Output 1.7.2. Pearson Correlation Coefficients

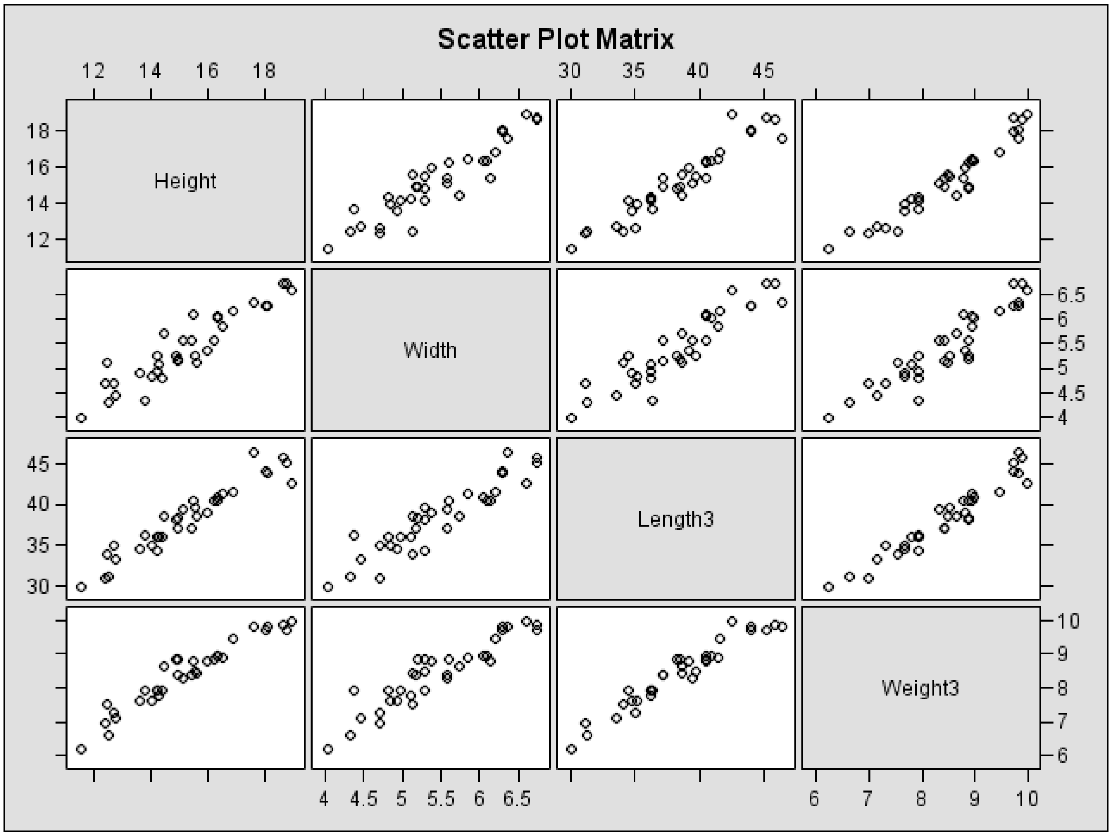
```
                         Fish Measurement Data

                Pearson Correlation Coefficients, N = 34
                       Prob > |r| under H0: Rho=0

                  Height         Width       Length3       Weight3

Height           1.00000       0.92632       0.95492       0.96261
                                <.0001        <.0001        <.0001

Width            0.92632       1.00000       0.92171       0.92789
                  <.0001                      <.0001        <.0001

Length3          0.95492       0.92171       1.00000       0.96523
                  <.0001        <.0001                      <.0001

Weight3          0.96261       0.92789       0.96523       1.00000
                  <.0001        <.0001        <.0001
```

The variables are highly correlated. For example, the correlation between Height and Width is 0.92632.

The experimental PLOTS=MATRIX option requests a scatter plot matrix for the VAR statement variables, which is shown in Output 1.7.3.

In order to create this display, you must specify the experimental ODS GRAPHICS statement in addition to the PLOTS=MATRIX option. For general information about ODS graphics, refer to Chapter 15, “Statistical Graphics Using ODS” (*SAS/STAT User’s Guide*). For specific information about ODS graphics available in the CORR procedure, see the section “ODS Graphics” on page 31.

Output 1.7.3. Scatter Plot Matrix (Experimental)

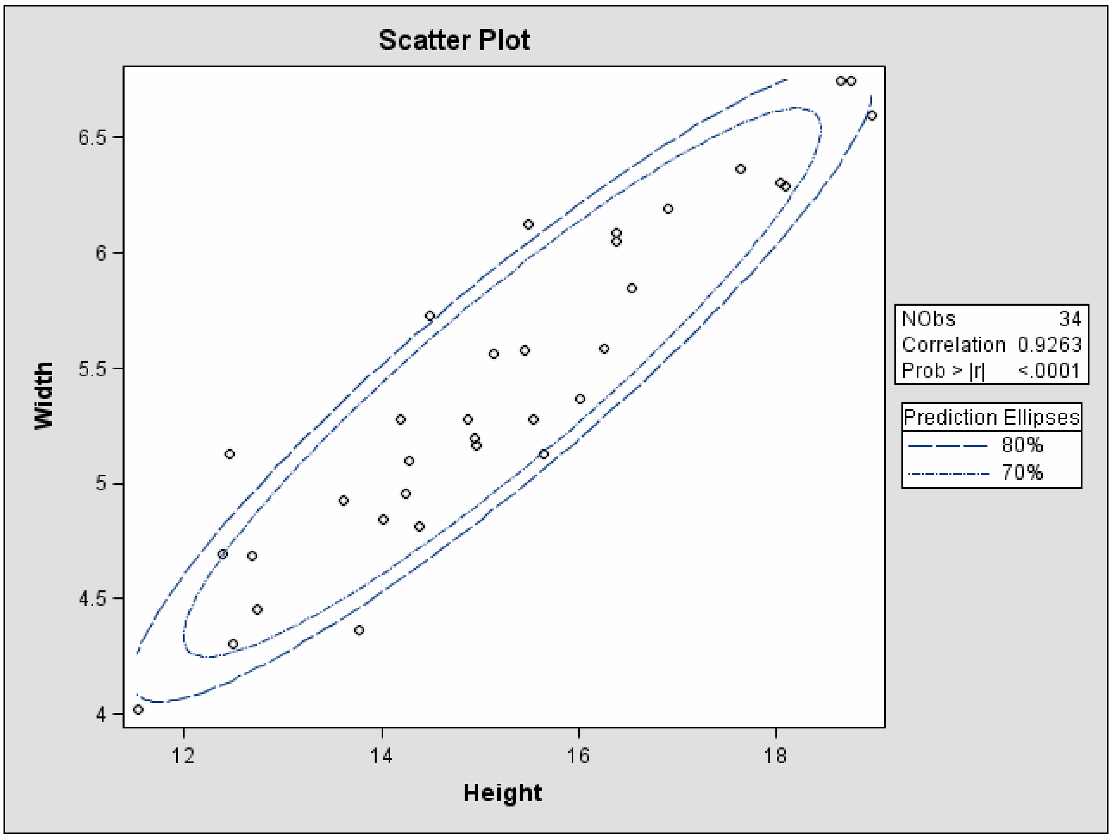


To explore the correlation between Height and Width, the following statements request a scatter plot with prediction ellipses for the two variables, which is shown in Output 1.7.4. A prediction ellipse is a region for predicting a new observation from the population, assuming bivariate normality. It also approximates a region containing a specified percentage of the population.

```
ods html;
ods graphics on;

proc corr data=fish1 nomiss noprint
          plots=scatter(nmaxvar=2 alpha=.20 .30);
   var Height Width Length3 Weight3;
 run;

ods graphics off;
ods html close;
```

The NOMISS option is specified with the original VAR statement to ensure that the same set of 34 observations is used for this analysis. The experimental PLOTS=SCATTER(NMAXVAR=2) option requests a scatter plot for the first two variables in the VAR list. The ALPHA= suboption requests 80% and 70% prediction ellipses.

Output 1.7.4. Scatter Plot with Prediction Ellipses (Experimental)

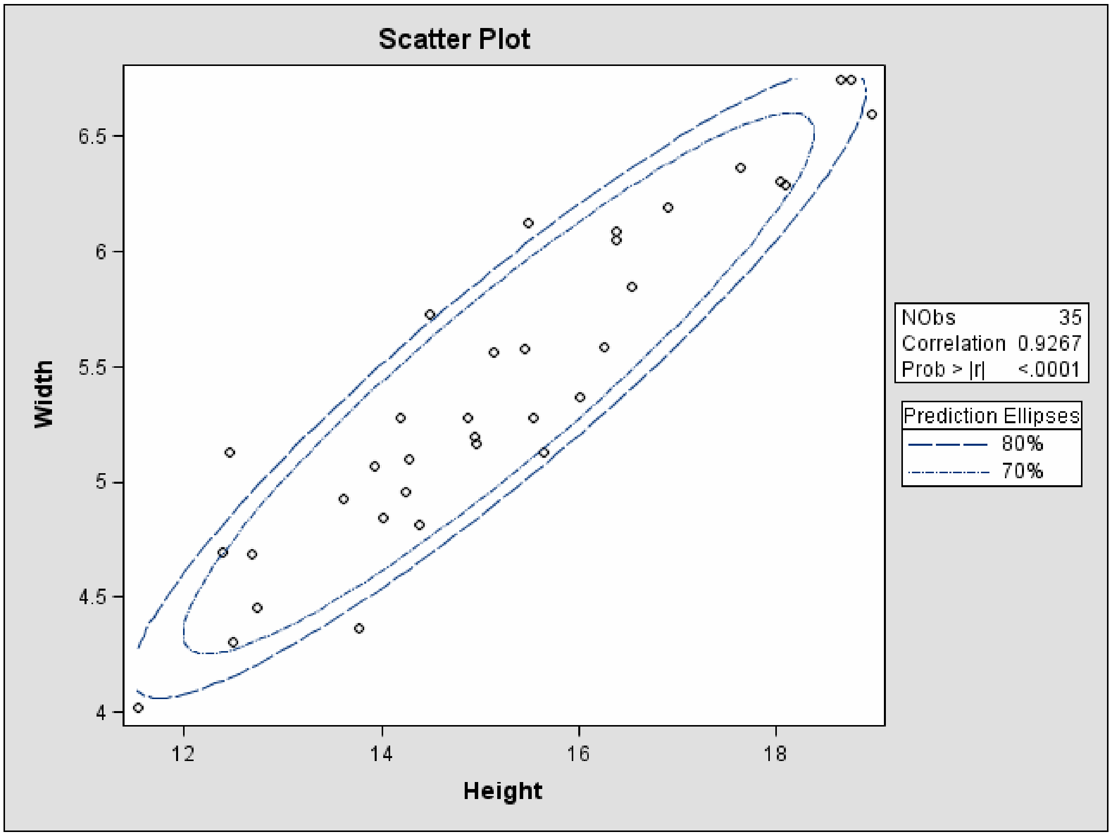


The prediction ellipse is centered at the means $(\bar{x}, \bar{y})$. For further details, see the section "Confidence and Prediction Ellipses" on page 33.

Note that the following statements can also be used to create a scatter plot for Height and Width:

```
ods html;
ods graphics on;

proc corr data=fish1 noprint
          plots=scatter(alpha=.20 .30);
   var Height Width;
 run;

ods graphics off;
ods html close;
```

Output 1.7.5. Scatter Plot with Prediction Ellipses (Experimental)



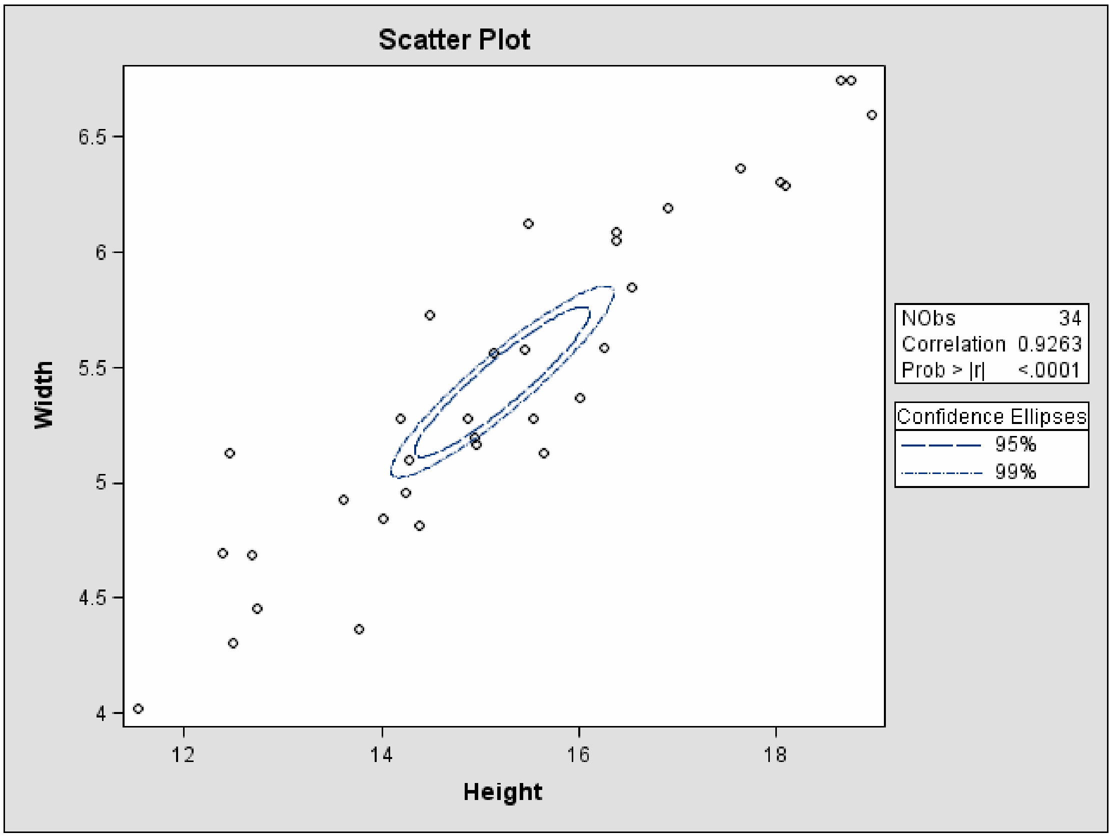
Output 1.7.5 includes the point $(13.9, 5.1)$, which was excluded from Output 1.7.4 because the observation had a missing value for Weight3. The prediction ellipses in Output 1.7.5 also reflect the inclusion of this observation.

The following statements request a scatter plot with confidence ellipses for the mean, which is shown in Output 1.7.6:

```
ods html;
ods graphics on;

title 'Fish Measurement Data';
proc corr data=fish1 nomiss noprint
          plots=scatter(ellipse=mean nmaxvar=2 alpha=.05 .01);
   var Height Width Length3 Weight3;
 run;

ods graphics off;
ods html close;
```

The experimental PLOTS=SCATTER option requests scatter plots for all the variables in the VAR statement, and the NMAXVAR=2 suboption restricts the number of plots created to the first two variables in the VAR statement. The ELLIPSE=MEAN and ALPHA= suboptions request 95% and 99% confidence ellipses for the mean.

Output 1.7.6. Scatter Plot with Confidence Ellipses (Experimental)



The confidence ellipse for the mean is centered at the means $(\bar{x}, \bar{y})$. For further details, see the section “Confidence and Prediction Ellipses” on page 33.

Example 1.8. Computing Partial Correlations

A partial correlation measures the strength of the linear relationship between two variables, while adjusting for the effect of other variables.

The following statements request a partial correlation analysis of variables Height and Width while adjusting for the variables Length3 and Weight. The latter variables, which are said to be “partialled out” of the analysis, are specified with the PARTIAL statement.

```
ods html;
ods graphics on;

title 'Fish Measurement Data';
proc corr data=fish1 plots=scatter(alpha=.20 .30);
   var Height Width;
   partial Length3 Weight3;
 run;

ods graphics off;
ods html close;
```

By default, the CORR procedure displays descriptive statistics for all the variables and the partial variance and partial standard deviation for the VAR statement variables, as shown in Output 1.8.1.

Output 1.8.1. Descriptive Statistics

```
                          Fish Measurement Data

                           The CORR Procedure

                  2 Partial Variables:    Length3   Weight3
                  2         Variables:    Height    Width


                           Simple Statistics

Variable          N        Mean     Std Dev         Sum     Minimum     Maximum

Length3          34    38.38529     4.21628        1305    30.00000    46.50000
Weight3          34     8.44751     0.97574   287.21524     6.23168    10.00000
Height           34    15.22057     1.98159   517.49950    11.52000    18.95700
Width            34     5.43805     0.72967   184.89370     4.02000     6.74970

                           Simple Statistics

                                 Partial       Partial
                      Variable   Variance      Std Dev

                      Length3
                      Weight3
                      Height      0.26607      0.51582
                      Width       0.07315      0.27047
```

When a PARTIAL statement is specified, observations with missing values are excluded from the analysis. The partial correlations for the VAR statement variables are shown in Output 1.8.2.

The partial correlation between the variables Height and Width is 0.25692, which is much less than the unpartialled correlation, 0.92632. The p-value for the partial correlation is 0.1558.

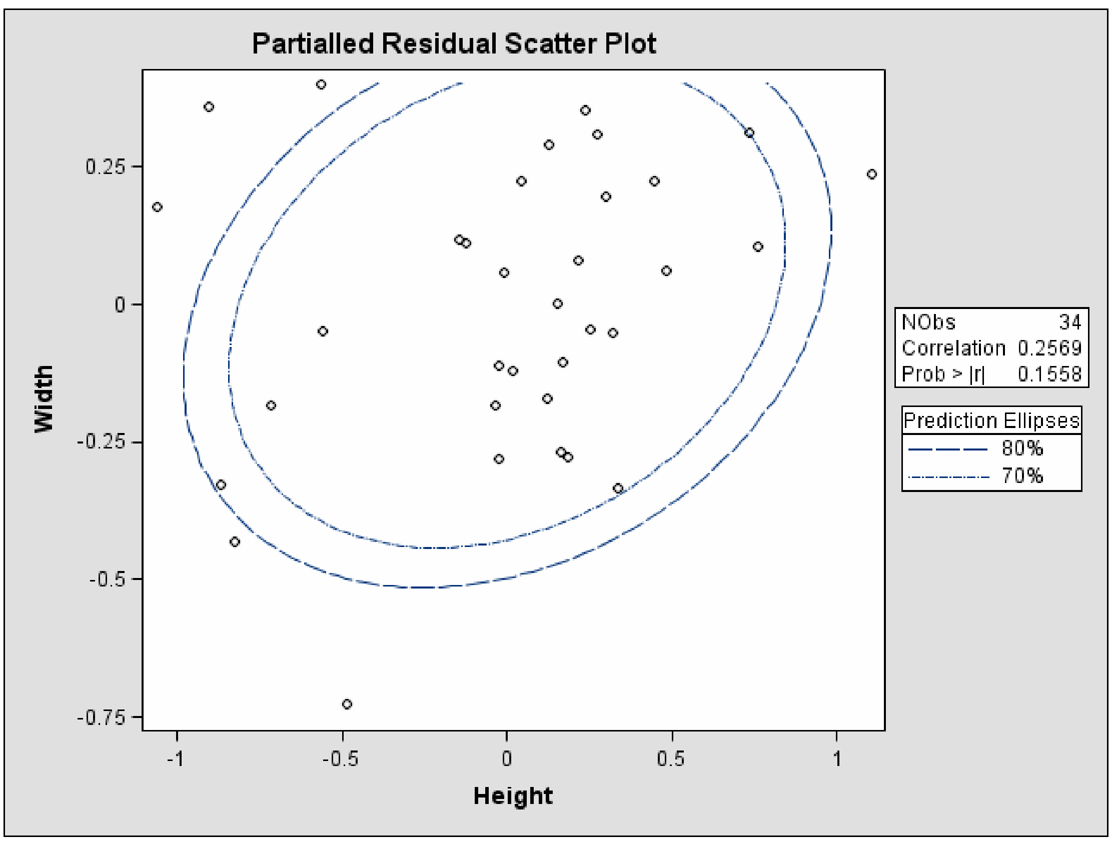
The PLOTS=SCATTER option requests a scatter plot of the residuals for the variables Height and Width after controlling for the effect of variables Length3 and Weight.

The ALPHA= suboption requests 80% and 70% prediction ellipses. The scatter plot is shown in Output 1.8.3.

Output 1.8.2. Pearson Partial Correlation Coefficients

```
                    Fish Measurement Data

     Pearson Partial Correlation Coefficients, N = 34
            Prob > |r| under H0: Partial Rho=0

                         Height          Width

            Height      1.00000        0.25692
                                        0.1558

            Width       0.25692        1.00000
                         0.1558
```

Output 1.8.3. Partial Residual Scatter Plot (Experimental)



In Output 1.8.3, a standard deviation of Height has roughly the same length on the X-axis as a standard deviation of Width on the Y-axis. The major axis length is not significantly larger than the minor axis length, indicating a weak partial correlation between Height and Width.

References

Anderson, T.W. (1984), *An Introduction to Multivariate Statistical Analysis,* Second Edition, New York: John Wiley & Sons.

Blum, J.R., Kiefer, J., and Rosenblatt, M. (1961), “Distribution Free Tests of Independence Based on the Sample Distribution Function,” *Annals of Mathematical Statistics* , 32, 485–498.

Conover, W.J. (1998), *Practical Nonparametric Statistics*, Third Edition, New York: John Wiley & Sons, Inc.

Cronbach, L.J. (1951), “Coefficient Alpha and the Internal Structure of Tests,” *Psychometrika*, 16, 297–334.

Fisher, R.A. (1915), “Frequency Distribution of the Values of the Correlation Coefficient in Samples from an Indefinitely Large Population,” *Biometrika*, 10, 507–521.

Fisher, R.A. (1921), “On the “Probable Error” of a Coefficient of Correlation Deduced from a Small Sample,” *Metron*, 1, 3–32.

Fisher, R.A. (1936), “The Use of Multiple Measurements in Taxonomic Problems,” *Annals of Eugenics*, 7, 179–188.

Fisher, R.A. (1970), *Statistical Methods for Research Workers,* Fourteenth Edition, Davien, CT: Hafner Publishing Company.

Hoeffding, W. (1948), “A Non-Parametric Test of Independence,” *Annals of Mathematical Statistics* , 19, 546–557.

Hollander, M. and Wolfe, D. (1999), *Nonparametric Statistical Methods*, Second Edition, New York: John Wiley & Sons, Inc.

Keeping, E.S. (1962), *Introduction to Statistical Inference,* New York: D. Van Nostrand Cimpany, Inc.

Knight, W.E. (1966), “A Computer Method for Calculating Kendall’s Tau with Ungrouped Data,” *Journal of the American Statistical Association* , 61, 436–439.

Moore, D.S. (2000), *Statistics: Concepts and Controversies*, Fifth Edition, New York: W.H. Freeman & Company.

Mudholkar, G.S. (1983), “Fisher’s z-Transformation,” *Encyclopedia of Statistical Sciences,* 3, 130–135.

Noether, G.E. (1967), *Elements of Nonparametric Statistics*, New York: John Wiley & Sons, Inc.

Novick, M.R. (1967), “Coefficient Alpha and the Reliability of Composite Measurements,” *Psychometrika* , 32, 1–13.

Nunnally, J.C. and Bernstein, I.H. (1994), *Psychometric Theory,* Third Edition, New York: McGraw-Hill Companies.

Ott, R.L. and Longnecker, M.T. (2000), *An Introduction to Statistical Methods and Data Analysis*, Fifth Edition, Belmont, CA: Wadsworth Publishing Company.

SAS Institute Inc., “Measuring the Internal Consistency of a Test, Using PROC CORR to Compute Cronbach’s Coefficient Alpha,” *SAS Communications* , 20:4, TT2–TT5.

Spector, P.E. (1992), *Summated Rating Scale Construction: An Introduction*, Newbury Park: Sage.

Chapter 2
The FREQ Procedure

Chapter Contents

Chapter 2
The FREQ Procedure

Overview

The FREQ procedure produces one-way to *n*-way frequency and crosstabulation (contingency) tables. For two-way tables, PROC FREQ computes tests and measures of association. For *n*-way tables, PROC FREQ does stratified analysis, computing statistics within, as well as across, strata. Frequencies and statistics can also be output to SAS data sets.

For one-way frequency tables, PROC FREQ can compute statistics to test for equal proportions, specified proportions, or the binomial proportion. For contingency tables, PROC FREQ can compute various statistics to examine the relationships between two classification variables adjusting for any stratification variables. PROC FREQ automatically displays the output in a report and can also save the output in a SAS data set.

For some pairs of variables, you may want to examine the existence or the strength of any association between the variables. To determine if an association exists, chi-square tests are computed. To estimate the strength of an association, PROC FREQ computes measures of association that tend to be close to zero when there is no association and close to the maximum (or minimum) value when there is perfect association. The statistics for contingency tables include

- chi-square tests and measures
- measures of association
- risks (binomial proportions) and risk differences for 2×2 tables
- odds ratios and relative risks for 2×2 tables
- tests for trend
- tests and measures of agreement
- Cochran-Mantel-Haenszel statistics

PROC FREQ computes asymptotic standard errors, confidence intervals, and tests for measures of association and measures of agreement. Exact *p*-values and confidence intervals are available for various test statistics and measures. PROC FREQ also performs stratified analyses that compute statistics within, as well as across, strata for *n*-way tables. The statistics include Cochran-Mantel-Haenszel statistics and measures of agreement.

In choosing measures of association to use in analyzing a two-way table, you should consider the study design (which indicates whether the row and column variables are dependent or independent), the measurement scale of the variables (nominal, ordinal, or interval), the type of association that each measure is designed to detect, and any assumptions required for valid interpretation of a measure. You should exercise care in selecting measures that are appropriate for your data.

Similar comments apply to the choice and interpretation of the test statistics. For example, the Mantel-Haenszel chi-square statistic requires an ordinal scale for both variables and is designed to detect a linear association. The Pearson chi-square, on the other hand, is appropriate for all variables and can detect any kind of association, but it is less powerful for detecting a linear association because its power is dispersed over a greater number of degrees of freedom (except for 2×2 tables).

Several SAS procedures produce frequency counts; only PROC FREQ computes chi-square tests for one-way to *n*-way tables and measures of association and agreement for contingency tables. Other procedures to consider for counting are TABULATE, CHART, and UNIVARIATE. When you want to fit models to categorical data, use a procedure such as CATMOD, GENMOD, LOGISTIC, PHREG, or PROBIT.

For more information on selecting the appropriate statistical analyses, refer to Agresti (1996) or Stokes, Davis, and Koch (1995).

Getting Started

Frequency Tables and Statistics

The FREQ procedure provides easy access to statistics for testing for association in a crosstabulation table.

In this example, high school students applied for courses in a summer enrichment program: these courses included journalism, art history, statistics, graphic arts, and computer programming. The students accepted were randomly assigned to classes with and without internships in local companies. The following table contains counts of the students who enrolled in the summer program by gender and whether they were assigned an internship slot.

Table 2.1. Summer Enrichment Data

		Enrollment		
Gender	**Internship**	**Yes**	**No**	**Total**
boys	yes	35	29	64
boys	no	14	27	41
girls	yes	32	10	42
girls	no	53	23	76

The SAS data set SummerSchool is created by inputting the summer enrichment data as cell count data, or providing the frequency count for each combination of variable values. The following DATA step statements create the SAS data set SummerSchool.

```
data SummerSchool;
   input Gender $ Internship $ Enrollment $ Count @@;
   datalines;
boys  yes yes 35   boys  yes no 29
boys   no yes 14   boys   no no 27
girls yes yes 32   girls yes no 10
girls  no yes 53   girls  no no 23
;
```

The variable Gender takes the values 'boys' or 'girls', the variable Internship takes the values 'yes' and 'no', and the variable Enrollment takes the values 'yes' and 'no'. The variable Count contains the number of students corresponding to each combination of data values. The double at sign (@@) indicates that more than one observation is included on a single data line. In this DATA step, two observations are included on each line.

Researchers are interested in whether there is an association between internship status and summer program enrollment. The Pearson chi-square statistic is an appropriate statistic to assess the association in the corresponding 2×2 table. The following PROC FREQ statements specify this analysis.

You specify the table for which you want to compute statistics with the TABLES statement. You specify the statistics you want to compute with options after a slash (/) in the TABLES statement.

```
proc freq data=SummerSchool order=data;
   weight count;
   tables Internship*Enrollment / chisq;
run;
```

The ORDER= option controls the order in which variable values are displayed in the rows and columns of the table. By default, the values are arranged according to the alphanumeric order of their unformatted values. If you specify ORDER=DATA, the data are displayed in the same order as they occur in the input data set. Here, since 'yes' appears before 'no' in the data, 'yes' appears first in any table. Other options for controlling order include ORDER=FORMATTED, which orders according to the formatted values, and ORDER=FREQUENCY, which orders by descending frequency count.

In the TABLES statement, Internship*Enrollment specifies a table where the rows are internship status and the columns are program enrollment. The CHISQ option requests chi-square statistics for assessing association between these two variables. Since the input data are in cell count form, the WEIGHT statement is required. The WEIGHT statement names the variable Count, which provides the frequency of each combination of data values.

Figure 2.1 presents the crosstabulation of Internship and Enrollment. In each cell, the values printed under the cell count are the table percentage, row percentage, and column percentage, respectively. For example, in the first cell, 63.21 percent of those offered courses with internships accepted them and 36.79 percent did not.

```
                        The FREQ Procedure

                  Table of Internship by Enrollment

                 Internship     Enrollment

                 Frequency|
                 Percent  |
                 Row Pct  |
                 Col Pct  |yes     |no      |  Total
                 ---------+--------+--------+
                 yes      |     67 |     39 |    106
                          |  30.04 |  17.49 |  47.53
                          |  63.21 |  36.79 |
                          |  50.00 |  43.82 |
                 ---------+--------+--------+
                 no       |     67 |     50 |    117
                          |  30.04 |  22.42 |  52.47
                          |  57.26 |  42.74 |
                          |  50.00 |  56.18 |
                 ---------+--------+--------+
                 Total         134       89      223
                             60.09    39.91   100.00
```

Figure 2.1. Crosstabulation Table

```
         Statistics for Table of Internship by Enrollment

        Statistic                     DF       Value      Prob
        ------------------------------------------------------
        Chi-Square                     1      0.8189    0.3655
        Likelihood Ratio Chi-Square    1      0.8202    0.3651
        Continuity Adj. Chi-Square     1      0.5899    0.4425
        Mantel-Haenszel Chi-Square     1      0.8153    0.3666
        Phi Coefficient                       0.0606
        Contingency Coefficient               0.0605
        Cramer's V                            0.0606


                          Fisher's Exact Test
                  ----------------------------------
                  Cell (1,1) Frequency (F)        67
                  Left-sided Pr <= F          0.8513
                  Right-sided Pr >= F         0.2213

                  Table Probability (P)       0.0726
                  Two-sided Pr <= P           0.4122

                          Sample Size = 223
```

Figure 2.2. Statistics Produced with the CHISQ Option

Figure 2.2 displays the statistics produced by the CHISQ option. The Pearson chi-square statistic is labeled 'Chi-Square' and has a value of 0.8189 with 1 degree of freedom. The associated p-value is 0.3655, which means that there is no significant evidence of an association between internship status and program enrollment. The other chi-square statistics have similar values and are asymptotically equivalent. The other statistics (Phi Coefficient, Contingency Coefficient, and Cramer's V) are measures of association derived from the Pearson chi-square. For Fisher's exact test, the two-sided p-value is 0.4122, which also shows no association between internship status and program enrollment.

The analysis, so far, has ignored gender. However, it may be of interest to ask whether program enrollment is associated with internship status after adjusting for gender. You can address this question by doing an analysis of a set of tables, in this case, by analyzing the set consisting of one for boys and one for girls. The Cochran-Mantel-Haenszel statistic is appropriate for this situation: it addresses whether rows and columns are associated after controlling for the stratification variable. In this case, you would be stratifying by gender.

The FREQ statements for this analysis are very similar to those for the first analysis, except that there is a third variable, Gender, in the TABLES statement. When you cross more than two variables, the two rightmost variables construct the rows and columns of the table, respectively, and the leftmost variables determine the stratification.

```
proc freq data=SummerSchool;
   weight count;
   tables Gender*Internship*Enrollment / chisq cmh;
run;
```

This execution of PROC FREQ first produces two individual crosstabulation tables of Internship*Enrollment, one for boys and one for girls. Chi-square statistics are produced for each individual table. Figure 2.3 shows the results for boys. Note that the chi-square statistic for boys is significant at the $\alpha = 0.05$ level of significance. Boys offered a course with an internship are more likely to enroll than boys who are not.

If you look at the individual table for girls, displayed in Figure 2.4, you see that there is no evidence of association for girls between internship offers and program enrollment.

```
                         The FREQ Procedure

                Table 1 of Internship by Enrollment
                   Controlling for Gender=boys

                Internship     Enrollment

                Frequency|
                Percent  |
                Row Pct  |
                Col Pct  |no      |yes     |  Total
                ---------+--------+--------+
                no       |     27 |     14 |     41
                         |  25.71 |  13.33 |  39.05
                         |  65.85 |  34.15 |
                         |  48.21 |  28.57 |
                ---------+--------+--------+
                yes      |     29 |     35 |     64
                         |  27.62 |  33.33 |  60.95
                         |  45.31 |  54.69 |
                         |  51.79 |  71.43 |
                ---------+--------+--------+
                Total          56       49      105
                            53.33    46.67   100.00


         Statistics for Table 1 of Internship by Enrollment
                   Controlling for Gender=boys

       Statistic                     DF       Value      Prob
       ------------------------------------------------------
       Chi-Square                     1      4.2366    0.0396
       Likelihood Ratio Chi-Square    1      4.2903    0.0383
       Continuity Adj. Chi-Square     1      3.4515    0.0632
       Mantel-Haenszel Chi-Square     1      4.1963    0.0405
       Phi Coefficient                       0.2009
       Contingency Coefficient               0.1969
       Cramer's V                            0.2009


                        Fisher's Exact Test
                 ----------------------------------
                 Cell (1,1) Frequency (F)        27
                 Left-sided Pr <= F          0.9885
                 Right-sided Pr >= F         0.0311

                 Table Probability (P)       0.0196
                 Two-sided Pr <= P           0.0467

                         Sample Size = 105
```

Figure 2.3. Crosstabulation Table and Statistics for Boys

```
                Table 2 of Internship by Enrollment
                    Controlling for Gender=girls

                Internship     Enrollment

                Frequency|
                Percent  |
                Row Pct  |
                Col Pct  |no      |yes     |  Total
                ---------+--------+--------+
                no       |     23 |     53 |     76
                         |  19.49 |  44.92 |  64.41
                         |  30.26 |  69.74 |
                         |  69.70 |  62.35 |
                ---------+--------+--------+
                yes      |     10 |     32 |     42
                         |   8.47 |  27.12 |  35.59
                         |  23.81 |  76.19 |
                         |  30.30 |  37.65 |
                ---------+--------+--------+
                Total          33       85      118
                            27.97    72.03   100.00


         Statistics for Table 2 of Internship by Enrollment
                    Controlling for Gender=girls

       Statistic                     DF       Value      Prob
       ------------------------------------------------------
       Chi-Square                     1      0.5593    0.4546
       Likelihood Ratio Chi-Square    1      0.5681    0.4510
       Continuity Adj. Chi-Square     1      0.2848    0.5936
       Mantel-Haenszel Chi-Square     1      0.5545    0.4565
       Phi Coefficient                       0.0688
       Contingency Coefficient               0.0687
       Cramer's V                            0.0688


                       Fisher's Exact Test
                 ----------------------------------
                 Cell (1,1) Frequency (F)        23
                 Left-sided Pr <= F          0.8317
                 Right-sided Pr >= F         0.2994

                 Table Probability (P)       0.1311
                 Two-sided Pr <= P           0.5245

                         Sample Size = 118
```

Figure 2.4. Crosstabulation Table and Statistics for Girls

These individual table results demonstrate the occasional problems with combining information into one table and not accounting for information in other variables such as Gender. Figure 2.5 contains the CMH results. There are three summary (CMH) statistics; which one you use depends on whether your rows and/or columns have an order in $r \times c$ tables. However, in the case of 2×2 tables, ordering does not matter and all three statistics take the same value. The CMH statistic follows the chi-square distribution under the hypothesis of no association, and here, it takes the value 4.0186 with 1 degree of freedom. The associated p-value is 0.0450, which indicates a significant association at the $\alpha = 0.05$ level.

Thus, when you adjust for the effect of gender in these data, there is an association between internship and program enrollment. But, if you ignore gender, no association is found. Note that the CMH option also produces other statistics, including estimates and confidence limits for relative risk and odds ratios for 2×2 tables and the Breslow-Day Test. These results are not displayed here.

```
             Summary Statistics for Internship by Enrollment
                          Controlling for Gender

          Cochran-Mantel-Haenszel Statistics (Based on Table Scores)

         Statistic    Alternative Hypothesis    DF       Value      Prob
         ---------------------------------------------------------------
             1        Nonzero Correlation        1      4.0186    0.0450
             2        Row Mean Scores Differ     1      4.0186    0.0450
             3        General Association        1      4.0186    0.0450

                           Total Sample Size = 223
```

Figure 2.5. Test for the Hypothesis of No Association

Agreement Study Example

Medical researchers are interested in evaluating the efficacy of a new treatment for a skin condition. Dermatologists from participating clinics were trained to conduct the study and to evaluate the condition. After the training, two dermatologists examined patients with the skin condition from a pilot study and rated the same patients. The possible evaluations are terrible, poor, marginal, and clear. Table 2.2 contains the data.

Table 2.2. Skin Condition Data

	Dermatologist 2			
Dermatologist 1	**Terrible**	**Poor**	**Marginal**	**Clear**
Terrible	10	4	1	0
Poor	5	10	12	2
Marginal	2	4	12	5
Clear	0	2	6	13

The dermatologists' evaluations of the patients are contained in the variables derm1 and derm2; the variable count is the number of patients given a particular pair of ratings. In order to evaluate the agreement of the diagnoses (a possible contribution to measurement error in the study), the *kappa coefficient* is computed. You specify the AGREE option in the TABLES statement and use the TEST statement to request a test for the null hypothesis that their agreement is purely by chance. You specify the keyword KAPPA to perform this test for the kappa coefficient. The results are shown in Figure 2.6.

```
data SkinCondition;
   input derm1 $ derm2 $ count;
   datalines;
terrible terrible 10
terrible     poor 4
terrible marginal 1
terrible    clear 0
poor     terrible 5
poor         poor 10
poor     marginal 12
poor        clear 2
marginal terrible 2
marginal     poor 4
marginal marginal 12
marginal    clear 5
clear    terrible 0
clear        poor 2
clear    marginal 6
clear       clear 13
;
```

```
proc freq data=SkinCondition order=data;
   weight count;
   tables derm1*derm2 / agree noprint;
   test kappa;
run;
```

```
                        The FREQ Procedure

              Statistics for Table of derm1 by derm2

                     Simple Kappa Coefficient
                  --------------------------------
                  Kappa                     0.3449
                  ASE                       0.0724
                  95% Lower Conf Limit      0.2030
                  95% Upper Conf Limit      0.4868

                       Test of H0: Kappa = 0

                  ASE under H0              0.0612
                  Z                         5.6366
                  One-sided Pr >  Z         <.0001
                  Two-sided Pr > |Z|        <.0001

                          Sample Size = 88
```

Figure 2.6. Agreement Study

The kappa coefficient has the value 0.3449, which indicates slight agreement between the dermatologists, and the hypothesis test confirms that you can reject the null hypothesis of no agreement. This conclusion is further supported by the confidence interval of (0.2030, 0.4868), which suggests that the true kappa is greater than zero. The AGREE option also produces Bowker's test for symmetry and the weighted kappa coefficient, but that output is not shown.

Syntax

The following statements are available in PROC FREQ.

PROC FREQ < *options* > **;**
  BY *variables* **;**
  EXACT *statistic-options* < **/** *computation-options* > **;**
  OUTPUT < **OUT=***SAS-data-set* > *options* **;**
  TABLES *requests* < **/** *options* > **;**
  TEST *options* **;**
  WEIGHT *variable* < **/** *option* > **;**

The PROC FREQ statement is the only required statement for the FREQ procedure. If you specify the following statements, PROC FREQ produces a one-way frequency table for each variable in the most recently created data set.

```
proc freq;
run;
```

The rest of this section gives detailed syntax information for the BY, EXACT, OUTPUT, TABLES, TEST, and WEIGHT statements in alphabetical order after the description of the PROC FREQ statement. Table 2.3 summarizes the basic functions of each statement.

Table 2.3. Summary of PROC FREQ Statements

Statement	Description
BY	calculates separate frequency or crosstabulation tables for each BY group.
EXACT	requests exact tests for specified statistics.
OUTPUT	creates an output data set that contains specified statistics.
TABLES	specifies frequency or crosstabulation tables and requests tests and measures of association.
TEST	requests asymptotic tests for measures of association and agreement.
WEIGHT	identifies a variable with values that weight each observation.

PROC FREQ Statement

PROC FREQ < *options* > **;**

The PROC FREQ statement invokes the procedure.

The following table lists the options available in the PROC FREQ statement. Descriptions follow in alphabetical order.

Table 2.4. PROC FREQ Statement Options

Option	Description
DATA=	specifies the input data set.
COMPRESS	begins the next one-way table on the current page
FORMCHAR=	specifies the outline and cell divider characters for the cells of the crosstabulation table.
NLEVELS	displays the number of levels for all TABLES variables
NOPRINT	suppresses all displayed output.
ORDER=	specifies the order for listing variable values.
PAGE	displays one table per page.

You can specify the following options in the PROC FREQ statement.

COMPRESS

begins display of the next one-way frequency table on the same page as the preceding one-way table if there is enough space to begin the table. By default, the next one-way table begins on the current page only if the entire table fits on that page. The COMPRESS option is not valid with the PAGE option.

DATA=*SAS-data-set*

names the SAS data set to be analyzed by PROC FREQ. If you omit the DATA= option, the procedure uses the most recently created SAS data set.

FORMCHAR (1,2,7) =’*formchar-string***’**

defines the characters to be used for constructing the outlines and dividers for the cells of contingency tables. The FORMCHAR= option can specify 20 different SAS formatting characters used to display output; however, PROC FREQ uses only the first, second, and seventh formatting characters. Therefore, the proper specification for PROC FREQ is FORMCHAR(1,2,7)= ’*formchar-string*’. The *formchar-string* should be three characters long. The characters are used to denote (1) vertical separator, (2) horizontal separator, and (7) vertical-horizontal intersection. You can use any character in *formchar-string*, including hexadecimal characters. If you use hexadecimal characters, you must put an `x` after the closing quote. For information on which hexadecimal codes to use for which characters, consult the documentation for your hardware.

Specifying all blanks for *formchar-string* produces tables with no outlines or dividers:

```
formchar (1,2,7)='   '
```

If you do not specify the FORMCHAR= option, PROC FREQ uses the default

```
formchar (1,2,7)='|-+'
```

Refer to the CALENDAR, PLOT, and TABULATE procedures in the *Base SAS 9.1 Procedures Guide* for more information on form characters.

Table 2.5. Formatting Characters Used by PROC FREQ

Position	Default	Used to Draw
1	|	vertical separators
2	-	horizontal separators
7	+	intersections of vertical and horizontal separators

NLEVELS

displays the “Number of Variable Levels” table. This table provides the number of levels for each variable named in the TABLES statements. See the section “Number of Variable Levels Table” on page 151 for more information. PROC FREQ determines the variable levels from the formatted variable values, as described in the section “Grouping with Formats” on page 99.

NOPRINT

suppresses the display of all output. Note that this option temporarily disables the Output Delivery System (ODS). For more information, see Chapter 14, “Using the Output Delivery System.” (*SAS/STAT User’s Guide*) .

Note: A NOPRINT option is also available in the TABLES statement. It suppresses display of the crosstabulation tables but allows display of the requested statistics.

ORDER=DATA | FORMATTED | FREQ | INTERNAL

specifies the order in which the values of the frequency and crosstabulation table variables are to be reported. The following table shows how PROC FREQ interprets values of the ORDER= option.

DATA	orders values according to their order in the input data set.
FORMATTED	orders values by their formatted values. This order is operating-environment dependent. By default, the order is ascending.
FREQ	orders values by descending frequency count.
INTERNAL	orders values by their unformatted values, which yields the same order that the SORT procedure does. This order is operating-environment dependent.

By default, ORDER=INTERNAL. The ORDER= option does not apply to missing values, which are always ordered first.

PAGE
displays only one table per page. Otherwise, PROC FREQ displays multiple tables per page as space permits. The PAGE option is not valid with the COMPRESS option.

BY Statement

BY *variables* **;**

You can specify a BY statement with PROC FREQ to obtain separate analyses on observations in groups defined by the BY variables. When a BY statement appears, the procedure expects the input data set to be sorted in order of the BY variables.

If your input data set is not sorted in ascending order, use one of the following alternatives:

- Sort the data using the SORT procedure with a similar BY statement.
- Specify the BY statement option NOTSORTED or DESCENDING in the BY statement for the FREQ procedure. The NOTSORTED option does not mean that the data are unsorted but rather that the data are arranged in groups (according to values of the BY variables) and that these groups are not necessarily in alphabetical or increasing numeric order.
- Create an index on the BY variables using the DATASETS procedure.

For more information on the BY statement, refer to the discussion in *SAS Language Reference: Concepts*. For more information on the DATASETS procedure, refer to the discussion in the *Base SAS 9.1 Procedures Guide*.

EXACT Statement

EXACT *statistic-options* < */ computation-options* > **;**

The EXACT statement requests exact tests or confidence limits for the specified statistics. Optionally, PROC FREQ computes Monte Carlo estimates of the exact *p*-values. The *statistic-options* specify the statistics for which to provide exact tests or confidence limits. The *computation-options* specify options for the computation of exact statistics.

CAUTION: PROC FREQ computes exact tests with fast and efficient algorithms that are superior to direct enumeration. Exact tests are appropriate when a data set is small, sparse, skewed, or heavily tied. For some large problems, computation of exact tests may require a large amount of time and memory. Consider using asymptotic tests for such problems. Alternatively, when asymptotic methods may not be sufficient for such large problems, consider using Monte Carlo estimation of exact *p*-values. See the section "Computational Resources" on page 145 for more information.

Statistic-Options

The *statistic-options* specify the statistics for which exact tests or confidence limits are computed. PROC FREQ can compute exact *p*-values for the following hypothesis tests: chi-square goodness-of-fit test for one-way tables; Pearson chi-square, likelihood-ratio chi-square, Mantel-Haenszel chi-square, Fisher's exact test, Jonckheere-Terpstra test, Cochran-Armitage test for trend, and McNemar's test for two-way tables. PROC FREQ can also compute exact *p*-values for tests of the following statistics: Pearson correlation coefficient, Spearman correlation coefficient, simple kappa coefficient, weighted kappa coefficient, and common odds ratio. PROC FREQ can compute exact *p*-values for the binomial proportion test for one-way tables, as well as exact confidence limits for the binomial proportion. Additionally, PROC FREQ can compute exact confidence limits for the odds ratio for 2×2 tables, as well as exact confidence limits for the common odds ratio for stratified 2×2 tables.

Table 2.6 lists the available *statistic-options* and the exact statistics computed. Most of the option names are identical to the corresponding options in the TABLES statement and the OUTPUT statement. You can request exact computations for groups of statistics by using options that are identical to the following TABLES statement options: CHISQ, MEASURES, and AGREE. For example, when you specify the CHISQ option in the EXACT statement, PROC FREQ computes exact *p*-values for the Pearson chi-square, likelihood-ratio chi-square, and Mantel-Haenszel chi-square tests. You request exact *p*-values for an individual test by specifying one of the *statistic-options* shown in Table 2.6 .

Table 2.6. EXACT Statement Statistic-Options

Option	Exact Statistics Computed
AGREE	McNemar's test for 2×2 tables, simple kappa coefficient, and weighted kappa coefficient
BINOMIAL	binomial proportion test for one-way tables
CHISQ	chi-square goodness-of-fit test for one-way tables; Pearson chi-square, likelihood-ratio chi-square, and Mantel-Haenszel chi-square tests for two-way tables
COMOR	confidence limits for the common odds ratio for $h \times 2 \times 2$ tables; common odds ratio test
FISHER	Fisher's exact test
JT	Jonckheere-Terpstra test
KAPPA	test for the simple kappa coefficient
LRCHI	likelihood-ratio chi-square test
MCNEM	McNemar's test
MEASURES	tests for the Pearson correlation and the Spearman correlation, and the odds ratio confidence limits for 2×2 tables
MHCHI	Mantel-Haenszel chi-square test
OR	confidence limits for the odds ratio for 2×2 tables
PCHI	Pearson chi-square test
PCORR	test for the Pearson correlation coefficient
SCORR	test for the Spearman correlation coefficient
TREND	Cochran-Armitage test for trend
WTKAP	test for the weighted kappa coefficient

Computation-Options

The *computation-options* specify options for computation of exact statistics. You can specify the following *computation-options* in the EXACT statement.

ALPHA=α

specifies the level of the confidence limits for Monte Carlo *p*-value estimates. The value of the ALPHA= option must be between 0 and 1, and the default is 0.01. A confidence level of α produces $100(1-\alpha)$% confidence limits. The default of ALPHA=.01 produces 99% confidence limits for the Monte Carlo estimates. The ALPHA= option invokes the MC option.

MAXTIME=*value*

specifies the maximum clock time (in seconds) that PROC FREQ can use to compute an exact *p*-value. If the procedure does not complete the computation within the specified time, the computation terminates. The value of the MAXTIME= option must be a positive number. The MAXTIME= option is valid for Monte Carlo estimation of exact *p*-values, as well as for direct exact *p*-value computation.

See the section "Computational Resources" on page 145 for more information.

MC

requests Monte Carlo estimation of exact *p*-values instead of direct exact *p*-value computation. Monte Carlo estimation can be useful for large problems that require a great amount of time and memory for exact computations but for which asymptotic approximations may not be sufficient. See the section "Monte Carlo Estimation" on page 146 for more information.

The MC option is available for all EXACT *statistic-options* except BINOMIAL, COMOR, MCNEM, and OR. PROC FREQ computes only exact tests or confidence limits for those statistics.

The ALPHA=, N=, and SEED= options also invoke the MC option.

N=*n*

specifies the number of samples for Monte Carlo estimation. The value of the N= option must be a positive integer, and the default is 10000 samples. Larger values of *n* produce more precise estimates of exact *p*-values. Because larger values of *n* generate more samples, the computation time increases. The N= option invokes the MC option.

POINT

requests exact point probabilities for the test statistics.

The POINT option is available for all the EXACT statement *statistic-options* except the OR option, which provides exact confidence limits as opposed to an exact test. The POINT option is not available with the MC option.

SEED=*number*

specifies the initial seed for random number generation for Monte Carlo estimation. The value of the SEED= option must be an integer. If you do not specify the SEED= option, or if the SEED= value is negative or zero, PROC FREQ uses the time of day from the computer's clock to obtain the initial seed. The SEED= option invokes the MC option.

Using TABLES Statement Options with the EXACT Statement

If you use only one TABLES statement, you do not need to specify options in the TABLES statement that are identical to options appearing in the EXACT statement. PROC FREQ automatically invokes the corresponding TABLES statement option when you specify the option in the EXACT statement. However, when you use multiple TABLES statements and want exact computations, you must specify options in the TABLES statement to compute the desired statistics. PROC FREQ then performs exact computations for all statistics that are also specified in the EXACT statement.

OUTPUT Statement

OUTPUT < **OUT=** *SAS-data-set* > *options* **;**

The OUTPUT statement creates a SAS data set containing statistics computed by PROC FREQ. The variables contain statistics for each two-way table or stratum, as well as summary statistics across all strata.

Only one OUTPUT statement is allowed for each execution of PROC FREQ. You must specify a TABLES statement with the OUTPUT statement. If you use multiple TABLES statements, the contents of the OUTPUT data set correspond to the last TABLES statement. If you use multiple table requests in a TABLES statement, the contents of the OUTPUT data set correspond to the last table request.

For more information, see the section "Output Data Sets" on page 148.

Note that you can use the Output Delivery System (ODS) to create a SAS data set from any piece of PROC FREQ output. For more information, see Table 2.11 on page 159 and Chapter 14, "Using the Output Delivery System." (*SAS/STAT User's Guide*)

You can specify the following options in an OUTPUT statement.

OUT=*SAS-data-set*

names the output data set. If you omit the OUT= option, the data set is named DATA*n*, where *n* is the smallest integer that makes the name unique.

options

specify the statistics that you want in the output data set. Available statistics are those produced by PROC FREQ for each one-way or two-way table, as well as the summary statistics across all strata. When you request a statistic, the OUTPUT data set contains that estimate or test statistic plus any associated standard error, confidence limits, *p*-values, and degrees of freedom. You can output statistics by using group options identical to those specified in the TABLES statement: AGREE, ALL, CHISQ, CMH, and MEASURES. Alternatively, you can request an individual statistic by specifying one of the options shown in the following table.

Table 2.7. OUTPUT Statement Options and Required TABLES Statement Options

Option	**Output Data Set Statistics**	**Required TABLES Statement Option**
AGREE	McNemar's test for 2×2 tables, simple kappa coefficient, and weighted kappa coefficient; for square tables with more than two response categories, Bowker's test of symmetry; for multiple strata, overall simple and weighted kappa statistics, and tests for equal kappas among strata; for multiple strata with two response categories, Cochran's Q test	AGREE
AJCHI	continuity-adjusted chi-square for 2×2 tables	ALL or CHISQ
ALL	all statistics under CHISQ, MEASURES, and CMH, and the number of nonmissing subjects	ALL
BDCHI	Breslow-Day test	ALL or CMH or CMH1 or CMH2
BIN | BINOMIAL	for one-way tables, binomial proportion statistics	BINOMIAL
CHISQ	chi-square goodness-of-fit test for one-way tables; for two-way tables, Pearson chi-square, likelihood-ratio chi-square, continuity-adjusted chi-square for 2×2 tables, Mantel-Haenszel chi-square, Fisher's exact test for 2×2 tables, phi coefficient, contingency coefficient, and Cramer's V	ALL or CHISQ
CMH	Cochran-Mantel-Haenszel correlation, row mean scores (ANOVA), and general association statistics; for 2×2 tables, logit and Mantel-Haenszel adjusted odds ratios, relative risks, and Breslow-Day test	ALL or CMH
CMH1	same as CMH, but excludes general association and row mean scores (ANOVA) statistics	ALL or CMH or CMH1
CMH2	same as CMH, but excludes the general association statistic	ALL or CMH or CMH2
CMHCOR	Cochran-Mantel-Haenszel correlation statistic	ALL or CMH or CMH1 or CMH2
CMHGA	Cochran-Mantel-Haenszel general association statistic	ALL or CMH
CMHRMS	Cochran-Mantel-Haenszel row mean scores (ANOVA) statistic	ALL or CMH or CMH2
COCHQ	Cochran's Q	AGREE

Table 2.7. (continued)

Option	Output Data Set Statistics	Required TABLES Statement Option
CONTGY	contingency coefficient	ALL or CHISQ
CRAMV	Cramer's V	ALL or CHISQ
EQKAP	test for equal simple kappas	AGREE
EQWKP	test for equal weighted kappas	AGREE
FISHER | EXACT	Fisher's exact test	ALL or CHISQ *
GAMMA	gamma	ALL or MEASURES
JT	Jonckheere-Terpstra test	JT
KAPPA	simple kappa coefficient	AGREE
KENTB	Kendall's tau-b	ALL or MEASURES
LAMCR	lambda asymmetric $(C\|R)$	ALL or MEASURES
LAMDAS	lambda symmetric	ALL or MEASURES
LAMRC	lambda asymmetric $(R\|C)$	ALL or MEASURES
LGOR	adjusted logit odds ratio	ALL or CMH or CMH1 or CMH2
LGRRC1	adjusted column 1 logit relative risk	ALL or CMH or CMH1 or CMH2
LGRRC2	adjusted column 2 logit relative risk	ALL or CMH or CMH1 or CMH2
LRCHI	likelihood-ratio chi-square	ALL or CHISQ
MCNEM	McNemar's test	AGREE
MEASURES	gamma, Kendall's tau-b, Stuart's tau-c, Somers' $D(C\|R)$, Somers' $D(R\|C)$, Pearson correlation coefficient, Spearman correlation coefficient, lambda asymmetric $(C\|R)$, lambda asymmetric $(R\|C)$, lambda symmetric, uncertainty coefficient $(C\|R)$, uncertainty coefficient $(R\|C)$, and symmetric uncertainty coefficient; for 2×2 tables, odds ratio and relative risks	ALL or MEASURES
MHCHI	Mantel-Haenszel chi-square	ALL or CHISQ
MHOR	adjusted Mantel-Haenszel odds ratio	ALL or CMH or CMH1 or CMH2
MHRRC1	adjusted column 1 Mantel-Haenszel relative risk	ALL or CMH or CMH1 or CMH2
MHRRC2	adjusted column 2 Mantel-Haenszel relative risk	ALL or CMH or CMH1 or CMH2
N	number of nonmissing subjects for the stratum	
NMISS	number of missing subjects for the stratum	

*ALL and CHISQ compute Fisher's exact test for 2×2 tables. Use the FISHER option to compute Fisher's exact test for general *rxc* tables.

Table 2.7. (continued)

Option	**Output Data Set Statistics**	**Required TABLES Statement Option**
OR	odds ratio	ALL or MEASURES or RELRISK
PCHI	chi-square goodness-of-fit test for one-way tables; for two-way tables, Pearson chi-square	ALL or CHISQ
PCORR	Pearson correlation coefficient	ALL or MEASURES
PHI	phi coefficient	ALL or CHISQ
PLCORR	polychoric correlation coefficient	PLCORR
RDIF1	column 1 risk difference (row 1 - row 2)	RISKDIFF
RDIF2	column 2 risk difference (row 1 - row 2)	RISKDIFF
RELRISK	odds ratio and relative risks for 2×2 tables	ALL or MEASURES or RELRISK
RISKDIFF	risks and risk differences	RISKDIFF
RISKDIFF1	column 1 risks and risk difference	RISKDIFF
RISKDIFF2	column 2 risks and risk difference	RISKDIFF
RRC1	column 1 relative risk	ALL or MEASURES or RELRISK
RRC2	column 2 relative risk	ALL or MEASURES or RELRISK
RSK1	column 1 risk (overall)	RISKDIFF
RSK11	column 1 risk, for row 1	RISKDIFF
RSK12	column 2 risk, for row 1	RISKDIFF
RSK2	column 2 risk (overall)	RISKDIFF
RSK21	column 1 risk, for row 2	RISKDIFF
RSK22	column 2 risk, for row 2	RISKDIFF
SCORR	Spearman correlation coefficient	ALL or MEASURES
SMDCR	Somers' $D(C\|R)$	ALL or MEASURES
SMDRC	Somers' $D(R\|C)$	ALL or MEASURES
STUTC	Stuart's tau-c	ALL or MEASURES
TREND	Cochran-Armitage test for trend	TREND
TSYMM	Bowker's test of symmetry	AGREE
U	symmetric uncertainty coefficient	ALL or MEASURES
UCR	uncertainty coefficient $(C\|R)$	ALL or MEASURES
URC	uncertainty coefficient $(R\|C)$	ALL or MEASURES
WTKAP	weighted kappa coefficient	AGREE

Using the TABLES Statement with the OUTPUT Statement

In order to specify that the OUTPUT data set contain a particular statistic, you must have PROC FREQ compute the statistic by using the corresponding option in the TABLES statement or the EXACT statement. For example, you cannot specify the option PCHI (Pearson chi-square) in the OUTPUT statement without also specifying a TABLES statement option or an EXACT statement option to compute the Pearson chi-square. The TABLES statement option ALL or CHISQ computes the Pearson chi-square. Additionally, if you have only one TABLES statement, the EXACT statement option CHISQ or PCHI computes the Pearson chi-square.

TABLES Statement

TABLES *requests* < **/** *options* > **;**

The TABLES statement requests one-way to *n*-way frequency and crosstabulation tables and statistics for those tables.

If you omit the TABLES statement, PROC FREQ generates one-way frequency tables for all data set variables that are not listed in the other statements.

The following argument is required in the TABLES statement.

requests

specify the frequency and crosstabulation tables to produce. A request is composed of one variable name or several variable names separated by asterisks. To request a one-way frequency table, use a single variable. To request a two-way crosstabulation table, use an asterisk between two variables. To request a multiway table (an *n*-way table, where $n>2$), separate the desired variables with asterisks. The unique values of these variables form the rows, columns, and strata of the table.

For two-way to multiway tables, the values of the last variable form the crosstabulation table columns, while the values of the next-to-last variable form the rows. Each level (or combination of levels) of the other variables forms one stratum. PROC FREQ produces a separate crosstabulation table for each stratum. For example, a specification of A*B*C*D in a TABLES statement produces *k* tables, where *k* is the number of different combinations of values for A and B. Each table lists the values for C down the side and the values for D across the top.

You can use multiple TABLES statements in the PROC FREQ step. PROC FREQ builds all the table requests in one pass of the data, so that there is essentially no loss of efficiency. You can also specify any number of table requests in a single TABLES statement. To specify multiple table requests 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.

Table 2.8. Grouping Syntax

Request	Equivalent to
tables A*(B C);	tables A*B A*C;
tables (A B)*(C D);	tables A*C B*C A*D B*D;
tables (A B C)*D;	tables A*D B*D C*D;
tables A – – C;	tables A B C;
tables (A – – C)*D;	tables A*D B*D C*D;

Without Options

If you request a one-way frequency table for a variable without specifying options, PROC FREQ produces frequencies, cumulative frequencies, percentages of the total frequency, and cumulative percentages for each value of the variable. If you request a two-way or an *n*-way crosstabulation table without specifying options, PROC FREQ produces crosstabulation tables that include cell frequencies, cell percentages of the total frequency, cell percentages of row frequencies, and cell percentages of column frequencies. The procedure excludes observations with missing values from the table but displays the total frequency of missing observations below each table.

Options

The following table lists the options available with the TABLES statement. Descriptions follow in alphabetical order.

Table 2.9. TABLES Statement Options

Option	Description
Control Statistical Analysis	
AGREE	requests tests and measures of classification agreement
ALL	requests tests and measures of association produced by CHISQ, MEASURES, and CMH
ALPHA=	sets the confidence level for confidence limits
BDT	requests Tarone's adjustment for the Breslow-Day test
BINOMIAL	requests binomial proportion, confidence limits and test for one-way tables
BINOMIALC	requests BINOMIAL statistics with a continuity correction
CHISQ	requests chi-square tests and measures of association based on chi-square
CL	requests confidence limits for the MEASURES statistics
CMH	requests all Cochran-Mantel-Haenszel statistics
CMH1	requests the CMH correlation statistic, and adjusted relative risks and odds ratios
CMH2	requests CMH correlation and row mean scores (ANOVA) statistics, and adjusted relative risks and odds ratios
CONVERGE=	specifies convergence criterion to compute polychoric correlation
FISHER	requests Fisher's exact test for tables larger than 2×2
JT	requests Jonckheere-Terpstra test
MAXITER=	specifies maximum number of iterations to compute polychoric correlation
MEASURES	requests measures of association and their asymptotic standard errors
MISSING	treats missing values as nonmissing
PLCORR	requests polychoric correlation
RELRISK	requests relative risk measures for 2×2 tables
RISKDIFF	requests risks and risk differences for 2×2 tables
RISKDIFFC	requests RISKDIFF statistics with a continuity correction
SCORES=	specifies the type of row and column scores
TESTF=	specifies expected frequencies for a one-way table chi-square test
TESTP=	specifies expected proportions for a one-way table chi-square test
TREND	requests Cochran-Armitage test for trend

Table 2.9. (continued)

Option	Description
Control Additional Table Information	
CELLCHI2	displays each cell's contribution to the total Pearson chi-square statistic
CUMCOL	displays the cumulative column percentage in each cell
DEVIATION	displays the deviation of the cell frequency from the expected value for each cell
EXPECTED	displays the expected cell frequency for each cell
MISSPRINT	displays missing value frequencies
SPARSE	lists all possible combinations of variable levels even when a combination does not occur
TOTPCT	displays percentage of total frequency on *n*-way tables when $n > 2$
Control Displayed Output	
CONTENTS=	specifies the HTML contents link for crosstabulation tables
CROSSLIST	displays crosstabulation tables in ODS column format
FORMAT=	formats the frequencies in crosstabulation tables
LIST	displays two-way to *n*-way tables in list format
NOCOL	suppresses display of the column percentage for each cell
NOCUM	suppresses display of cumulative frequencies and cumulative percentages in one-way frequency tables and in list format
NOFREQ	suppresses display of the frequency count for each cell
NOPERCENT	suppresses display of the percentage, row percentage, and column percentage in crosstabulation tables, or percentages and cumulative percentages in one-way frequency tables and in list format
NOPRINT	suppresses display of tables but displays statistics
NOROW	suppresses display of the row percentage for each cell
NOSPARSE	suppresses zero cell frequencies in the list display and in the OUT= data set when ZEROS is specified
NOWARN	suppresses log warning message for the chi-square test
PRINTKWT	displays kappa coefficient weights
SCOROUT	displays the row and the column scores
Create an Output Data Set	
OUT=	specifies an output data set to contain variable values and frequency counts
OUTCUM	includes the cumulative frequency and cumulative percentage in the output data set for one-way tables
OUTEXPECT	includes the expected frequency of each cell in the output data set
OUTPCT	includes the percentage of column frequency, row frequency, and two-way table frequency in the output data set

You can specify the following options in a TABLES statement.

AGREE $<$ **(WT=FC)** $>$

requests tests and measures of classification agreement for square tables. The AGREE option provides McNemar's test for 2×2 tables and Bowker's test of symmetry for tables with more than two response categories. The AGREE option also produces the simple kappa coefficient, the weighted kappa coefficient, the asymptotic standard errors for the simple and weighted kappas, and the corresponding confidence limits. When there are multiple strata, the AGREE option provides overall simple and weighted kappas as well as tests for equal kappas among strata. When there are multiple strata and two response categories, PROC FREQ computes Cochran's Q test. For more information, see the section "Tests and Measures of Agreement" on page 127.

The (WT=FC) specification requests that PROC FREQ use Fleiss-Cohen weights to compute the weighted kappa coefficient. By default, PROC FREQ uses Cicchetti-Allison weights. See the section "Weighted Kappa Coefficient" on page 130 for more information. You can specify the PRINTKWT option to display the kappa coefficient weights.

AGREE statistics are computed only for square tables, where the number of rows equals the number of columns. If your table is not square due to observations with zero weights, you can use the ZEROS option in the WEIGHT statement to include these observations. For more details, see the section "Tables with Zero Rows and Columns" on page 133.

ALL

requests all of the tests and measures that are computed by the CHISQ, MEASURES, and CMH options. The number of CMH statistics computed can be controlled by the CMH1 and CMH2 options.

ALPHA=α

specifies the level of confidence limits. The value of the ALPHA= option must be between 0 and 1, and the default is 0.05. A confidence level of α produces $100(1-\alpha)$% confidence limits. The default of ALPHA=0.05 produces 95% confidence limits.

ALPHA= applies to confidence limits requested by TABLES statement options. There is a separate ALPHA= option in the EXACT statement that sets the level of confidence limits for Monte Carlo estimates of exact *p*-values, which are requested in the EXACT statement.

BDT

requests Tarone's adjustment in the Breslow-Day test for homogeneity of odds ratios. (You must specify the CMH option to compute the Breslow-Day test.) See the section "Breslow-Day Test for Homogeneity of the Odds Ratios" on page 142 for more information.

BINOMIAL $<$ **(P=** *value***) | (LEVEL=** *level-number* **|** *level-value***)** $>$

requests the binomial proportion for one-way tables. The BINOMIAL option also provides the asymptotic standard error, asymptotic and exact confidence intervals, and the asymptotic test for the binomial proportion. To request an exact test for the binomial proportion, use the BINOMIAL option in the EXACT statement.

To specify the null hypothesis proportion for the test, use P=. If you omit P=*value*, PROC FREQ uses 0.5 as the default for the test. By default, BINOMIAL computes the proportion of observations for the first variable level that appears in the output. To specify a different level, use LEVEL=*level-number* or LEVEL=*level-value*, where *level-number* is the variable level's number or order in the output, and *level-value* is the formatted value of the variable level.

To include a continuity correction in the asymptotic confidence interval and test, use the BINOMIALC option instead of the BINOMIAL option.

See the section "Binomial Proportion" on page 118 for more information.

BINOMIALC $<$ **(P=** *value***) | (LEVEL=** *level-number* **|** *level-value***)** $>$

requests the BINOMIAL option statistics for one-way tables, and includes a continuity correction in the asymptotic confidence interval and the asymptotic test. The BINOMIAL option statistics include the binomial proportion, the asymptotic standard error, asymptotic and exact confidence intervals, and the asymptotic test for the binomial proportion. To request an exact test for the binomial proportion, use the BINOMIAL option in the EXACT statement.

To specify the null hypothesis proportion for the test, use P=. If you omit P=*value*, PROC FREQ uses 0.5 as the default for the test. By default BINOMIALC computes the proportion of observations for the first variable level that appears in the output. To specify a different level, use LEVEL=*level-number* or LEVEL=*level-value*, where *level-number* is the variable level's number or order in the output, and *level-value* is the formatted value of the variable level.

See the section "Binomial Proportion" on page 118 for more information.

CELLCHI2

displays each crosstabulation table cell's contribution to the total Pearson chi-square statistic, which is computed as

$$\frac{(\mathit{frequency} - \mathit{expected})^2}{\mathit{expected}}$$

The CELLCHI2 option has no effect for one-way tables or for tables that are displayed with the LIST option.

CHISQ

requests chi-square tests of homogeneity or independence and measures of association based on chi-square. The tests include the Pearson chi-square, likelihood-ratio chi-square, and Mantel-Haenszel chi-square. The measures include the phi coefficient, the contingency coefficient, and Cramer's V. For 2×2 tables, the CHISQ option includes Fisher's exact test and the continuity-adjusted chi-square. For one-way tables, the CHISQ option requests a chi-square goodness-of-fit test for equal proportions. If you specify the null hypothesis proportions with the TESTP= option, then PROC FREQ computes a chi-square goodness-of-fit test for the specified proportions. If you specify null hypothesis frequencies with the TESTF= option, PROC

FREQ computes a chi-square goodness-of-fit test for the specified frequencies. See the section "Chi-Square Tests and Statistics" on page 103 for more information.

CL

requests confidence limits for the MEASURES statistics. If you omit the MEASURES option, the CL option invokes MEASURES. The FREQ procedure determines the confidence coefficient using the ALPHA= option, which, by default, equals 0.05 and produces 95% confidence limits.

For more information, see the section "Confidence Limits" on page 109.

CMH

requests Cochran-Mantel-Haenszel statistics, which test for association between the row and column variables after adjusting for the remaining variables in a multiway table. In addition, for 2×2 tables, PROC FREQ computes the adjusted Mantel-Haenszel and logit estimates of the odds ratios and relative risks and the corresponding confidence limits. For the stratified 2×2 case, PROC FREQ computes the Breslow-Day test for homogeneity of odds ratios. (To request Tarone's adjustment for the Breslow-Day test, use the BDT option.) The CMH1 and CMH2 options control the number of CMH statistics that PROC FREQ computes. For more information, see the section "Cochran-Mantel-Haenszel Statistics" on page 134.

CMH1

requests the Cochran-Mantel-Haenszel correlation statistic and, for 2×2 tables, the adjusted Mantel-Haenszel and logit estimates of the odds ratios and relative risks and the corresponding confidence limits. For the stratified 2×2 case, PROC FREQ computes the Breslow-Day test for homogeneity of odds ratios. Except for 2×2 tables, the CMH1 option requires less memory than the CMH option, which can require an enormous amount for large tables.

CMH2

requests the Cochran-Mantel-Haenszel correlation statistic, row mean scores (ANOVA) statistic, and, for 2×2 tables, the adjusted Mantel-Haenszel and logit estimates of the odds ratios and relative risks and the corresponding confidence limits. For the stratified 2×2 case, PROC FREQ computes the Breslow-Day test for homogeneity of odds ratios. Except for tables with two columns, the CMH2 option requires less memory than the CMH option, which can require an enormous amount for large tables.

CONTENTS=*link-text*

specifies the text for the HTML contents file links to crosstabulation tables. For information on HTML output, refer to the *SAS Output Delivery System User's Guide*. The CONTENTS= option affects only the HTML contents file, and not the HTML body file.

If you omit the CONTENTS= option, by default, the HTML link text for crosstabulation tables is "Cross-Tabular Freq Table."

Note that links to all crosstabulation tables produced by a single TABLES statement use the same text. To specify different text for different crosstabulation table links, request the tables in separate TABLES statements and use the CONTENTS= option in each TABLES statement.

The CONTENTS= option affects only links to crosstabulation tables. It does not affect links to other PROC FREQ tables. To specify link text for any other PROC FREQ table, you can use PROC TEMPLATE to create a customized table definition. The CONTENTS_LABEL attribute in the DEFINE TABLE statement of PROC TEMPLATE specifies the contents file link for the table. For detailed information, refer to the chapter titled "The TEMPLATE Procedure" in the *SAS Output Delivery System User's Guide*.

CONVERGE=*value*

specifies the convergence criterion for computing the polychoric correlation when you specify the PLCORR option. The value of the CONVERGE= option must be a positive number; by default, CONVERGE=0.0001. Iterative computation of the polychoric correlation stops when the convergence measure falls below the value of the CONVERGE= option or when the number of iterations exceeds the value specified in the MAXITER= option, whichever happens first.

See the section "Polychoric Correlation" on page 116 for more information.

CROSSLIST

displays crosstabulation tables in ODS column format, instead of the default crosstabulation cell format. In a CROSSLIST table display, the rows correspond to the crosstabulation table cells, and the columns correspond to descriptive statistics such as Frequency, Percent, and so on. See the section "Multiway Tables" on page 152 for details on the contents of the CROSSLIST table.

The CROSSLIST table displays the same information as the default crosstabulation table, but uses an ODS column format instead of the table cell format. Unlike the default crosstabulation table, the CROSSLIST table has a table definition that you can customize with PROC TEMPLATE. For more information, refer to the chapter titled "The TEMPLATE Procedure" in the *SAS Output Delivery System User's Guide*.

You can control the contents of a CROSSLIST table with the same options available for the default crosstabulation table. These include the NOFREQ, NOPERCENT, NOROW, and NOCOL options. You can request additional information in a CROSSLIST table with the CELLCHI2, DEVIATION, EXPECTED, MISSPRINT, and TOTPCT options.

The FORMAT= option and the CUMCOL option have no effect for CROSSLIST tables. You cannot specify both the LIST option and the CROSSLIST option in the same TABLES statement.

You can use the NOSPARSE option to suppress display of variable levels with zero frequency in CROSSLIST tables. By default for CROSSLIST tables, PROC FREQ displays all levels of the column variable within each level of the row variable, including any column variable levels with zero frequency for that row. And for multiway tables displayed with the CROSSLIST option, the procedure displays all levels of the row variable for each stratum of the table by default, including any row variable levels with zero frequency for the stratum.

CUMCOL
displays the cumulative column percentages in the cells of the crosstabulation table.

DEVIATION
displays the deviation of the cell frequency from the expected frequency for each cell of the crosstabulation table. The DEVIATION option is valid for contingency tables but has no effect on tables produced with the LIST option.

EXPECTED
displays the expected table cell frequencies under the hypothesis of independence (or homogeneity). The EXPECTED option is valid for crosstabulation tables but has no effect on tables produced with the LIST option.

FISHER | EXACT
requests Fisher's exact test for tables that are larger than 2×2. This test is also known as the Freeman-Halton test. For more information, see the section "Fisher's Exact Test" on page 106 and the "EXACT Statement" section on page 77.

If you omit the CHISQ option in the TABLES statement, the FISHER option invokes CHISQ. You can also request Fisher's exact test by specifying the FISHER option in the EXACT statement.

CAUTION: For tables with many rows or columns or with large total frequency, PROC FREQ may require a large amount of time or memory to compute exact *p*-values. See the section "Computational Resources" on page 145 for more information.

FORMAT=*format-name*
specifies a format for the following crosstabulation table cell values: frequency, expected frequency, and deviation. PROC FREQ also uses this format to display the total row and column frequencies for crosstabulation tables.

You can specify any standard SAS numeric format or a numeric format defined with the FORMAT procedure. The format length must not exceed 24. If you omit FORMAT=, by default, PROC FREQ uses the BEST6. format to display frequencies less than 1E6, and the BEST7. format otherwise.

To change formats for all other FREQ tables, you can use PROC TEMPLATE. For information on this procedure, refer to the chapter titled "The TEMPLATE Procedure" in the *SAS Output Delivery System User's Guide*.

JT
performs the Jonckheere-Terpstra test. For more information, see the section "Jonckheere-Terpstra Test" on page 125.

LIST
displays two-way to n-way tables in a list format rather than as crosstabulation tables. PROC FREQ ignores the LIST option when you request statistical tests or measures of association.

MAXITER=*number*

specifies the maximum number of iterations for computing the polychoric correlation when you specify the PLCORR option. The value of the MAXITER= option must be a positive integer; by default, MAXITER=20. Iterative computation of the polychoric correlation stops when the number of iterations exceeds the value of the MAXITER= option, or when the convergence measure falls below the value of the CONVERGE= option, whichever happens first. For more information see the section "Polychoric Correlation" on page 116.

MEASURES

requests several measures of association and their asymptotic standard errors (ASE). The measures include gamma, Kendall's tau-b, Stuart's tau-c, Somers' $D(C|R)$, Somers' $D(R|C)$, the Pearson and Spearman correlation coefficients, lambda (symmetric and asymmetric), uncertainty coefficients (symmetric and asymmetric). To request confidence limits for these measures of association, you can specify the CL option.

For 2×2 tables, the MEASURES option also provides the odds ratio, column 1 relative risk, column 2 relative risk, and the corresponding confidence limits. Alternatively, you can obtain the odds ratio and relative risks, without the other measures of association, by specifying the RELRISK option.

For more information, see the section "Measures of Association" on page 108.

MISSING

treats missing values as nonmissing and includes them in calculations of percentages and other statistics.

For more information, see the section "Missing Values" on page 100.

MISSPRINT

displays missing value frequencies for all tables, even though PROC FREQ does not use the frequencies in the calculation of statistics. For more information, see the section "Missing Values" on page 100.

NOCOL

suppresses the display of column percentages in cells of the crosstabulation table.

NOCUM

suppresses the display of cumulative frequencies and cumulative percentages for one-way frequency tables and for crosstabulation tables in list format.

NOFREQ

suppresses the display of cell frequencies for crosstabulation tables. This also suppresses frequencies for row totals.

NOPERCENT

suppresses the display of cell percentages, row total percentages, and column total percentages for crosstabulation tables. For one-way frequency tables and crosstabulation tables in list format, the NOPERCENT option suppresses the display of percentages and cumulative percentages.

NOPRINT

suppresses the display of frequency and crosstabulation tables but displays all requested tests and statistics. Use the NOPRINT option in the PROC FREQ statement to suppress the display of all tables.

NOROW

suppresses the display of row percentages in cells of the crosstabulation table.

NOSPARSE

requests that PROC FREQ not invoke the SPARSE option when you specify the ZEROS option in the WEIGHT statement. The NOSPARSE option suppresses the display of cells with a zero frequency count in the list output, and it also omits them from the OUT= data set. By default, the ZEROS option invokes the SPARSE option, which displays table cells with a zero frequency count in the LIST output and includes them in the OUT= data set. For more information, see the description of the ZEROS option.

For CROSSLIST tables, the NOSPARSE option suppresses display of variable levels with zero frequency. By default for CROSSLIST tables, PROC FREQ displays all levels of the column variable within each level of the row variable, including any column variable levels with zero frequency for that row. And for multiway tables displayed with the CROSSLIST option, the procedure displays all levels of the row variable for each stratum of the table by default, including any row variable levels with zero frequency for the stratum.

NOWARN

suppresses the log warning message that the asymptotic chi-square test may not be valid. By default, PROC FREQ displays this log message when more than 20 percent of the table cells have expected frequencies less than five.

OUT=*SAS-data-set*

names the output data set that contains variable values and frequency counts. The variable COUNT contains the frequencies and the variable PERCENT contains the percentages. If more than one table request appears in the TABLES statement, the contents of the data set correspond to the last table request in the TABLES statement. For more information, see the section "Output Data Sets" on page 148 and see the following descriptions for the options OUTCUM, OUTEXPECT, and OUTPCT.

OUTCUM

includes the cumulative frequency and the cumulative percentage for one-way tables in the output data set when you specify the OUT= option in the TABLES statement. The variable CUM_FREQ contains the cumulative frequency for each level of the analysis variable, and the variable CUM_PCT contains the cumulative percentage for each level. The OUTCUM option has no effect for two-way or multiway tables.

For more information, see the section "Output Data Sets" on page 148.

OUTEXPECT

includes the expected frequency in the output data set for crosstabulation tables when you specify the OUT= option in the TABLES statement. The variable EXPECTED contains the expected frequency for each table cell. The EXPECTED option is valid for two-way or multiway tables, and has no effect for one-way tables.

For more information, see the section "Output Data Sets" on page 148.

OUTPCT

includes the following additional variables in the output data set when you specify the OUT= option in the TABLES statement for crosstabulation tables:

PCT_COL	the percentage of column frequency
PCT_ROW	the percentage of row frequency
PCT_TABL	the percentage of stratum frequency, for n-way tables where $n > 2$

The OUTPCT option is valid for two-way or multiway tables, and has no effect for one-way tables.

For more information, see the section "Output Data Sets" on page 148.

PLCORR

requests the polychoric correlation coefficient. For 2×2 tables, this statistic is more commonly known as the tetrachoric correlation coefficient, and it is labeled as such in the displayed output. If you omit the MEASURES option, the PLCORR option invokes MEASURES. For more information, see the section "Polychoric Correlation" on page 116 and the descriptions for the CONVERGE= and MAXITER= options in this list.

PRINTKWT

displays the weights PROC FREQ uses to compute the weighted kappa coefficient. You must also specify the AGREE option, which requests the weighted kappa coefficient. You can specify (WT=FC) with the AGREE option to request Fleiss-Cohen weights. By default, PROC FREQ uses Cicchetti-Allison weights.

See the section "Weighted Kappa Coefficient" on page 130 for more information.

RELRISK

requests relative risk measures and their confidence limits for 2×2 tables. These measures include the odds ratio and the column 1 and 2 relative risks. For more information, see the section "Odds Ratio and Relative Risks for 2 x 2 Tables" on page 122. You can also obtain the RELRISK measures by specifying the MEASURES option, which produces other measures of association in addition to the relative risks.

RISKDIFF

requests column 1 and 2 risks (or binomial proportions), risk differences, and their confidence limits for 2×2 tables. See the section "Risks and Risk Differences" on page 120 for more information.

RISKDIFFC

requests the RISKDIFF option statistics for 2×2 tables, and includes a continuity correction in the asymptotic confidence limits. The RISKDIFF option statistics include the column 1 and 2 risks (or binomial proportions), risk differences, and their confidence limits. See the section “Risks and Risk Differences” on page 120 for more information.

SCORES=*type*

specifies the type of row and column scores that PROC FREQ uses with the Mantel-Haenszel chi-square, Pearson correlation, Cochran-Armitage test for trend, weighted kappa coefficient, and Cochran-Mantel-Haenszel statistics, where *type* is one of the following (the default is SCORE=TABLE):

- MODRIDIT
- RANK
- RIDIT
- TABLE

By default, the row or column scores are the integers 1,2,... for character variables and the actual variable values for numeric variables. Using other types of scores yields nonparametric analyses. For more information, see the section “Scores” on page 102.

To display the row and column scores, you can use the SCOROUT option.

SCOROUT

displays the row and the column scores. You specify the score type with the SCORES= option. PROC FREQ uses the scores when it calculates the Mantel-Haenszel chi-square, Pearson correlation, Cochran-Armitage test for trend, weighted kappa coefficient, or Cochran-Mantel-Haenszel statistics. The SCOROUT option displays the row and column scores only when statistics are computed for two-way tables. To store the scores in an output data set, use the Output Delivery System.

For more information, see the section “Scores” on page 102.

SPARSE

lists all possible combinations of the variable values for an *n*-way table when $n > 1$, even if a combination does not occur in the data. The SPARSE option applies only to crosstabulation tables displayed in list format and to the OUT= output data set. Otherwise, if you do not use the LIST option or the OUT= option, the SPARSE option has no effect.

When you specify the SPARSE and LIST options, PROC FREQ displays all combinations of variable variables in the table listing, including those values with a frequency count of zero. By default, without the SPARSE option, PROC FREQ does not display zero-frequency values in list output. When you use the SPARSE and OUT= options, PROC FREQ includes empty crosstabulation table cells in the output data set. By default, PROC FREQ does not include zero-frequency table cells in the output data set.

For more information, see the section “Missing Values” on page 100.

TESTF=(*values*)

specifies the null hypothesis frequencies for a one-way chi-square test for specified frequencies. You can separate *values* with blanks or commas. The sum of the frequency values must equal the total frequency for the one-way table. The number of TESTF= values must equal the number of variable levels in the one-way table. List these values in the order in which the corresponding variable levels appear in the output. If you omit the CHISQ option, the TESTF= option invokes CHISQ.

For more information, see the section "Chi-Square Test for One-Way Tables" on page 104.

TESTP=(*values*)

specifies the null hypothesis proportions for a one-way chi-square test for specified proportions. You can separate *values* with blanks or commas. Specify *values* in probability form as numbers between 0 and 1, where the proportions sum to 1. Or specify *values* in percentage form as numbers between 0 and 100, where the percentages sum to 100. The number of TESTP= values must equal the number of variable levels in the one-way table. List these values in the order in which the corresponding variable levels appear in the output. If you omit the CHISQ option, the TESTP= option invokes CHISQ.

For more information, see the section "Chi-Square Test for One-Way Tables" on page 104.

TOTPCT

displays the percentage of total frequency in crosstabulation tables, for n-way tables where $n > 2$. This percentage is also available with the LIST option or as the PERCENT variable in the OUT= output data set.

TREND

performs the Cochran-Armitage test for trend. The table must be $2 \times C$ or $R \times 2$. For more information, see the section "Cochran-Armitage Test for Trend" on page 124.

TEST Statement

TEST *options* **;**

The TEST statement requests asymptotic tests for the specified measures of association and measures of agreement. You must use a TABLES statement with the TEST statement.

options

specify the statistics for which to provide asymptotic tests. The available statistics are those measures of association and agreement listed in Table 2.10 . The option names are identical to those in the TABLES statement and the OUTPUT statement. You can request all available tests for groups of statistics by using group options MEASURES or AGREE. Or you can request tests individually by using one of the options shown in Table 2.10 .

For each measure of association or agreement that you specify, the TEST statement provides an asymptotic test that the measure equals zero. When you request an asymptotic test, PROC FREQ gives the asymptotic standard error under the null hypothesis, the test statistic, and the *p*-values. Additionally, PROC FREQ reports the confidence limits for that measure. The ALPHA= option in the TABLES statement determines the confidence level, which, by default, equals 0.05 and provides 95% confidence limits. For more information, see the sections "Asymptotic Tests" on page 109 and "Confidence Limits" on page 109, and see "Statistical Computations" beginning on page 102 for sections describing the individual measures.

In addition to these asymptotic tests, exact tests for selected measures of association and agreement are available with the EXACT statement. See the section "EXACT Statement" on page 77 for more information.

Table 2.10. TEST Statement Options and Required TABLES Statement Options

Option	Asymptotic Tests Computed	Required TABLES Statement Option
AGREE	simple kappa coefficient and weighted kappa coefficient	AGREE
GAMMA	gamma	ALL or MEASURES
KAPPA	simple kappa coefficient	AGREE
KENTB	Kendall's tau-b	ALL or MEASURES
MEASURES	gamma, Kendall's tau-b, Stuart's tau-c, Somers' $D(C\|R)$, Somers' $D(R\|C)$, the Pearson correlation, and the Spearman correlation	ALL or MEASURES
PCORR	Pearson correlation coefficient	ALL or MEASURES
SCORR	Spearman correlation coefficient	ALL or MEASURES
SMDCR	Somers' $D(C\|R)$	ALL or MEASURES
SMDRC	Somers' $D(R\|C)$	ALL or MEASURES
STUTC	Stuart's tau-c	ALL or MEASURES
WTKAP	weighted kappa coefficient	AGREE

WEIGHT Statement

WEIGHT *variable* < **/** *option* > **;**

The WEIGHT statement specifies a numeric *variable* with a value that represents the frequency of the observation. The WEIGHT statement is most commonly used to input cell count data. See the "Inputting Frequency Counts" section on page 98 for more information. If you use the WEIGHT statement, PROC FREQ assumes that an observation represents *n* observations, where *n* is the value of *variable*. The value of the weight variable need not be an integer. When a weight value is missing, PROC FREQ ignores the corresponding observation. When a weight value is zero, PROC FREQ ignores the corresponding observation unless you specify the ZEROS option, which includes observations with zero weights. If a WEIGHT statement does not appear, each observation has a default weight of 1. The sum of the weight variable values represents the total number of observations.

If any value of the weight variable is negative, PROC FREQ displays the frequencies (as measured by the weighted values) but does not compute percentages and other statistics. If you create an output data set using the OUT= option in the TABLES statement, PROC FREQ creates the PERCENT variable and assigns a missing value for each observation. PROC FREQ also assigns missing values to the variables that the OUTEXPECT and OUTPCT options create. You cannot create an output data set using the OUTPUT statement since statistics are not computed when there are negative weights.

Option

ZEROS

includes observations with zero weight values. By default, PROC FREQ ignores observations with zero weights.

If you specify the ZEROS option, frequency and and crosstabulation tables display any levels corresponding to observations with zero weights. Without the ZEROS option, PROC FREQ does not process observations with zero weights, and so does not display levels that contain only observations with zero weights.

With the ZEROS option, PROC FREQ includes levels with zero weights in the chi-square goodness-of-fit test for one-way tables. Also, PROC FREQ includes any levels with zero weights in binomial computations for one-way tables. This enables computation of binomial estimates and tests when there are no observations with positive weights in the specified level.

For two-way tables, the ZEROS option enables computation of kappa statistics when there are levels containing no observations with positive weight. For more information, see the section "Tables with Zero Rows and Columns" on page 133.

Note that even with the ZEROS option, PROC FREQ does not compute the CHISQ or MEASURES statistics for two-way tables when the table has a zero row or zero column, because most of these statistics are undefined in this case.

The ZEROS option invokes the SPARSE option in the TABLES statement, which includes table cells with a zero frequency count in the list output and the OUT= data set. By default, without the SPARSE option, PROC FREQ does not include zero frequency cells in the list output or in the OUT= data set. If you specify the ZEROS option in the WEIGHT statement but do not want the SPARSE option, you can specify the NOSPARSE option in the TABLES statement.

Details

Inputting Frequency Counts

PROC FREQ can use either raw data or cell count data to produce frequency and crosstabulation tables. *Raw data*, also known as case-record data, report the data as one record for each subject or sample member. *Cell count data* report the data as a table, listing all possible combinations of data values along with the frequency counts. This way of presenting data often appears in published results.

The following DATA step statements store raw data in a SAS data set:

```
data Raw;
   input Subject $ R C @@;
   datalines;
01 1 1  02 1 1  03 1 1  04 1 1  05 1 1
06 1 2  07 1 2  08 1 2  09 2 1  10 2 1
11 2 1  12 2 1  13 2 2  14 2 2  14 2 2
;
```

You can store the same data as cell counts using the following DATA step statements:

```
data CellCounts;
   input R C Count @@;
   datalines;
1 1 5   1 2 3
2 1 4   2 2 3
;
```

The variable R contains the values for the rows, and the variable C contains the values for the columns. The Count variable contains the cell count for each row and column combination.

Both the Raw data set and the CellCounts data set produce identical frequency counts, two-way tables, and statistics. With the CellCounts data set, you must use a WEIGHT statement to specify that the Count variable contains cell counts. For example, to create a two-way crosstabulation table, submit the following statements:

```
proc freq data=CellCounts;
   weight Count;
   tables R*C;
run;
```

Grouping with Formats

PROC FREQ groups a variable's values according to its formatted values. If you assign a format to a variable with a FORMAT statement, PROC FREQ formats the variable values before dividing observations into the levels of a frequency or crosstabulation table.

For example, suppose that a variable X has the values 1.1, 1.4, 1.7, 2.1, and 2.3. Each of these values appears as a level in the frequency table. If you decide to round each value to a single digit, include the following statement in the PROC FREQ step:

```
format X 1.;
```

Now the table lists the frequency count for formatted level 1 as two and formatted level 2 as three.

PROC FREQ treats formatted character variables in the same way. The formatted values are used to group the observations into the levels of a frequency table or crosstabulation table. PROC FREQ uses the entire value of a character format to classify an observation.

You can also use the FORMAT statement to assign formats that were created with the FORMAT procedure to the variables. User-written formats determine the number of levels for a variable and provide labels for a table. If you use the same data with different formats, then you can produce frequency counts and statistics for different classifications of the variable values.

When you use PROC FORMAT to create a user-written format that combines missing and nonmissing values into one category, PROC FREQ treats the entire category of formatted values as missing. For example, a questionnaire codes 1 as yes, 2 as no, and 8 as a no answer. The following PROC FORMAT step creates a user-written format:

```
proc format;
   value Questfmt 1  ='Yes'
                  2  ='No'
                  8,.='Missing';
run;
```

When you use a FORMAT statement to assign Questfmt. to a variable, the variable's frequency table no longer includes a frequency count for the response of 8. You must use the MISSING or MISSPRINT option in the TABLES statement to list the frequency for no answer. The frequency count for this level includes observations with either a value of 8 or a missing value (.).

The frequency or crosstabulation table lists the values of both character and numeric variables in ascending order based on internal (unformatted) variable values unless you change the order with the ORDER= option. To list the values in ascending order by formatted values, use ORDER=FORMATTED in the PROC FREQ statement.

For more information on the FORMAT statement, refer to *SAS Language Reference: Concepts*.

Missing Values

By default, PROC FREQ excludes missing values before it constructs the frequency and crosstabulation tables. PROC FREQ also excludes missing values before computing statistics. However, the total frequency of observations with missing values is displayed below each table. The following options change the way in which PROC FREQ handles missing values:

MISSPRINT includes missing value frequencies in frequency or crosstabulation tables.

MISSING includes missing values in percentage and statistical calculations.

The OUT= option in the TABLES statement includes an observation in the output data set that contains the frequency of missing values. The NMISS option in the OUTPUT statement creates a variable in the output data set that contains the number of missing values.

Figure 2.7 shows three ways in which PROC FREQ handles missing values. The first table uses the default method; the second table uses the MISSPRINT option; and the third table uses the MISSING option.

```
                          *** Default ***

                        The FREQ Procedure

                                           Cumulative    Cumulative
           A    Frequency     Percent     Frequency       Percent
           -------------------------------------------------------
           1           2       50.00             2         50.00
           2           2       50.00             4        100.00

                      Frequency Missing = 2


                     *** MISSPRINT Option ***

                        The FREQ Procedure

                                           Cumulative    Cumulative
           A    Frequency     Percent     Frequency       Percent
           -------------------------------------------------------
           .           2         .               .           .
           1           2       50.00             2         50.00
           2           2       50.00             4        100.00

                      Frequency Missing = 2


                      *** MISSING Option ***

                        The FREQ Procedure

                                           Cumulative    Cumulative
           A    Frequency     Percent     Frequency       Percent
           -------------------------------------------------------
           .           2       33.33             2         33.33
           1           2       33.33             4         66.67
           2           2       33.33             6        100.00
```

Figure 2.7. Missing Values in Frequency Tables

When a combination of variable values for a crosstabulation is missing, PROC FREQ assigns zero to the frequency count for the table cell. By default, PROC FREQ omits missing combinations in list format and in the output data set that is created in a TABLES statement. To include the missing combinations, use the SPARSE option with the LIST or OUT= option in the TABLES statement.

PROC FREQ treats missing BY variable values like any other BY variable value. The missing values form a separate BY group. When the value of a WEIGHT variable is missing, PROC FREQ excludes the observation from the analysis.

Statistical Computations

Definitions and Notation

In this chapter, a two-way table represents the crosstabulation of variables X and Y. Let the rows of the table be labeled by the values X_i, $i = 1, 2, \ldots, R$, and the columns by Y_j, $j = 1, 2, \ldots, C$. Let n_{ij} denote the cell frequency in the ith row and the jth column and define the following:

$$
\begin{aligned}
n_{i\cdot} &= \sum_j n_{ij} \quad \text{(row totals)} \\
n_{\cdot j} &= \sum_i n_{ij} \quad \text{(column totals)} \\
n &= \sum_i \sum_j n_{ij} \quad \text{(overall total)} \\
p_{ij} &= n_{ij}/n \quad \text{(cell percentages)} \\
p_{i\cdot} &= n_{i\cdot}/n \quad \text{(row percentages)} \\
p_{\cdot j} &= n_{\cdot j}/n \quad \text{(column percentages)} \\
R_i &= \text{score for row } i \\
C_j &= \text{score for column } j \\
\bar{R} &= \sum_i n_{i\cdot} R_i / n \quad \text{(average row score)} \\
\bar{C} &= \sum_j n_{\cdot j} C_j / n \quad \text{(average column score)} \\
A_{ij} &= \sum_{k>i} \sum_{l>j} n_{kl} + \sum_{k<i} \sum_{l<j} n_{kl} \\
D_{ij} &= \sum_{k>i} \sum_{l<j} n_{kl} + \sum_{k<i} \sum_{l>j} n_{kl} \\
P &= \sum_i \sum_j n_{ij} A_{ij} \quad \text{(twice the number of concordances)} \\
Q &= \sum_i \sum_j n_{ij} D_{ij} \quad \text{(twice the number of discordances)}
\end{aligned}
$$

Scores

PROC FREQ uses scores for the variable values when computing the Mantel-Haenszel chi-square, Pearson correlation, Cochran-Armitage test for trend, weighted kappa coefficient, and Cochran-Mantel-Haenszel statistics. The SCORES= option in the TABLES statement specifies the score type that PROC FREQ uses. The available score types are TABLE, RANK, RIDIT, and MODRIDIT scores. The default score type is TABLE.

For numeric variables, table scores are the values of the row and column levels. If the row or column variables are formatted, then the table score is the internal numeric value corresponding to that level. If two or more numeric values are classified into the same formatted level, then the internal numeric value for that level is the smallest of these values. For character variables, table scores are defined as the row numbers and column numbers (that is, 1 for the first row, 2 for the second row, and so on).

Rank scores, which you can use to obtain nonparametric analyses, are defined by

$$\begin{aligned} \text{Row scores:} \quad R1_i &= \sum_{k<i} n_{k\cdot} + (n_{i\cdot} + 1)/2 \qquad i = 1, 2, \ldots, R \\ \text{Column scores:} \quad C1_j &= \sum_{l<j} n_{\cdot l} + (n_{\cdot j} + 1)/2 \qquad j = 1, 2, \ldots, C \end{aligned}$$

Note that rank scores yield midranks for tied values.

Ridit scores (Bross 1958; Mack and Skillings 1980) also yield nonparametric analyses, but they are standardized by the sample size. Ridit scores are derived from rank scores as

$$\begin{aligned} R2_i &= R1_i / n \\ C2_j &= C1_j / n \end{aligned}$$

Modified ridit (MODRIDIT) scores (van Elteren 1960; Lehmann 1975), which also yield nonparametric analyses, represent the expected values of the order statistics for the uniform distribution on (0,1). Modified ridit scores are derived from rank scores as

$$\begin{aligned} R3_i &= R1_i / (n+1) \\ C3_j &= C1_j / (n+1) \end{aligned}$$

Chi-Square Tests and Statistics

When you specify the CHISQ option in the TABLES statement, PROC FREQ performs the following chi-square tests for each two-way table: Pearson chi-square, continuity-adjusted chi-square for 2×2 tables, likelihood-ratio chi-square, Mantel-Haenszel chi-square, and Fisher's exact test for 2×2 tables. Also, PROC FREQ computes the following statistics derived from the Pearson chi-square: the phi coefficient, the contingency coefficient, and Cramer's V. PROC FREQ computes Fisher's exact test for general $R \times C$ tables when you specify the FISHER (or EXACT) option in the TABLES statement, or, equivalently, when you specify the FISHER option in the EXACT statement.

For one-way frequency tables, PROC FREQ performs a chi-square goodness-of-fit test when you specify the CHISQ option. The other chi-square tests and statistics described in this section are defined only for two-way tables and so are not computed for one-way frequency tables.

All the two-way test statistics described in this section test the null hypothesis of no association between the row variable and the column variable. When the sample size n is large, these test statistics are distributed approximately as chi-square when the null hypothesis is true. When the sample size is not large, exact tests may be useful. PROC FREQ computes exact tests for the following chi-square statistics when you specify the corresponding option in the EXACT statement: Pearson chi-square, likelihood-ratio chi-square, and Mantel-Haenszel chi-square. See the section "Exact Statistics" beginning on page 142 for more information.

Note that the Mantel-Haenszel chi-square statistic is appropriate only when both variables lie on an ordinal scale. The other chi-square tests and statistics in this section are appropriate for either nominal or ordinal variables. The following sections give the formulas that PROC FREQ uses to compute the chi-square tests and statistics. For further information on the formulas and on the applicability of each statistic, refer to Agresti (1996), Stokes, Davis, and Koch (1995), and the other references cited for each statistic.

Chi-Square Test for One-Way Tables

For one-way frequency tables, the CHISQ option in the TABLES statement computes a chi-square goodness-of-fit test. Let C denote the number of classes, or levels, in the one-way table. Let f_i denote the frequency of class i (or the number of observations in class i) for $i = 1, 2, ..., C$. Then PROC FREQ computes the chi-square statistic as

$$Q_P = \sum_{i=1}^{C} \frac{(f_i - e_i)^2}{e_i}$$

where e_i is the expected frequency for class i under the null hypothesis.

In the test for equal proportions, which is the default for the CHISQ option, the null hypothesis specifies equal proportions of the total sample size for each class. Under this null hypothesis, the expected frequency for each class equals the total sample size divided by the number of classes,

$$e_i = n \; / \; C \qquad \text{for } i = 1, 2, \ldots, C$$

In the test for specified frequencies, which PROC FREQ computes when you input null hypothesis frequencies using the TESTF= option, the expected frequencies are those TESTF= values. In the test for specified proportions, which PROC FREQ computes when you input null hypothesis proportions using the TESTP= option, the expected frequencies are determined from the TESTP= proportions p_i, as

$$e_i = p_i \times n \qquad \text{for } i = 1, 2, \ldots, C$$

Under the null hypothesis (of equal proportions, specified frequencies, or specified proportions), this test statistic has an asymptotic chi-square distribution, with $C - 1$ degrees of freedom. In addition to the asymptotic test, PROC FREQ computes the exact one-way chi-square test when you specify the CHISQ option in the EXACT statement.

Chi-Square Test for Two-Way Tables

The Pearson chi-square statistic for two-way tables involves the differences between the observed and expected frequencies, where the expected frequencies are computed under the null hypothesis of independence. The chi-square statistic is computed as

$$Q_P = \sum_i \sum_j \frac{(n_{ij} - e_{ij})^2}{e_{ij}}$$

where

$$e_{ij} = \frac{n_{i\cdot}\, n_{\cdot j}}{n}$$

When the row and column variables are independent, Q_P has an asymptotic chi-square distribution with $(R-1)(C-1)$ degrees of freedom. For large values of Q_P, this test rejects the null hypothesis in favor of the alternative hypothesis of general association. In addition to the asymptotic test, PROC FREQ computes the exact chi-square test when you specify the PCHI or CHISQ option in the EXACT statement.

For a 2×2 table, the Pearson chi-square is also appropriate for testing the equality of two binomial proportions or, for $R \times 2$ and $2 \times C$ tables, the homogeneity of proportions. Refer to Fienberg (1980).

Likelihood-Ratio Chi-Square Test

The likelihood-ratio chi-square statistic involves the ratios between the observed and expected frequencies. The statistic is computed as

$$G^2 = 2 \sum_i \sum_j n_{ij} \, \ln \left(\frac{n_{ij}}{e_{ij}} \right)$$

When the row and column variables are independent, G^2 has an asymptotic chi-square distribution with $(R - 1)(C - 1)$ degrees of freedom. In addition to the asymptotic test, PROC FREQ computes the exact test when you specify the LRCHI or CHISQ option in the EXACT statement.

Continuity-Adjusted Chi-Square Test

The continuity-adjusted chi-square statistic for 2×2 tables is similar to the Pearson chi-square, except that it is adjusted for the continuity of the chi-square distribution. The continuity-adjusted chi-square is most useful for small sample sizes. The use of the continuity adjustment is controversial; this chi-square test is more conservative, and more like Fisher's exact test, when your sample size is small. As the sample size increases, the statistic becomes more and more like the Pearson chi-square.

The statistic is computed as

$$Q_C = \sum_i \sum_j \frac{[\,\max(0, |n_{ij} - e_{ij}| - 0.5)\,]^2}{e_{ij}}$$

Under the null hypothesis of independence, Q_C has an asymptotic chi-square distribution with $(R-1)(C-1)$ degrees of freedom.

Mantel-Haenszel Chi-Square Test

The Mantel-Haenszel chi-square statistic tests the alternative hypothesis that there is a linear association between the row variable and the column variable. Both variables must lie on an ordinal scale. The statistic is computed as

$$Q_{MH} = (n-1)r^2$$

where r^2 is the Pearson correlation between the row variable and the column variable. For a description of the Pearson correlation, see the "Pearson Correlation Coefficient" section on page 113. The Pearson correlation and, thus, the Mantel-Haenszel chi-square statistic use the scores that you specify in the SCORES= option in the TABLES statement.

Under the null hypothesis of no association, Q_{MH} has an asymptotic chi-square distribution with one degree of freedom. In addition to the asymptotic test, PROC FREQ computes the exact test when you specify the MHCHI or CHISQ option in the EXACT statement.

Refer to Mantel and Haenszel (1959) and Landis, Heyman, and Koch (1978).

Fisher's Exact Test

Fisher's exact test is another test of association between the row and column variables. This test assumes that the row and column totals are fixed, and then uses the hypergeometric distribution to compute probabilities of possible tables with these observed row and column totals. Fisher's exact test does not depend on any large-sample distribution assumptions, and so it is appropriate even for small sample sizes and for sparse tables.

2 × 2 Tables

For 2×2 tables, PROC FREQ gives the following information for Fisher's exact test: table probability, two-sided p-value, left-sided p-value, and right-sided p-value. The table probability equals the hypergeometric probability of the observed table, and is in fact the value of the test statistic for Fisher's exact test.

Where p is the hypergeometric probability of a specific table with the observed row and column totals, Fisher's exact p-values are computed by summing probabilities p over defined sets of tables,

$$\mathit{PROB} = \sum_A p$$

The two-sided p-value is the sum of all possible table probabilties (for tables having the observed row and column totals) that are less than or equal to the observed table probability. So, for the two-sided p-value, the set A includes all possible tables with hypergeometric probabilities less than or equal to the probability of the observed table. A small two-sided p-value supports the alternative hypothesis of association between the row and column variables.

One-sided tests are defined in terms of the frequency of the cell in the first row and first column of the table, the (1,1) cell. Denoting the observed (1,1) cell frequency by F, the left-sided p-value for Fisher's exact test is probability that the (1,1) cell frequency is less than or equal to F. So, for the left-sided p-value, the set A includes those tables with a (1,1) cell frequency less than or equal to F. A small left-sided p-value supports the alternative hypothesis that the probability of an observation being in the first cell is less than expected under the null hypothesis of independent row and column variables.

Similarly, for a right-sided alternative hypothesis, A is the set of tables where the frequency of the (1,1) cell is greater than or equal to that in the observed table. A small right-sided p-value supports the alternative that the probability of the first cell is greater than that expected under the null hypothesis.

Because the (1,1) cell frequency completely determines the 2×2 table when the marginal row and column sums are fixed, these one-sided alternatives can be equivalently stated in terms of other cell probabilities or ratios of cell probabilities. The left-sided alternative is equivalent to an odds ratio greater than 1, where the odds ratio equals ($n_{11}\, n_{22} \;/\; n_{12}\, n_{21}$). Additionally, the left-sided alternative is equivalent to the column 1 risk for row 1 being less than the column 1 risk for row 2, $p_{1|1} < p_{1|2}$. Similarly, the right-sided alternative is equivalent to the column 1 risk for row 1 being greater than the column 1 risk for row 2, $p_{1|1} > p_{1|2}$. Refer to Agresti (1996).

R × *C* Tables

Fisher's exact test was extended to general $R \times C$ tables by Freeman and Halton (1951), and this test is also known as the Freeman-Halton test. For $R \times C$ tables, the two-sided p-value is defined the same as it is for 2×2 tables. The set A contains all tables with p less than or equal to the probability of the observed table. A small p-value supports the alternative hypothesis of association between the row and column variables. For $R \times C$ tables, Fisher's exact test is inherently two-sided. The alternative hypothesis is defined only in terms of general, and not linear, association. Therefore, PROC FREQ does not provide right-sided or left-sided p-values for general $R \times C$ tables.

For $R \times C$ tables, PROC FREQ computes Fisher's exact test using the network algorithm of Mehta and Patel (1983), which provides a faster and more efficient solution than direct enumeration. See the section "Exact Statistics" beginning on page 142 for more details.

Phi Coefficient

The phi coefficient is a measure of association derived from the Pearson chi-square statistic. It has the range $-1 \leq \phi \leq 1$ for 2×2 tables. Otherwise, the range is $0 \leq \phi \leq min(\sqrt{R-1}, \sqrt{C-1})$ (Liebetrau 1983). The phi coefficient is computed as

$$\phi = \frac{n_{11}\, n_{22} - n_{12}\, n_{21}}{\sqrt{n_{1\cdot}\, n_{2\cdot}\, n_{\cdot 1}\, n_{\cdot 2}}} \qquad \text{for } 2 \times 2 \text{ tables}$$

$$\phi = \sqrt{Q_P / n} \qquad \text{otherwise}$$

Refer to Fleiss (1981, pp. 59–60).

Contingency Coefficient

The contingency coefficient is a measure of association derived from the Pearson chi-square. It has the range $0 \leq P \leq \sqrt{(m-1)/m}$, where $m = min(R, C)$ (Liebetrau 1983). The contingency coefficient is computed as

$$P = \sqrt{\frac{Q_P}{Q_P + n}}$$

Refer to Kendall and Stuart (1979, pp. 587–588).

Cramer's *V*

Cramer's V is a measure of association derived from the Pearson chi-square. It is designed so that the attainable upper bound is always 1. It has the range $-1 \leq V \leq 1$ for 2×2 tables; otherwise, the range is $0 \leq V \leq 1$. Cramer's V is computed as

$$V = \phi \qquad \text{for } 2 \times 2 \text{ tables}$$

$$V = \sqrt{\frac{Q_P / n}{\min(R-1, C-1)}} \qquad \text{otherwise}$$

Refer to Kendall and Stuart (1979, p. 588).

Measures of Association

When you specify the MEASURES option in the TABLES statement, PROC FREQ computes several statistics that describe the association between the two variables of the contingency table. The following are measures of ordinal association that consider whether the variable Y tends to increase as X increases: gamma, Kendall's tau-b, Stuart's tau-c, and Somers' D. These measures are appropriate for ordinal variables, and they classify pairs of observations as *concordant* or *discordant*. A pair is concordant if the observation with the larger value of X also has the larger value of Y. A pair is discordant if the observation with the larger value of X has the smaller value of Y. Refer to Agresti (1996) and the other references cited in the discussion of each measure of association.

The Pearson correlation coefficient and the Spearman rank correlation coefficient are also appropriate for ordinal variables. The Pearson correlation describes the strength of the linear association between the row and column variables, and it is computed using the row and column scores specified by the SCORES= option in the TABLES statement. The Spearman correlation is computed with rank scores. The polychoric correlation (requested by the PLCORR option) also requires ordinal variables and assumes that the variables have an underlying bivariate normal distribution. The following measures of association do not require ordinal variables, but they are appropriate for nominal variables: lambda asymmetric, lambda symmetric, and uncertainty coefficients.

PROC FREQ computes estimates of the measures according to the formulas given in the discussion of each measure of association. For each measure, PROC FREQ computes an asymptotic standard error (ASE), which is the square root of the asymptotic variance denoted by var in the following sections.

Confidence Limits

If you specify the CL option in the TABLES statement, PROC FREQ computes asymptotic confidence limits for all MEASURES statistics. The confidence coefficient is determined according to the value of the ALPHA= option, which, by default, equals 0.05 and produces 95% confidence limits.

The confidence limits are computed as

$$est \;\pm\; (\, z_{\alpha/2} \;\times\; \mathrm{ASE}\,)$$

where est is the estimate of the measure, $z_{\alpha/2}$ is the $100(1-\alpha/2)$ percentile of the standard normal distribution, and ASE is the asymptotic standard error of the estimate.

Asymptotic Tests

For each measure that you specify in the TEST statement, PROC FREQ computes an asymptotic test of the null hypothesis that the measure equals zero. Asymptotic tests are available for the following measures of association: gamma, Kendall's tau-b, Stuart's tau-c, Somers' $D(R|C)$, Somers' $D(C|R)$, the Pearson correlation coefficient, and the Spearman rank correlation coefficient. To compute an asymptotic test, PROC FREQ uses a standardized test statistic z, which has an asymptotic standard normal distribution under the null hypothesis. The standardized test statistic is computed as

$$z \;=\; \frac{est}{\sqrt{var_0(est)}}$$

where est is the estimate of the measure and $var_0(est)$ is the variance of the estimate under the null hypothesis. Formulas for $var_0(est)$ are given in the discussion of each measure of association.

Note that the ratio of est to $\sqrt{var_0(est)}$ is the same for the following measures: gamma, Kendall's tau-b, Stuart's tau-c, Somers' $D(R|C)$, and Somers' $D(C|R)$. Therefore, the tests for these measures are identical. For example, the p-values for the test of H_0: gamma $= 0$ equal the p-values for the test of H_0: tau-$b = 0$.

PROC FREQ computes one-sided and two-sided p-values for each of these tests. When the test statistic z is greater than its null hypothesis expected value of zero, PROC FREQ computes the right-sided p-value, which is the probability of a larger value of the statistic occurring under the null hypothesis. A small right-sided p-value supports the alternative hypothesis that the true value of the measure is greater than zero. When the test statistic is less than or equal to zero, PROC FREQ computes the left-sided p-value, which is the probability of a smaller value of the statistic occurring under the null hypothesis. A small left-sided p-value supports the alternative hypothesis that the true value of the measure is less than zero. The one-sided p-value P_1 can be expressed as

$$P_1 = \text{Prob}\ (\ Z > z\ ) \qquad \text{if} \quad z > 0$$

$$P_1 = \text{Prob}\ (\ Z < z\ ) \qquad \text{if} \quad z \leq 0$$

where Z has a standard normal distribution. The two-sided p-value P_2 is computed as

$$P_2 = \text{Prob}\ (\ |Z| > |z|\ )$$

Exact Tests

Exact tests are available for two measures of association, the Pearson correlation coefficient and the Spearman rank correlation coefficient. If you specify the PCORR option in the EXACT statement, PROC FREQ computes the exact test of the hypothesis that the Pearson correlation equals zero. If you specify the SCORR option in the EXACT statement, PROC FREQ computes the exact test of the hypothesis that the Spearman correlation equals zero. See the section "Exact Statistics" beginning on page 142 for information on exact tests.

Gamma

The estimator of gamma is based only on the number of concordant and discordant pairs of observations. It ignores tied pairs (that is, pairs of observations that have equal values of X or equal values of Y). Gamma is appropriate only when both variables lie on an ordinal scale. It has the range $-1 \leq \Gamma \leq 1$. If the two variables are independent, then the estimator of gamma tends to be close to zero. Gamma is estimated by

$$G = \frac{P - Q}{P + Q}$$

with asymptotic variance

$$var = \frac{16}{(P + Q)^4} \sum_i \sum_j n_{ij} (QA_{ij} - PD_{ij})^2$$

The variance of the estimator under the null hypothesis that gamma equals zero is computed as

$$var_0(G) = \frac{4}{(P+Q)^2}\left(\sum_i \sum_j n_{ij}(A_{ij} - D_{ij})^2 - (P-Q)^2/n\right)$$

For 2×2 tables, gamma is equivalent to Yule's Q. Refer to Goodman and Kruskal (1979), Agresti (1990), and Brown and Benedetti (1977).

Kendall's Tau-*b*

Kendall's tau-b is similar to gamma except that tau-b uses a correction for ties. Tau-b is appropriate only when both variables lie on an ordinal scale. Tau-b has the range $-1 \le \tau_b \le 1$. It is estimated by

$$t_b = \frac{P-Q}{\sqrt{w_r w_c}}$$

with

$$var = \frac{1}{w^4}\left(\sum_i \sum_j n_{ij}(2wd_{ij} + t_b v_{ij})^2 - n^3 t_b^2 (w_r + w_c)^2\right)$$

where

$$\begin{aligned}
w &= \sqrt{w_r w_c} \\
w_r &= n^2 - \sum_i n_{i\cdot}^2 \\
w_c &= n^2 - \sum_j n_{\cdot j}^2 \\
d_{ij} &= A_{ij} - D_{ij} \\
v_{ij} &= n_{i\cdot} w_c + n_{\cdot j} w_r
\end{aligned}$$

The variance of the estimator under the null hypothesis that tau-b equals zero is computed as

$$var_0(t_b) = \frac{4}{w_r w_c}\left(\sum_i \sum_j n_{ij}(A_{ij} - D_{ij})^2 - (P-Q)^2/n\right)$$

Refer to Kendall (1955) and Brown and Benedetti (1977).

Stuart's Tau-*c*

Stuart's tau-c makes an adjustment for table size in addition to a correction for ties. Tau-c is appropriate only when both variables lie on an ordinal scale. Tau-c has the range $-1 \leq \tau_c \leq 1$. It is estimated by

$$t_c = \frac{m(P-Q)}{n^2(m-1)}$$

with

$$var = \frac{4m^2}{(m-1)^2 n^4} \left(\sum_i \sum_j n_{ij} d_{ij}^2 - (P-Q)^2/n \right)$$

where

$$\begin{aligned} m &= \min(R, C) \\ d_{ij} &= A_{ij} - D_{ij} \end{aligned}$$

The variance of the estimator under the null hypothesis that tau-c equals zero is

$$var_0(t_c) = var$$

Refer to Brown and Benedetti (1977).

Somers' *D* (*C* |*R*) and *D* (*R* |*C*)

Somers' $D(C|R)$ and Somers' $D(R|C)$ are asymmetric modifications of tau-b. $C|R$ denotes that the row variable X is regarded as an independent variable, while the column variable Y is regarded as dependent. Similarly, $R|C$ denotes that the column variable Y is regarded as an independent variable, while the row variable X is regarded as dependent. Somers' D differs from tau-b in that it uses a correction only for pairs that are tied on the independent variable. Somers' D is appropriate only when both variables lie on an ordinal scale. It has the range $-1 \leq D \leq 1$. Formulas for Somers' $D(R|C)$ are obtained by interchanging the indices.

$$D(C|R) = \frac{P-Q}{w_r}$$

with

$$var = \frac{4}{w_r^4} \sum_i \sum_j n_{ij} \left(w_r d_{ij} - (P-Q)(n - n_{i\cdot}) \right)^2$$

where

$$
\begin{aligned}
w_r &= n^2 - \sum_i n_{i\cdot}^2 \\
d_{ij} &= A_{ij} - D_{ij}
\end{aligned}
$$

The variance of the estimator under the null hypothesis that $D(C|R)$ equals zero is computed as

$$var_0(D(C|R)) = \frac{4}{w_r^2} \left(\sum_i \sum_j n_{ij}(A_{ij} - D_{ij})^2 - (P-Q)^2/n \right)$$

Refer to Somers (1962), Goodman and Kruskal (1979), and Liebetrau (1983).

Pearson Correlation Coefficient

PROC FREQ computes the Pearson correlation coefficient using the scores specified in the SCORES= option. The Pearson correlation is appropriate only when both variables lie on an ordinal scale. It has the range $-1 \leq \rho \leq 1$. The Pearson correlation coefficient is computed as

$$r = \frac{v}{w} = \frac{ss_{rc}}{\sqrt{ss_r ss_c}}$$

with

$$var = \frac{1}{w^4} \sum_i \sum_j n_{ij} \left(w(R_i - \bar{R})(C_j - \bar{C}) - \frac{b_{ij} v}{2w} \right)^2$$

The row scores R_i and the column scores C_j are determined by the SCORES= option in the TABLES statement, and

$$
\begin{aligned}
ss_r &= \sum_i \sum_j n_{ij}(R_i - \bar{R})^2 \\
ss_c &= \sum_i \sum_j n_{ij}(C_j - \bar{C})^2 \\
ss_{rc} &= \sum_i \sum_j n_{ij}(R_i - \bar{R})(C_j - \bar{C}) \\
b_{ij} &= (R_i - \bar{R})^2 ss_c + (C_j - \bar{C})^2 ss_r \\
v &= ss_{rc} \\
w &= \sqrt{ss_r ss_c}
\end{aligned}
$$

Refer to Snedecor and Cochran (1989) and Brown and Benedetti (1977).

To compute an asymptotic test for the Pearson correlation, PROC FREQ uses a standardized test statistic r^*, which has an asymptotic standard normal distribution under the null hypothesis that the correlation equals zero. The standardized test statistic is computed as

$$r^* = \frac{r}{\sqrt{var_0(r)}}$$

where $var_0(r)$ is the variance of the correlation under the null hypothesis.

$$var_0(r) = \frac{\sum_i \sum_j n_{ij}(R_i - \bar{R})^2 (C_j - \bar{C})^2 - ss_{rc}^2/n}{ss_r ss_c}$$

The asymptotic variance is derived for multinomial sampling in a contingency table framework, and it differs from the form obtained under the assumption that both variables are continuous and normally distributed. Refer to Brown and Benedetti (1977).

PROC FREQ also computes the exact test for the hypothesis that the Pearson correlation equals zero when you specify the PCORR option in the EXACT statement. See the section "Exact Statistics" beginning on page 142 for information on exact tests.

Spearman Rank Correlation Coefficient

The Spearman correlation coefficient is computed using rank scores $R1_i$ and $C1_j$, defined in the section "Scores" beginning on page 102. It is appropriate only when both variables lie on an ordinal scale. It has the range $-1 \leq \rho_s \leq 1$. The Spearman correlation coefficient is computed as

$$r_s = \frac{v}{w}$$

with

$$var = \frac{1}{n^2 w^4} \sum_i \sum_j n_{ij} (z_{ij} - \bar{z})^2$$

where

$$\begin{aligned} v &= \sum_i \sum_j n_{ij} R(i) C(j) \\ w &= \frac{1}{12} \sqrt{FG} \\ F &= n^3 - \sum_i n_{i\cdot}^3 \end{aligned}$$

$$
\begin{aligned}
G &= n^3 - \sum_j n_{\cdot j}^3 \\
R(i) &= R1_i - n/2 \\
C(j) &= C1_j - n/2 \\
\bar{z} &= \frac{1}{n} \sum_i \sum_j n_{ij} z_{ij} \\
z_{ij} &= w v_{ij} - v w_{ij} \\
v_{ij} &= n \Biggl(R(i)C(j) + \frac{1}{2} \sum_l n_{il} C(l) + \frac{1}{2} \sum_k n_{kj} R(k) + \\
&\qquad \sum_l \sum_{k>i} n_{kl} C(l) + \sum_k \sum_{l>j} n_{kl} R(k) \Biggr) \\
w_{ij} &= \frac{-n}{96w} \left(F n_{\cdot j}^2 + G n_{i\cdot}^2 \right)
\end{aligned}
$$

Refer to Snedecor and Cochran (1989) and Brown and Benedetti (1977).

To compute an asymptotic test for the Spearman correlation, PROC FREQ uses a standardized test statistic r_s^*, which has an asymptotic standard normal distribution under the null hypothesis that the correlation equals zero. The standardized test statistic is computed as

$$
r_s^* = \frac{r_s}{\sqrt{var_0(r_s)}}
$$

where $var_0(r_s)$ is the variance of the correlation under the null hypothesis.

$$
var_0(r_s) = \frac{1}{n^2 w^2} \sum_i \sum_j n_{ij} (v_{ij} - \bar{v})^2
$$

where

$$
\bar{v} = \sum_i \sum_j n_{ij} v_{ij} / n
$$

The asymptotic variance is derived for multinomial sampling in a contingency table framework, and it differs from the form obtained under the assumption that both variables are continuous and normally distributed. Refer to Brown and Benedetti (1977).

PROC FREQ also computes the exact test for the hypothesis that the Spearman rank correlation equals zero when you specify the SCORR option in the EXACT statement. See the section "Exact Statistics" beginning on page 142 for information on exact tests.

Polychoric Correlation

When you specify the PLCORR option in the TABLES statement, PROC FREQ computes the polychoric correlation. This measure of association is based on the assumption that the ordered, categorical variables of the frequency table have an underlying bivariate normal distribution. For 2×2 tables, the polychoric correlation is also known as the tetrachoric correlation. Refer to Drasgow (1986) for an overview of polychoric correlation. The polychoric correlation coefficient is the maximum likelihood estimate of the product-moment correlation between the normal variables, estimating thresholds from the observed table frequencies. The range of the polychoric correlation is from -1 to 1. Olsson (1979) gives the likelihood equations and an asymptotic covariance matrix for the estimates.

To estimate the polychoric correlation, PROC FREQ iteratively solves the likelihood equations by a Newton-Raphson algorithm using the Pearson correlation coefficient as the initial approximation. Iteration stops when the convergence measure falls below the convergence criterion or when the maximum number of iterations is reached, whichever occurs first. The CONVERGE= option sets the convergence criterion, and the default value is 0.0001. The MAXITER= option sets the maximum number of iterations, and the default value is 20.

Lambda Asymmetric

Asymmetric lambda, $\lambda(C|R)$, is interpreted as the probable improvement in predicting the column variable Y given knowledge of the row variable X. Asymmetric lambda has the range $0 \leq \lambda(C|R) \leq 1$. It is computed as

$$\lambda(C|R) = \frac{\sum_i r_i - r}{n - r}$$

with

$$var = \frac{n - \sum_i r_i}{(n-r)^3} \left(\sum_i r_i + r - 2 \sum_i (r_i \mid l_i = l) \right)$$

where

$$\begin{aligned} r_i &= \max_j (n_{ij}) \\ r &= \max_j (n_{\cdot j}) \end{aligned}$$

Also, let l_i be the unique value of j such that $r_i = n_{ij}$, and let l be the unique value of j such that $r = n_{\cdot j}$.

Because of the uniqueness assumptions, ties in the frequencies or in the marginal totals must be broken in an arbitrary but consistent manner. In case of ties, l is defined here as the smallest value of j such that $r = n_{\cdot j}$. For a given i, if there is at least one value j such that $n_{ij} = r_i = c_j$, then l_i is defined here to be the smallest such value of j. Otherwise, if $n_{il} = r_i$, then l_i is defined to be equal to l. If neither condition is true, then l_i is taken to be the smallest value of j such that $n_{ij} = r_i$. The formulas for lambda asymmetric $(R|C)$ can be obtained by interchanging the indices.

Refer to Goodman and Kruskal (1979).

Lambda Symmetric

The nondirectional lambda is the average of the two asymmetric lambdas, $\lambda(C|R)$ and $\lambda(R|C)$. Lambda symmetric has the range $0 \le \lambda \le 1$. Lambda symmetric is defined as

$$\lambda = \frac{\sum_i r_i + \sum_j c_j - r - c}{2n - r - c} = \frac{w - v}{w}$$

with

$$var = \frac{1}{w^4}\left(wvy - 2w^2\left[n - \sum_i \sum_j (n_{ij} \mid j = l_i, i = k_j)\right] - 2v^2(n - n_{kl})\right)$$

where

$$\begin{aligned}
c_j &= \max_i(n_{ij}) \\
c &= \max_i(n_{i\cdot}) \\
w &= 2n - r - c \\
v &= 2n - \sum_i r_i - \sum_j c_j \\
x &= \sum_i (r_i \mid l_i = l) \ + \ \sum_j (c_j \mid k_j = k) \ + \ r_k \ + \ c_l \\
y &= 8n - w - v - 2x
\end{aligned}$$

Refer to Goodman and Kruskal (1979).

Uncertainty Coefficients (*C* |*R*) and (*R* |*C*)

The uncertainty coefficient, $U(C|R)$, is the proportion of uncertainty (entropy) in the column variable Y that is explained by the row variable X. It has the range $0 \le U(C|R) \le 1$. The formulas for $U(R|C)$ can be obtained by interchanging the indices.

$$U(C|R) = \frac{H(X) + H(Y) - H(XY)}{H(Y)} = \frac{v}{w}$$

with

$$var = \frac{1}{n^2 w^4} \sum_i \sum_j n_{ij} \left(H(Y) \ln \left(\frac{n_{ij}}{n_{i\cdot}} \right) + (H(X) - H(XY)) \ln \left(\frac{n_{\cdot j}}{n} \right) \right)^2$$

where

$$\begin{aligned}
v &= H(X) + H(Y) - H(XY) \\
w &= H(Y) \\
H(X) &= -\sum_i \left(\frac{n_{i\cdot}}{n} \right) \ln \left(\frac{n_{i\cdot}}{n} \right) \\
H(Y) &= -\sum_j \left(\frac{n_{\cdot j}}{n} \right) \ln \left(\frac{n_{\cdot j}}{n} \right) \\
H(XY) &= -\sum_i \sum_j \left(\frac{n_{ij}}{n} \right) \ln \left(\frac{n_{ij}}{n} \right)
\end{aligned}$$

Refer to Theil (1972, pp. 115–120) and Goodman and Kruskal (1979).

Uncertainty Coefficient (U)

The uncertainty coefficient, U, is the symmetric version of the two asymmetric coefficients. It has the range $0 \leq U \leq 1$. It is defined as

$$U = \frac{2(H(X) + H(Y) - H(XY))}{H(X) + H(Y)}$$

with

$$var = 4 \sum_i \sum_j \frac{n_{ij} \left(H(XY) \ln \left(\frac{n_{i\cdot} n_{\cdot j}}{n^2} \right) - (H(X) + H(Y)) \ln \left(\frac{n_{ij}}{n} \right) \right)^2}{n^2 \left(H(X) + H(Y) \right)^4}$$

Refer to Goodman and Kruskal (1979).

Binomial Proportion

When you specify the BINOMIAL option in the TABLES statement, PROC FREQ computes a binomial proportion for one-way tables. By default this is the proportion of observations in the first variable level, or class, that appears in the output. To specify a different level, use the LEVEL= option.

$$\hat{p} = n_1 \,/\, n$$

where n_1 is the frequency for the first level and n is the total frequency for the one-way table. The standard error for the binomial proportion is computed as

$$se(\hat{p}) \quad = \quad \sqrt{\hat{p}\,(1-\hat{p})\;/\;n}$$

Using the normal approximation to the binomial distribution, PROC FREQ constructs asymptotic confidence limits for p according to

$$\hat{p} \quad \pm \quad (\; z_{\alpha/2} \;\times\; se(\hat{p}) \;)$$

where $z_{\alpha/2}$ is the $100(1-\alpha/2)$ percentile of the standard normal distribution. The confidence level α is determined by the ALPHA= option, which, by default, equals 0.05 and produces 95% confidence limits.

If you specify the BINOMIALC option, PROC FREQ includes a continuity correction of $1/2n$ in the asymptotic confidence limits for p. The purpose of this correction is to adjust for the difference between the normal approximation and the binomial distribution, which is a discrete distribution. Refer to Fleiss (1981). With the continuity correction, the asymptotic confidence limits for p are

$$\hat{p} \quad \pm \quad (\; z_{\alpha/2} \;\times\; se(\hat{p}) \quad + \quad (1/2n) \;)$$

Additionally, PROC FREQ computes exact confidence limits for the binomial proportion using the *F* distribution method given in Collett (1991) and also described by Leemis and Trivedi (1996).

PROC FREQ computes an asymptotic test of the hypothesis that the binomial proportion equals p_0, where the value of p_0 is specified by the P= option in the TABLES statement. If you do not specify a value for the P= option, PROC FREQ uses $p_0 = 0.5$ by default. The asymptotic test statistic is

$$z \quad = \quad \frac{\hat{p} - p_0}{\sqrt{p_0\;(1-p_0)\;/\;n}}$$

If you specify the BINOMIALC option, PROC FREQ includes a continuity correction in the asymptotic test statistic, towards adjusting for the difference between the normal approximation and the discrete binomial distribution. Refer to Fleiss (1981). The continuity correction of $(1/2n)$ is subtracted from $(\hat{p} - p_0)$ in the numerator of the test statistic z if $(\hat{p} - p_0)$ is positive; otherwise, the continuity correction is added to the numerator.

PROC FREQ computes one-sided and two-sided p-values for this test. When the test statistic z is greater than zero, its expected value under the null hypothesis, PROC FREQ computes the right-sided p-value, which is the probability of a larger value of the statistic occurring under the null hypothesis. A small right-sided p-value supports the alternative hypothesis that the true value of the proportion is greater than p_0. When the test statistic is less than or equal to zero, PROC FREQ computes the left-sided p-value, which is the probability of a smaller value of the statistic occurring under the null hypothesis. A small left-sided p-value supports the alternative hypothesis that the true value of the proportion is less than p_0. The one-sided p-value P_1 can be expressed as

$$P_1 = \text{Prob}\ (\ Z > z\ ) \qquad \text{if} \quad z > 0$$

$$P_1 = \text{Prob}\ (\ Z < z\ ) \qquad \text{if} \quad z \le 0$$

where Z has a standard normal distribution. The two-sided *p*-value P_2 is computed as

$$P_2 = \text{Prob}\ (\ |Z| > |z|\ )$$

When you specify the BINOMIAL option in the EXACT statement, PROC FREQ also computes an exact test of the null hypothesis $H_0: p = p_0$. To compute this exact test, PROC FREQ uses the binomial probability function

$$\text{Prob}\ (X = x \mid p_0) = \binom{n}{x} p_0^{\,x}\ (1 - p_0)^{\,(n-x)} \qquad x = 0, 1, 2, \ldots, n$$

where the variable X has a binomial distribution with parameters n and p_0. To compute $\text{Prob}(X \le n_1)$, PROC FREQ sums these binomial probabilities over x from zero to n_1. To compute $\text{Prob}(X \ge n_1)$, PROC FREQ sums these binomial probabilities over x from n_1 to n. Then the exact one-sided *p*-value is

$$P_1 = \min\ (\ \text{Prob}(X \le n_1 \mid p_0),\ \text{Prob}(X \ge n_1 \mid p_0)\ )$$

and the exact two-sided *p*-value is

$$P_2 = 2 \times P_1$$

Risks and Risk Differences

The RISKDIFF option in the TABLES statement provides estimates of risks (or binomial proportions) and risk differences for 2×2 tables. This analysis may be appropriate when comparing the proportion of some characteristic for two groups, where row 1 and row 2 correspond to the two groups, and the columns correspond to two possible characteristics or outcomes. For example, the row variable might be a treatment or dose, and the column variable might be the response. Refer to Collett (1991), Fleiss (1981), and Stokes, Davis, and Koch (1995).

Let the frequencies of the 2×2 table be represented as follows.

	Column 1	Column 2	Total
Row 1	n_{11}	n_{12}	$n_{1\cdot}$
Row 2	n_{21}	n_{22}	$n_{2\cdot}$
Total	$n_{\cdot 1}$	$n_{\cdot 2}$	n

The column 1 risk for row 1 is the proportion of row 1 observations classified in column 1,

$$p_{1|1} \;=\; n_{11} \;/\; n_{1\cdot}$$

This estimates the conditional probability of the column 1 response, given the first level of the row variable.

The column 1 risk for row 2 is the proportion of row 2 observations classified in column 1,

$$p_{1|2} \;=\; n_{21} \;/\; n_{2\cdot}$$

and the overall column 1 risk is the proportion of all observations classified in column 1,

$$p_{\cdot 1} \;=\; n_{\cdot 1} \;/\; n$$

The column 1 risk difference compares the risks for the two rows, and it is computed as the column 1 risk for row 1 minus the column 1 risk for row 2,

$$(\mathit{pdiff})_1 \;=\; p_{1|1} - p_{1|2}$$

The risks and risk difference are defined similarly for column 2.

The standard error of the column 1 risk estimate for row i is computed as

$$se(p_{1|i}) \;=\; \sqrt{p_{1|i}\;(1 - p_{1|i})\;/\;n_{i\cdot}}$$

The standard error of the overall column 1 risk estimate is computed as

$$se(p_{\cdot 1}) \;=\; \sqrt{p_{\cdot 1}\;(1 - p_{\cdot 1})\;/\;n}$$

If the two rows represent independent binomial samples, the standard error for the column 1 risk difference is computed as

$$se\,(\,(\mathit{pdiff})_1\,) \;=\; \sqrt{var(p_{1|1}) + var(p_{1|2})}$$

The standard errors are computed in a similar manner for the column 2 risks and risk difference.

Using the normal approximation to the binomial distribution, PROC FREQ constructs asymptotic confidence limits for the risks and risk differences according to

$$est \;\pm\; (\;z_{\alpha/2} \;\times\; se(est)\;)$$

where est is the estimate, $z_{\alpha/2}$ is the $100(1-\alpha/2)$ percentile of the standard normal distribution, and $se(est)$ is the standard error of the estimate. The confidence level α is determined from the value of the ALPHA= option, which, by default, equals 0.05 and produces 95% confidence limits.

If you specify the RISKDIFFC option, PROC FREQ includes continuity corrections in the asymptotic confidence limits for the risks and risk differences. Continuity corrections adjust for the difference between the normal approximation and the discrete binomial distribution. Refer to Fleiss (1981). Including a continuity correction, the asymptotic confidence limits become

$$est \quad \pm \quad (\ z_{\alpha/2} \ \times \ se(est) \ + \ cc\ )$$

where cc is the continuity correction. For the column 1 risk for row 1, $cc = (1/2n_{1\cdot})$; for the column 1 risk for row 2, $cc = (1/2n_{2\cdot})$; for the overall column 1 risk, $cc = (1/2n)$; and for the column 1 risk difference, $cc = ((1/n_{1\cdot} + 1/n_{2\cdot})/2)$. Continuity corrections are computed similarly for the column 2 risks and risk difference.

PROC FREQ computes exact confidence limits for the column 1, column 2, and overall risks using the *F* distribution method given in Collett (1991) and also described by Leemis and Trivedi (1996). PROC FREQ does not provide exact confidence limits for the risk differences. Refer to Agresti (1992) for a discussion of issues involved in constructing exact confidence limits for differences of proportions.

Odds Ratio and Relative Risks for 2 x 2 Tables

Odds Ratio (Case-Control Studies)

The odds ratio is a useful measure of association for a variety of study designs. For a retrospective design called a *case-control study*, the odds ratio can be used to estimate the relative risk when the probability of positive response is small (Agresti 1990). In a case-control study, two independent samples are identified based on a binary (yes-no) response variable, and the conditional distribution of a binary explanatory variable is examined, within fixed levels of the response variable. Refer to Stokes, Davis, and Koch (1995) and Agresti (1996).

The odds of a positive response (column 1) in row 1 is n_{11}/n_{12}. Similarly, the odds of a positive response in row 2 is n_{21}/n_{22}. The odds ratio is formed as the ratio of the row 1 odds to the row 2 odds. The odds ratio for 2×2 tables is defined as

$$\text{OR} = \frac{n_{11}/n_{12}}{n_{21}/n_{22}} = \frac{n_{11}\,n_{22}}{n_{12}\,n_{21}}$$

The odds ratio can be any nonnegative number. When the row and column variables are independent, the true value of the odds ratio equals 1. An odds ratio greater than 1 indicates that the odds of a positive response are higher in row 1 than in row 2. Values less than 1 indicate the odds of positive response are higher in row 2. The strength of association increases with the deviation from 1.

The transformation $G = (\text{OR} - 1)/(\text{OR} + 1)$ transforms the odds ratio to the range $(-1, 1)$ with $G = 0$ when $\text{OR} = 1$; $G = -1$ when $\text{OR} = 0$; and G approaches 1 as OR approaches infinity. G is the gamma statistic, which PROC FREQ computes when you specify the MEASURES option.

The asymptotic $100(1-\alpha)\%$ confidence limits for the odds ratio are

$$\left(\text{OR} \cdot \exp(-z\sqrt{v}),\ \text{OR} \cdot \exp(z\sqrt{v}) \right)$$

where

$$v = var(\ln \text{OR}) = \frac{1}{n_{11}} + \frac{1}{n_{12}} + \frac{1}{n_{21}} + \frac{1}{n_{22}}$$

and z is the $100(1-\alpha/2)$ percentile of the standard normal distribution. If any of the four cell frequencies are zero, the estimates are not computed.

When you specify option OR in the EXACT statement, PROC FREQ computes exact confidence limits for the odds ratio. Because this is a discrete problem, the confidence coefficient for these exact confidence limits is not exactly $1-\alpha$ but is at least $1-\alpha$. Thus, these confidence limits are conservative. Refer to Agresti (1992).

PROC FREQ computes exact confidence limits for the odds ratio with an algorithm based on that presented by Thomas (1971). Refer also to Gart (1971). The following two equations are solved iteratively for the lower and upper confidence limits, ϕ_1 and ϕ_2.

$$\begin{aligned}
\sum_{i=n_{11}}^{n_{\cdot 1}} \binom{n_{1\cdot}}{i} \binom{n_{2\cdot}}{n_{\cdot 1}-i} \phi_1^i \ \Big/ \ \sum_{i=0}^{n_{\cdot 1}} \binom{n_{1\cdot}}{i} \binom{n_{2\cdot}}{n_{\cdot 1}-i} \phi_1^i &= \alpha/2 \\
\sum_{i=0}^{n_{11}} \binom{n_{1\cdot}}{i} \binom{n_{2\cdot}}{n_{\cdot 1}-i} \phi_2^i \ \Big/ \ \sum_{i=0}^{n_{\cdot 1}} \binom{n_{1\cdot}}{i} \binom{n_{2\cdot}}{n_{\cdot 1}-i} \phi_2^i &= \alpha/2
\end{aligned}$$

When the odds ratio equals zero, which occurs when either $n_{11} = 0$ or $n_{22} = 0$, then PROC FREQ sets the lower exact confidence limit to zero and determines the upper limit with level α. Similarly, when the odds ratio equals infinity, which occurs when either $n_{12} = 0$ or $n_{21} = 0$, then PROC FREQ sets the upper exact confidence limit to infinity and determines the lower limit with level α.

Relative Risks (Cohort Studies)

These measures of relative risk are useful in *cohort* (prospective) study designs, where two samples are identified based on the presence or absence of an explanatory factor. The two samples are observed in future time for the binary (yes-no) response variable under study. Relative risk measures are also useful in cross-sectional studies, where two variable are observed simultaneously. Refer to Stokes, Davis, and Koch (1995) and Agresti (1996).

The column 1 relative risk is the ratio of the column 1 risks for row 1 to row 2. The column 1 risk for row 1 is the proportion of the row 1 observations classified in column 1,

$$p_{1|1} \quad = \quad n_{11} \,/\, n_{1\cdot}$$

Similarly, the column 1 risk for row 2 is

$$p_{1|2} \quad = \quad n_{21} \;/\; n_{2\cdot}$$

The column 1 relative risk is then computed as

$$\mathrm{RR}_1 = \frac{p_{1|1}}{p_{1|2}}$$

A relative risk greater than 1 indicates that the probability of positive response is greater in row 1 than in row 2. Similarly, a relative risk less than 1 indicates that the probability of positive response is less in row 1 than in row 2. The strength of association increases with the deviation from 1.

The asymptotic $100(1-\alpha)\%$ confidence limits for the column 1 relative risk are

$$\left(\, \mathrm{RR}_1 \cdot \exp(-z\sqrt{v}),\ \mathrm{RR}_1 \cdot \exp(z\sqrt{v}) \,\right)$$

where

$$v = var(\ln \mathrm{RR}_1) = \frac{1 - p_{1|1}}{n_{11}} + \frac{1 - p_{1|2}}{n_{21}}$$

and z is the $100(1-\alpha/2)$ percentile of the standard normal distribution. If either n_{11} or n_{21} is zero, the estimates are not computed.

PROC FREQ computes the column 2 relative risks in a similar manner.

Cochran-Armitage Test for Trend

The TREND option in the TABLES statement requests the Cochran-Armitage test for trend, which tests for trend in binomial proportions across levels of a single factor or covariate. This test is appropriate for a contingency table where one variable has two levels and the other variable is ordinal. The two-level variable represents the response, and the other variable represents an explanatory variable with ordered levels. When the contingency table has two columns and R rows, PROC FREQ tests for trend across the R levels of the row variable, and the binomial proportion is computed as the proportion of observations in the first column. When the table has two rows and C columns, PROC FREQ tests for trend across the C levels of the column variable, and the binomial proportion is computed as the proportion of observations in the first row.

The trend test is based upon the regression coefficient for the weighted linear regression of the binomial proportions on the scores of the levels of the explanatory variable. Refer to Margolin (1988) and Agresti (1990). If the contingency table has two columns and R rows, the trend test statistic is computed as

$$T \quad = \quad \frac{\sum_{i=1}^{R} n_{i1}(R_i - \bar{R})}{\sqrt{p_{\cdot 1}(1 - p_{\cdot 1}) s^2}}$$

where

$$s^2 \;=\; \sum_{i=1}^{R} n_{i\cdot}(R_i - \bar{R})^2$$

The row scores R_i are determined by the value of the SCORES= option in the TABLES statement. By default, PROC FREQ uses table scores. For character variables, the table scores for the row variable are the row numbers (for example, 1 for the first row, 2 for the second row, and so on). For numeric variables, the table score for each row is the numeric value of the row level. When you perform the trend test, the explanatory variable may be numeric (for example, dose of a test substance), and these variable values may be appropriate scores. If the explanatory variable has ordinal levels that are not numeric, you can assign meaningful scores to the variable levels. Sometimes equidistant scores, such as the table scores for a character variable, may be appropriate. For more information on choosing scores for the trend test, refer to Margolin (1988).

The null hypothesis for the Cochran-Armitage test is no trend, which means that the binomial proportion $p_{i1} = n_{i1}/n_{i\cdot}$ is the same for all levels of the explanatory variable. Under this null hypothesis, the trend test statistic is asymptotically distributed as a standard normal random variable. In addition to this asymptotic test, PROC FREQ can compute the exact trend test, which you request by specifying the TREND option in the EXACT statement. See the section "Exact Statistics" beginning on page 142 for information on exact tests.

PROC FREQ computes one-sided and two-sided *p*-values for the trend test. When the test statistic is greater than its null hypothesis expected value of zero, PROC FREQ computes the right-sided *p*-value, which is the probability of a larger value of the statistic occurring under the null hypothesis. A small right-sided *p*-value supports the alternative hypothesis of increasing trend in binomial proportions from row 1 to row R. When the test statistic is less than or equal to zero, PROC FREQ outputs the left-sided *p*-value. A small left-sided *p*-value supports the alternative of decreasing trend.

The one-sided *p*-value P_1 can be expressed as

$$P_1 \;=\; \mathrm{Prob}\ (\ \mathrm{Trend\ Statistic} > T\ ) \qquad \mathrm{if}\ \ T > 0$$

$$P_1 \;=\; \mathrm{Prob}\ (\ \mathrm{Trend\ Statistic} < T\ ) \qquad \mathrm{if}\ \ T \le 0$$

The two-sided *p*-value P_2 is computed as

$$P_2 \;=\; \mathrm{Prob}\ (\ |\mathrm{Trend\ Statistic}| > |T|\ )$$

Jonckheere-Terpstra Test

The JT option in the TABLES statement requests the Jonckheere-Terpstra test, which is a nonparametric test for ordered differences among classes. It tests the null hypothesis that the distribution of the response variable does not differ among classes. It is designed to detect alternatives of ordered class differences, which can be expressed as $\tau_1 \leq \tau_2 \leq \cdots \leq \tau_R$ (or $\tau_1 \geq \tau_2 \geq \cdots \geq \tau_R$), with at least one of the inequalities being strict, where τ_i denotes the effect of class i. For such ordered alternatives, the Jonckheere-Terpstra test can be preferable to tests of more general class difference alternatives, such as the Kruskal–Wallis test (requested by the option WILCOXON in the NPAR1WAY procedure). Refer to Pirie (1983) and Hollander and Wolfe (1973) for more information about the Jonckheere-Terpstra test.

The Jonckheere-Terpstra test is appropriate for a contingency table in which an ordinal column variable represents the response. The row variable, which can be nominal or ordinal, represents the classification variable. The levels of the row variable should be ordered according to the ordering you want the test to detect. The order of variable levels is determined by the ORDER= option in the PROC FREQ statement. The default is ORDER=INTERNAL, which orders by unformatted values. If you specify ORDER=DATA, PROC FREQ orders values according to their order in the input data set. For more information on how to order variable levels, see the ORDER= option on page 76.

The Jonckheere-Terpstra test statistic is computed by first forming $R(R-1)/2$ Mann-Whitney counts $M_{i,i'}$, where $i < i'$, for pairs of rows in the contingency table,

$$
\begin{aligned}
M_{i,i'} \quad = \quad & \{ \quad \text{number of times } X_{i,j} < X_{i',j'}, \\
& \quad\;\; j = 1, \ldots, n_{i.}; \quad j' = 1, \ldots, n_{i'.} \quad \} \\
+ \tfrac{1}{2} \; & \{ \quad \text{number of times } X_{i,j} = X_{i',j'}, \\
& \quad\;\; j = 1, \ldots, n_{i.}; \quad j' = 1, \ldots, n_{i'.} \quad \}
\end{aligned}
$$

where $X_{i,j}$ is response j in row i. Then the Jonckheere-Terpstra test statistic is computed as

$$
J \quad = \sum_{1 \leq i <} \sum_{i' \leq R} M_{i,i'}
$$

This test rejects the null hypothesis of no difference among classes for large values of J. Asymptotic *p*-values for the Jonckheere-Terpstra test are obtained by using the normal approximation for the distribution of the standardized test statistic. The standardized test statistic is computed as

$$
J^* \quad = \quad \frac{J - E_0(J)}{\sqrt{var_0(J)}}
$$

where $E_0(J)$ and $var_0(J)$ are the expected value and variance of the test statistic under the null hypothesis.

$$
E_0(J) \quad = \quad \left(n^2 - \sum_i n_{i.}^2 \right) / 4
$$

$$
var_0(J) \quad = \quad A/72 + B/\left[36n(n-1)(n-2)\right] + C/\left[8n(n-1)\right]
$$

where

$$A = n(n-1)(2n+5) - \sum_i n_{i\cdot}(n_{i\cdot}-1)(2n_{i\cdot}+5) - \sum_j n_{\cdot j}(n_{\cdot j}-1)(2n_{\cdot j}+5)$$

$$B = \left[\sum_i n_{i\cdot}(n_{i\cdot}-1)(n_{i\cdot}-2) \right] \left[\sum_j n_{\cdot j}(n_{\cdot j}-1)(n_{\cdot j}-2) \right]$$

$$C = \left[\sum_i n_{i\cdot}(n_{i\cdot}-1) \right] \left[\sum_j n_{\cdot j}(n_{\cdot j}-1) \right]$$

In addition to this asymptotic test, PROC FREQ can compute the exact Jonckheere-Terpstra test, which you request by specifying the JT option in the EXACT statement. See the section "Exact Statistics" beginning on page 142 for information on exact tests.

PROC FREQ computes one-sided and two-sided *p*-values for the Jonckheere-Terpstra test. When the standardized test statistic is greater than its null hypothesis expected value of zero, PROC FREQ computes the right-sided *p*-value, which is the probability of a larger value of the statistic occurring under the null hypothesis. A small right-sided *p*-value supports the alternative hypothesis of increasing order from row 1 to row R. When the standardized test statistic is less than or equal to zero, PROC FREQ computes the left-sided *p*-value. A small left-sided *p*-value supports the alternative of decreasing order from row 1 to row R.

The one-sided *p*-value P_1 can be expressed as

$$\begin{aligned} P_1 &= \text{Prob (Std JT Statistic} > J^* \text{)} \qquad \text{if } \; J^* > 0 \\ P_1 &= \text{Prob (Std JT Statistic} < J^* \text{)} \qquad \text{if } \; J^* \le 0 \end{aligned}$$

The two-sided *p*-value P_2 is computed as

$$P_2 = \text{Prob (} |\text{Std JT Statistic}| > |J^*| \text{)}$$

Tests and Measures of Agreement

When you specify the AGREE option in the TABLES statement, PROC FREQ computes tests and measures of agreement for square tables (that is, for tables where the number of rows equals the number of columns). For two-way tables, these tests and measures include McNemar's test for 2×2 tables, Bowker's test of symmetry, the simple kappa coefficient, and the weighted kappa coefficient. For multiple strata (n-way tables, where $n > 2$), PROC FREQ computes the overall simple kappa coefficient and the overall weighted kappa coefficient, as well as tests for equal kappas (simple and weighted) among strata. Cochran's Q is computed for multi-way tables when each variable has two levels, that is, for $2 \times 2 \times \cdots \times 2$ tables.

PROC FREQ computes the kappa coefficients (simple and weighted), their asymptotic standard errors, and their confidence limits when you specify the AGREE option in the TABLES statement. If you also specify the KAPPA option in the TEST statement, then PROC FREQ computes the asymptotic test of the hypothesis that simple kappa equals zero. Similarly, if you specify the WTKAP option in the TEST statement, PROC FREQ computes the asymptotic test for weighted kappa.

In addition to the asymptotic tests described in this section, PROC FREQ computes the exact p-value for McNemar's test when you specify the option MCNEM in the EXACT statement. For the kappa statistics, PROC FREQ computes the exact test of the hypothesis that kappa (or weighted kappa) equals zero when you specify the option KAPPA (or WTKAP) in the EXACT statement. See the section "Exact Statistics" beginning on page 142 for information on exact tests.

The discussion of each test and measures of agreement provides the formulas that PROC FREQ uses to compute the AGREE statistics. For information on the use and interpretation of these statistics, refer to Agresti (1990), Agresti (1996), Fleiss (1981), and the other references cited for each statistic.

McNemar's Test

PROC FREQ computes McNemar's test for 2×2 tables when you specify the AGREE option. McNemar's test is appropriate when you are analyzing data from matched pairs of subjects with a dichotomous (yes-no) response. It tests the null hypothesis of marginal homogeneity, or $p_{1\cdot} = p_{\cdot 1}$. McNemar's test is computed as

$$Q_M = \frac{(n_{12} - n_{21})^2}{n_{12} + n_{21}}$$

Under the null hypothesis, Q_M has an asymptotic chi-square distribution with one degree of freedom. Refer to McNemar (1947), as well as the references cited in the preceding section. In addition to the asymptotic test, PROC FREQ also computes the exact p-value for McNemar's test when you specify the MCNEM option in the EXACT statement.

Bowker's Test of Symmetry

For Bowker's test of symmetry, the null hypothesis is that the probabilities in the square table satisfy symmetry or that $p_{ij} = p_{ji}$ for all pairs of table cells. When there are more than two categories, Bowker's test of symmetry is calculated as

$$Q_B = \sum_{i<j}\sum \frac{(n_{ij} - n_{ji})^2}{n_{ij} + n_{ji}}$$

For large samples, Q_B has an asymptotic chi-square distribution with $R(R-1)/2$ degrees of freedom under the null hypothesis of symmetry of the expected counts. Refer to Bowker (1948). For two categories, this test of symmetry is identical to McNemar's test.

Simple Kappa Coefficient

The simple kappa coefficient, introduced by Cohen (1960), is a measure of interrater agreement:

$$\hat{\kappa} = \frac{P_o - P_e}{1 - P_e}$$

where $P_o = \sum_i p_{ii}$ and $P_e = \sum_i p_{i.}p_{.i}$. If the two response variables are viewed as two independent ratings of the n subjects, the kappa coefficient equals +1 when there is complete agreement of the raters. When the observed agreement exceeds chance agreement, kappa is positive, with its magnitude reflecting the strength of agreement. Although this is unusual in practice, kappa is negative when the observed agreement is less than chance agreement. The minimum value of kappa is between -1 and 0, depending on the marginal proportions.

The asymptotic variance of the simple kappa coefficient can be estimated by the following, according to Fleiss, Cohen, and Everitt (1969):

$$var = \frac{A + B - C}{(1 - P_e)^2 n}$$

where

$$A = \sum_i p_{ii} \left[1 - (p_{i\cdot} + p_{\cdot i})(1 - \hat{\kappa}) \right]^2$$

$$B = (1 - \hat{\kappa})^2 \sum\sum_{i \neq j} p_{ij} (p_{\cdot i} + p_{j\cdot})^2$$

and

$$C = \left[\hat{\kappa} - P_e (1 - \hat{\kappa}) \right]^2$$

PROC FREQ computes confidence limits for the simple kappa coefficient according to

$$\hat{\kappa} \quad \pm \quad (\ z_{\alpha/2} \ \times \ \sqrt{var}\ )$$

where $z_{\alpha/2}$ is the $100(1 - \alpha/2)$ percentile of the standard normal distribution. The value of α is determined by the value of the ALPHA= option, which, by default, equals 0.05 and produces 95% confidence limits.

To compute an asymptotic test for the kappa coefficient, PROC FREQ uses a standardized test statistic $\hat{\kappa}^*$, which has an asymptotic standard normal distribution under the null hypothesis that kappa equals zero. The standardized test statistic is computed as

$$\hat{\kappa}^* = \frac{\hat{\kappa}}{\sqrt{var_0(\hat{\kappa})}}$$

where $var_0(\hat{\kappa})$ is the variance of the kappa coefficient under the null hypothesis.

$$var_0(\hat{\kappa}) = \frac{P_e + P_e^2 - \sum_i p_{i\cdot}p_{\cdot i}(p_{i\cdot} + p_{\cdot i})}{(1 - P_e)^2 n}$$

Refer to Fleiss (1981).

In addition to the asymptotic test for kappa, PROC FREQ computes the exact test when you specify the KAPPA or AGREE option in the EXACT statement. See the section "Exact Statistics" beginning on page 142 for information on exact tests.

Weighted Kappa Coefficient

The weighted kappa coefficient is a generalization of the simple kappa coefficient, using weights to quantify the relative difference between categories. For 2×2 tables, the weighted kappa coefficient equals the simple kappa coefficient. PROC FREQ displays the weighted kappa coefficient only for tables larger than 2×2. PROC FREQ computes the weights from the column scores, using either the Cicchetti-Allison weight type or the Fleiss-Cohen weight type, both of which are described in the following section. The weights w_{ij} are constructed so that $0 \le w_{ij} < 1$ for all $i \ne j$, $w_{ii} = 1$ for all i, and $w_{ij} = w_{ji}$. The weighted kappa coefficient is defined as

$$\hat{\kappa}_w = \frac{P_{o(w)} - P_{e(w)}}{1 - P_{e(w)}}$$

where

$$P_{o(w)} = \sum_i \sum_j w_{ij} p_{ij}$$

and

$$P_{e(w)} = \sum_i \sum_j w_{ij} p_{i\cdot} p_{\cdot j}$$

The asymptotic variance of the weighted kappa coefficient can be estimated by the following, according to Fleiss, Cohen, and Everitt (1969):

$$var = \frac{\sum_i \sum_j p_{ij} \left[w_{ij} - (\overline{w}_{i\cdot} + \overline{w}_{\cdot j})(1 - \hat{\kappa}_w) \right]^2 - \left[\hat{\kappa}_w - P_{e(w)}(1 - \hat{\kappa}_w) \right]^2}{(1 - P_{e(w)})^2 n}$$

where

$$\overline{w}_{i\cdot} = \sum_j p_{\cdot j} w_{ij}$$

and

$$\overline{w}_{\cdot j} = \sum_i p_{i\cdot} w_{ij}$$

PROC FREQ computes confidence limits for the weighted kappa coefficient according to

$$\hat{\kappa}_w \quad \pm \quad (\; z_{\alpha/2} \; \times \; \sqrt{var} \;)$$

where $z_{\alpha/2}$ is the $100(1-\alpha/2)$ percentile of the standard normal distribution. The value of α is determined by the value of the ALPHA= option, which, by default, equals 0.05 and produces 95% confidence limits.

To compute an asymptotic test for the weighted kappa coefficient, PROC FREQ uses a standardized test statistic $\hat{\kappa}_w^*$, which has an asymptotic standard normal distribution under the null hypothesis that weighted kappa equals zero. The standardized test statistic is computed as

$$\hat{\kappa}_w^* = \frac{\hat{\kappa}_w}{\sqrt{var_0(\hat{\kappa}_w)}}$$

where $var_0(\hat{\kappa}_w)$ is the variance of the weighted kappa coefficient under the null hypothesis.

$$var_0(\hat{\kappa}_w) = \frac{\sum_i \sum_j p_{i\cdot} p_{\cdot j} \left[w_{ij} - (\overline{w}_{i\cdot} + \overline{w}_{\cdot j}) \right]^2 - P_{e(w)}^2}{(1 - P_{e(w)})^2 n}$$

Refer to Fleiss (1981).

In addition to the asymptotic test for weighted kappa, PROC FREQ computes the exact test when you specify the WTKAP or AGREE option in the EXACT statement. See the section "Exact Statistics" beginning on page 142 for information on exact tests.

Weights

PROC FREQ computes kappa coefficient weights using the column scores and one of two available weight types. The column scores are determined by the SCORES= option in the TABLES statement. The two available weight types are Cicchetti-Allison and Fleiss-Cohen, and PROC FREQ uses the Cicchetti-Allison type by default. If you specify (WT=FC) with the AGREE option, then PROC FREQ uses the Fleiss-Cohen weight type to construct kappa weights.

PROC FREQ computes Cicchetti-Allison kappa coefficient weights using a form similar to that given by Cicchetti and Allison (1971).

$$w_{ij} = 1 - \frac{|C_i - C_j|}{C_C - C_1}$$

where C_i is the score for column i, and C is the number of categories or columns. You can specify the score type using the SCORES= option in the TABLES statement; if you do not specify the SCORES= option, PROC FREQ uses table scores. For numeric variables, table scores are the values of the numeric row and column headings. You can assign numeric values to the categories in a way that reflects their level of similarity. For example, suppose you have four categories and order them according to similarity. If you assign them values of 0, 2, 4, and 10, the following weights are used for computing the weighted kappa coefficient: w_{12} = 0.8, w_{13} = 0.6, w_{14} = 0, w_{23} = 0.8, w_{24} = 0.2, and w_{34} = 0.4. Note that when there are only two categories (that is, C = 2), the weighted kappa coefficient is identical to the simple kappa coefficient.

If you specify (WT=FC) with the AGREE option in the TABLES statement, PROC FREQ computes Fleiss-Cohen kappa coefficient weights using a form similar to that given by Fleiss and Cohen (1973).

$$w_{ij} = 1 - \frac{(C_i - C_j)^2}{(C_C - C_1)^2}$$

For the preceding example, the weights used for computing the weighted kappa coefficient are: w_{12} = 0.96, w_{13} = 0.84, w_{14} = 0, w_{23} = 0.96, w_{24} = 0.36, and w_{34} = 0.64.

Overall Kappa Coefficient

When there are multiple strata, PROC FREQ combines the stratum-level estimates of kappa into an overall estimate of the supposed common value of kappa. Assume there are q strata, indexed by $h = 1, 2, \ldots, q$, and let $var(\hat{\kappa}_h)$ denote the squared standard error of $\hat{\kappa}_h$. Then the estimate of the overall kappa, according to Fleiss (1981), is computed as

$$\hat{\kappa}_{overall} = \sum_{h=1}^{q} \frac{\hat{\kappa}_h}{var(\hat{\kappa}_h)} \; / \; \sum_{h=1}^{q} \frac{1}{var(\hat{\kappa}_h)}$$

PROC FREQ computes an estimate of the overall weighted kappa in a similar manner.

Tests for Equal Kappa Coefficients

When there are multiple strata, the following chi-square statistic tests whether the stratum-level values of kappa are equal.

$$Q_K = \sum_{h=1}^{q} \frac{(\hat{\kappa}_h - \hat{\kappa}_{overall})^2}{var(\hat{\kappa}_h)}$$

Under the null hypothesis of equal kappas over the q strata, Q_K has an asymptotic chi-square distribution with $q - 1$ degrees of freedom. PROC FREQ computes a test for equal weighted kappa coefficients in a similar manner.

Cochran's *Q* Test

Cochran's Q is computed for multi-way tables when each variable has two levels, that is, for $2 \times 2 \cdots \times 2$ tables. Cochran's Q statistic is used to test the homogeneity of the one-dimensional margins. Let m denote the number of variables and N denote the total number of subjects. Then Cochran's Q statistic is computed as

$$Q_C = (m-1)\frac{m\sum_{j=1}^{m} T_j^2 - T^2}{mT - \sum_{k=1}^{N} S_k^2}$$

where T_j is the number of positive responses for variable j, T is the total number of positive responses over all variables, and S_k is the number of positive responses for subject k. Under the null hypothesis, Cochran's Q is an approximate chi-square statistic with $m - 1$ degrees of freedom. Refer to Cochran (1950). When there are only two binary response variables ($m = 2$), Cochran's Q simplifies to McNemar's test. When there are more than two response categories, you can test for marginal homogeneity using the repeated measures capabilities of the CATMOD procedure.

Tables with Zero Rows and Columns

The AGREE statistics are defined only for square tables, where the number of rows equals the number of columns. If the table is not square, PROC FREQ does not compute AGREE statistics. In the kappa statistic framework, where two independent raters assign ratings to each of n subjects, suppose one of the raters does not use all possible r rating levels. If the corresponding table has r rows but only $r-1$ columns, then the table is not square, and PROC FREQ does not compute the AGREE statistics. To create a square table in this situation, use the ZEROS option in the WEIGHT statement, which requests that PROC FREQ include observations with zero weights in the analysis. And input zero-weight observations to represent any rating levels that are not used by a rater, so that the input data set has at least one observation for each possible rater and rating combination. This includes all rating levels in the analysis, whether or not all levels are actually assigned by both raters. The resulting table is a square table, $r \times r$, and so all AGREE statistics can be computed.

For more information, see the description of the ZEROS option. By default, PROC FREQ does not process observations that have zero weights, because these observations do not contribute to the total frequency count, and because any resulting zero-weight row or column causes many of the tests and measures of association to be undefined. However, kappa statistics are defined for tables with a zero-weight row or column, and the ZEROS option allows input of zero-weight observations so you can construct the tables needed to compute kappas.

Cochran-Mantel-Haenszel Statistics

For *n*-way crosstabulation tables, consider the following example:

```
proc freq;
   tables A*B*C*D / cmh;
run;
```

The CMH option in the TABLES statement gives a stratified statistical analysis of the relationship between C and D, after controlling for A and B. The stratified analysis provides a way to adjust for the possible confounding effects of A and B without being forced to estimate parameters for them. The analysis produces Cochran-Mantel-Haenszel statistics, and for 2×2 tables, it includes estimation of the common odds ratio, common relative risks, and the Breslow-Day test for homogeneity of the odds ratios.

Let the number of strata be denoted by q, indexing the strata by $h = 1, 2, \ldots, q$. Each stratum contains a contingency table with X representing the row variable and Y representing the column variable. For table h, denote the cell frequency in row i and column j by n_{hij}, with corresponding row and column marginal totals denoted by $n_{hi.}$ and $n_{h.j}$, and the overall stratum total by n_h.

Because the formulas for the Cochran-Mantel-Haenszel statistics are more easily defined in terms of matrices, the following notation is used. Vectors are presumed to be column vectors unless they are transposed $(')$.

$$
\begin{array}{rcll}
\mathbf{n}'_{hi} & = & (n_{hi1}, n_{hi2}, \ldots, n_{hiC}) & (1 \times C) \\
\mathbf{n}'_{h} & = & (\mathbf{n}'_{h1}, \mathbf{n}'_{h2}, \ldots, \mathbf{n}'_{hR}) & (1 \times RC) \\
p_{hi\cdot} & = & \frac{n_{hi\cdot}}{n_h} & (1 \times 1) \\
p_{h\cdot j} & = & \frac{n_{h\cdot j}}{n_h} & (1 \times 1) \\
\mathbf{P}'_{h*\cdot} & = & (p_{h1\cdot}, p_{h2\cdot}, \ldots, p_{hR\cdot}) & (1 \times R) \\
\mathbf{P}'_{h\cdot *} & = & (p_{h\cdot 1}, p_{h\cdot 2}, \ldots, p_{h\cdot C}) & (1 \times C)
\end{array}
$$

Assume that the strata are independent and that the marginal totals of each stratum are fixed. The null hypothesis, H_0, is that there is no association between X and Y in any of the strata. The corresponding model is the multiple hypergeometric; this implies that, under H_0, the expected value and covariance matrix of the frequencies are, respectively,

$$\mathbf{m}_h = \mathbf{E}[\mathbf{n}_h \mid H_0] = n_h(\mathbf{P}_{h\cdot *} \otimes \mathbf{P}_{h*\cdot})$$

and

$$\mathbf{var}[\mathbf{n}_h \mid H_0] = c\left((\mathbf{D}_{\mathbf{P}h\cdot *} - \mathbf{P}_{h\cdot *}\mathbf{P}'_{h\cdot *}) \otimes (\mathbf{D}_{\mathbf{P}h*\cdot} - \mathbf{P}_{h*\cdot}\mathbf{P}'_{h*\cdot}) \right)$$

where

$$c = \frac{n_h^2}{n_h - 1}$$

and where $\otimes$ denotes Kronecker product multiplication and $\mathbf{D_a}$ is a diagonal matrix with elements of $\mathbf{a}$ on the main diagonal.

The generalized CMH statistic (Landis, Heyman, and Koch 1978) is defined as

$$Q_{\mathrm{CMH}} = \mathbf{G}'\mathbf{V_G}^{-1}\mathbf{G}$$

where

$$\begin{aligned}
\mathbf{G} &= \sum_h \mathbf{B}_h(\mathbf{n}_h - \mathbf{m}_h) \\
\mathbf{V_G} &= \sum_h \mathbf{B}_h \left(\mathbf{Var}(\mathbf{n}_h \mid H_0)\right) \mathbf{B}_h'
\end{aligned}$$

and where

$$\mathbf{B}_h = \mathbf{C}_h \otimes \mathbf{R}_h$$

is a matrix of fixed constants based on column scores $\mathbf{C}_h$ and row scores $\mathbf{R}_h$. When the null hypothesis is true, the CMH statistic has an asymptotic chi-square distribution with degrees of freedom equal to the rank of $\mathbf{B}_h$. If $\mathbf{V_G}$ is found to be singular, PROC FREQ prints a message and sets the value of the CMH statistic to missing.

PROC FREQ computes three CMH statistics using this formula for the generalized CMH statistic, with different row and column score definitions for each statistic. The CMH statistics that PROC FREQ computes are the correlation statistic, the ANOVA (row mean scores) statistic, and the general association statistic. These statistics test the null hypothesis of no association against different alternative hypotheses. The following sections describe the computation of these CMH statistics.

CAUTION: The CMH statistics have low power for detecting an association in which the patterns of association for some of the strata are in the opposite direction of the patterns displayed by other strata. Thus, a nonsignificant CMH statistic suggests either that there is no association or that no pattern of association has enough strength or consistency to dominate any other pattern.

Correlation Statistic

The correlation statistic, popularized by Mantel and Haenszel (1959) and Mantel (1963), has one degree of freedom and is known as the Mantel-Haenszel statistic.

The alternative hypothesis for the correlation statistic is that there is a linear association between X and Y in at least one stratum. If either X or Y does not lie on an ordinal (or interval) scale, then this statistic is not meaningful.

To compute the correlation statistic, PROC FREQ uses the formula for the generalized CMH statistic with the row and column scores determined by the SCORES= option in the TABLES statement. See the section "Scores" on page 102 for more information on the available score types. The matrix of row scores $\mathbf{R}_h$ has dimension $1 \times R$, and the matrix of column scores $\mathbf{C}_h$ has dimension $1 \times C$.

When there is only one stratum, this CMH statistic reduces to $(n-1)r^2$, where r is the Pearson correlation coefficient between X and Y. When nonparametric (RANK or RIDIT) scores are specified, then the statistic reduces to $(n-1)r_s^2$, where r_s is the Spearman rank correlation coefficient between X and Y. When there is more than one stratum, then this CMH statistic becomes a stratum-adjusted correlation statistic.

ANOVA (Row Mean Scores) Statistic

The ANOVA statistic can be used only when the column variable Y lies on an ordinal (or interval) scale so that the mean score of Y is meaningful. For the ANOVA statistic, the mean score is computed for each row of the table, and the alternative hypothesis is that, for at least one stratum, the mean scores of the R rows are unequal. In other words, the statistic is sensitive to location differences among the R distributions of Y.

The matrix of column scores $\mathbf{C}_h$ has dimension $1 \times C$, the column scores are determined by the SCORES= option.

The matrix of row scores $\mathbf{R}_h$ has dimension $(R-1) \times R$ and is created internally by PROC FREQ as

$$\mathbf{R}_h = [\mathbf{I}_{R-1}, -\mathbf{J}_{R-1}]$$

where $\mathbf{I}_{R-1}$ is an identity matrix of rank $R-1$, and $\mathbf{J}_{R-1}$ is an $(R-1) \times 1$ vector of ones. This matrix has the effect of forming $R-1$ independent contrasts of the R mean scores.

When there is only one stratum, this CMH statistic is essentially an analysis of variance (ANOVA) statistic in the sense that it is a function of the variance ratio F statistic that would be obtained from a one-way ANOVA on the dependent variable Y. If nonparametric scores are specified in this case, then the ANOVA statistic is a Kruskal-Wallis test.

If there is more than one stratum, then this CMH statistic corresponds to a stratum-adjusted ANOVA or Kruskal-Wallis test. In the special case where there is one subject per row and one subject per column in the contingency table of each stratum, this CMH statistic is identical to Friedman's chi-square. See Example 2.8 on page 180 for an illustration.

General Association Statistic

The alternative hypothesis for the general association statistic is that, for at least one stratum, there is some kind of association between X and Y. This statistic is always interpretable because it does not require an ordinal scale for either X or Y.

For the general association statistic, the matrix $\mathbf{R}_h$ is the same as the one used for the ANOVA statistic. The matrix $\mathbf{C}_h$ is defined similarly as

$$\mathbf{C}_h = [\mathbf{I}_{C-1}, -\mathbf{J}_{C-1}]$$

PROC FREQ generates both score matrices internally. When there is only one stratum, then the general association CMH statistic reduces to $Q_P(n-1)/n$, where Q_P is the Pearson chi-square statistic. When there is more than one stratum, then the CMH statistic becomes a stratum-adjusted Pearson chi-square statistic. Note that a similar adjustment can be made by summing the Pearson chi-squares across the strata. However, the latter statistic requires a large sample size in each stratum to support the resulting chi-square distribution with $q(R-1)(C-1)$ degrees of freedom. The CMH statistic requires only a large overall sample size since it has only $(R-1)(C-1)$ degrees of freedom.

Refer to Cochran (1954); Mantel and Haenszel (1959); Mantel (1963); Birch (1965); Landis, Heyman, and Koch (1978).

Adjusted Odds Ratio and Relative Risk Estimates

The CMH option provides adjusted odds ratio and relative risk estimates for stratified 2×2 tables. For each of these measures, PROC FREQ computes the Mantel-Haenszel estimate and the logit estimate. These estimates apply to *n*-way table requests in the TABLES statement, when the row and column variables both have only two levels.

For example,

```
proc freq;
   tables A*B*C*D / cmh;
run;
```

In this example, if the row and columns variables C and D both have two levels, PROC FREQ provides odds ratio and relative risk estimates, adjusting for the confounding variables A and B.

The choice of an appropriate measure depends on the study design. For case-control (retrospective) studies, the odds ratio is appropriate. For cohort (prospective) or cross-sectional studies, the relative risk is appropriate. See the section "Odds Ratio and Relative Risks for 2 x 2 Tables" beginning on page 122 for more information on these measures.

Throughout this section, z denotes the $100(1-\alpha/2)$ percentile of the standard normal distribution.

Odds Ratio, Case-Control Studies

Mantel-Haenszel Estimator

The Mantel-Haenszel estimate of the common odds ratio is computed as

$$\mathrm{OR}_{\mathrm{MH}} = \frac{\sum_h n_{h11}\, n_{h22}/n_h}{\sum_h n_{h12}\, n_{h21}/n_h}$$

It is always computed unless the denominator is zero. Refer to Mantel and Haenszel (1959) and Agresti (1990).

Using the estimated variance for $\log(\mathrm{OR_{MH}})$ given by Robins, Breslow, and Greenland (1986), PROC FREQ computes the corresponding $100(1-\alpha)\%$ confidence limits for the odds ratio as

$$(\ \mathrm{OR_{MH}} \cdot \exp(-z\hat{\sigma}),\ \mathrm{OR_{MH}} \cdot \exp(z\hat{\sigma})\ )$$

where

$$\begin{aligned}
\hat{\sigma}^2 &= \hat{var}[\, ln(\mathrm{OR_{MH}})\,] \\
&= \frac{\sum_h (n_{h11}+n_{h22})(n_{h11}\, n_{h22})/n_h^2}{2\left(\sum_h n_{h11}\, n_{h22}/n_h\right)^2} \\
&\quad + \frac{\sum_h [(n_{h11}+n_{h22})(n_{h12}\, n_{h21}) + (n_{h12}+n_{h21})(n_{h11}\, n_{h22})]/n_h^2}{2\left(\sum_h n_{h11}\, n_{h22}/n_h\right)\left(\sum_h n_{h12}\, n_{h21}/n_h\right)} \\
&\quad + \frac{\sum_h (n_{h12}+n_{h21})(n_{h12}\, n_{h21})/n_h^2}{2\left(\sum_h n_{h12}\, n_{h21}/n_h\right)^2}
\end{aligned}$$

Note that the Mantel-Haenszel odds ratio estimator is less sensitive to small n_h than the logit estimator.

Logit Estimator

The adjusted logit estimate of the odds ratio (Woolf 1955) is computed as

$$\mathrm{OR_L} = \exp\left(\frac{\sum_h w_h \ln(\mathrm{OR}_h)}{\sum_h w_h}\right)$$

and the corresponding $100(1-\alpha)\%$ confidence limits are

$$\left(\mathrm{OR_L} \cdot \exp\left(\frac{-z}{\sqrt{\sum_h w_h}}\right),\ \mathrm{OR_L} \cdot \exp\left(\frac{z}{\sqrt{\sum_h w_h}}\right)\right)$$

where OR_h is the odds ratio for stratum h, and

$$w_h = \frac{1}{var(\ln \mathrm{OR}_h)}$$

If any cell frequency in a stratum h is zero, then PROC FREQ adds 0.5 to each cell of the stratum before computing OR_h and w_h (Haldane 1955), and prints a warning.

Exact Confidence Limits for the Common Odds Ratio

When you specify the COMOR option in the EXACT statement, PROC FREQ computes exact confidence limits for the common odds ratio for stratified 2×2 tables.

This computation assumes that the odds ratio is constant over all the 2×2 tables. Exact confidence limits are constructed from the distribution of $S = \sum_h n_{h11}$, conditional on the marginal totals of the 2×2 tables.

Because this is a discrete problem, the confidence coefficient for these exact confidence limits is not exactly $1 - \alpha$ but is at least $1 - \alpha$. Thus, these confidence limits are conservative. Refer to Agresti (1992).

PROC FREQ computes exact confidence limits for the common odds ratio with an algorithm based on that presented by Vollset, Hirji, and Elashoff (1991). Refer also to Mehta, Patel, and Gray (1985).

Conditional on the marginal totals of 2×2 table h, let the random variable S_h denote the frequency of table cell $(1, 1)$. Given the row totals $n_{h1\cdot}$ and $n_{h2\cdot}$ and column totals $n_{h\cdot 1}$ and $n_{h\cdot 2}$, the lower and upper bounds for S_h are l_h and u_h,

$$\begin{aligned} l_h &= \max\,(\,0,\;\; n_{h1\cdot} - n_{h\cdot 2}\,) \\ u_h &= \min\,(\,n_{h1\cdot},\;\; n_{h\cdot 1}\,) \end{aligned}$$

Let C_{s_h} denote the hypergeometric coefficient,

$$C_{s_h} = \binom{n_{h\cdot 1}}{s_h}\binom{n_{h\cdot 2}}{n_{h1\cdot} - s_h}$$

and let ϕ denote the common odds ratio. Then the conditional distribution of S_h is

$$P(\,S_h = s_h \mid n_{1\cdot},\, n_{\cdot 1},\, n_{\cdot 2}\,) \;=\; C_{s_h}\,\phi^{\,s_h} \;/\; \sum_{x\,=\,l_h}^{x\,=\,u_h} C_x\,\phi^{\,x}$$

Summing over all the 2×2 tables, $S = \sum_h S_h$, and the lower and upper bounds of S are l and u,

$$l = \sum_h l_h \qquad \text{and} \qquad u = \sum_h u_h$$

The conditional distribution of the sum S is

$$P(\,S = s \mid n_{h1\cdot},\, n_{h\cdot 1},\, n_{h\cdot 2};\;\; h = 1, \ldots, q\,) \;=\; C_s\,\phi^{\,s} \;/\; \sum_{x\,=\,l}^{x\,=\,u} C_x\,\phi^{\,x}$$

where

$$C_s = \sum_{s_1 + \ldots. + s_q\,=\,s} \left(\prod_h C_{s_h} \right)$$

Let s_0 denote the observed sum of cell (1,1) frequencies over the q tables. The following two equations are solved iteratively for lower and upper confidence limits for the common odds ratio, ϕ_1 and ϕ_2,

$$\sum_{x\,=\,s_o}^{x\,=\,u} C_x\,\phi_1^{\,x} \;\Big/\; \sum_{x\,=\,l}^{x\,=\,u} C_x\,\phi_1^{\,x} \quad = \quad \alpha/2$$

$$\sum_{x\,=\,l}^{x\,=\,s_0} C_x\,\phi_2^{\,x} \;\Big/\; \sum_{x\,=\,l}^{x\,=\,u} C_x\,\phi_2^{\,x} \quad = \quad \alpha/2$$

When the observed sum s_0 equals the lower bound l, then PROC FREQ sets the lower exact confidence limit to zero and determines the upper limit with level α. Similarly, when the observed sum s_0 equals the upper bound u, then PROC FREQ sets the upper exact confidence limit to infinity and determines the lower limit with level α.

When you specify the COMOR option in the EXACT statement, PROC FREQ also computes the exact test that the common odds ratio equals one. Setting $\phi = 1$, the conditional distribution of the sum S under the null hypothesis becomes

$$P_0(\;S = s \;\mid\; n_{h1\cdot},\; n_{h\cdot 1},\; n_{h\cdot 2};\;\; h = 1, \ldots, q\;) \quad = \quad C_s \;\Big/\; \sum_{x\,=\,l}^{x\,=\,u} C_x$$

The point probability for this exact test is the probability of the observed sum s_0 under the null hypothesis, conditional on the marginals of the stratified 2×2 tables, and is denoted by $P_0(s_0)$. The expected value of S under the null hypothesis is

$$E_0(S) \quad = \quad \sum_{x\,=\,l}^{x\,=\,u} x\,C_x \;\Big/\; \sum_{x\,=\,l}^{x\,=\,u} C_x$$

The one-sided exact *p*-value is computed from the conditional distribution as $P_0(S >= s_0)$ or $P_0(S \leq s_0)$, depending on whether the observed sum s_0 is greater or less than $E_0(S)$.

$$P_1 \quad = \quad P_0(\;S >= s_0\;) \quad = \quad \sum_{x\,=\,s_0}^{x\,=\,u} C_x \;\Big/\; \sum_{x\,=\,l}^{x\,=\,u} C_x \qquad \text{if } s_0 \;>\; E_0(S)$$

$$P_1 \quad = \quad P_0(\;S <= s_0\;) \quad = \quad \sum_{x\,=\,l}^{x\,=\,s_0} C_x \;\Big/\; \sum_{x\,=\,l}^{x\,=\,u} C_x \qquad \text{if } s_0 \;\leq\; E_0(S)$$

PROC FREQ computes two-sided *p*-values for this test according to three different definitions. A two-sided *p*-value is computed as twice the one-sided *p*-value, setting the result equal to one if it exceeds one.

$$P_2^{\,a} \;=\; 2 \;\times\; P_1$$

Additionally, a two-sided *p*-value is computed as the sum of all probabilities less than or equal to the point probability of the observed sum s_0, summing over all possible values of s, $\ l \le s \le u$.

$$P_2{}^b \; = \sum_{l \le s \le u:\; P_0(s) \le P_0(s_0)} P_0(s)$$

Also, a two-sided *p*-value is computed as the sum of the one-sided *p*-value and the corresponding area in the opposite tail of the distribution, equidistant from the expected value.

$$P_2{}^c \; = \; P_0\; (\; |S - E_0(S)| \; \ge \; |s_0 - E_0(S)| \;)$$

Relative Risks, Cohort Studies

Mantel-Haenszel Estimator

The Mantel-Haenszel estimate of the common relative risk for column 1 is computed as

$$\mathrm{RR}_{\mathrm{MH}} = \frac{\sum_h n_{h11}\; n_{h2\cdot}/n_h}{\sum_h n_{h21}\; n_{h1\cdot}/n_h}$$

It is always computed unless the denominator is zero. Refer to Mantel and Haenszel (1959) and Agresti (1990).

Using the estimated variance for $\log(\mathrm{RR}_{\mathrm{MH}})$ given by Greenland and Robins (1985), PROC FREQ computes the corresponding $100(1-\alpha)$% confidence limits for the relative risk as

$$(\; \mathrm{RR}_{\mathrm{MH}} \cdot \exp(-z\hat{\sigma}), \; \mathrm{RR}_{\mathrm{MH}} \cdot \exp(z\hat{\sigma}) \;)$$

where

$$\begin{aligned} \hat{\sigma}^2 \quad &= \quad v\hat{a}r[\; ln(\mathrm{RR}_{\mathrm{MH}}) \;] \\ &= \quad \frac{\sum_h (n_{h1\cdot}\; n_{h2\cdot}\; n_{h\cdot 1} - n_{h11}\; n_{h21}\; n_h)/n_h^2}{(\sum_h n_{h11}\; n_{h2\cdot}/n_h)\; (\sum_h n_{h21}\; n_{h1\cdot}/n_h)} \end{aligned}$$

Logit Estimator

The adjusted logit estimate of the common relative risk for column 1 is computed as

$$\mathrm{RR}_{\mathrm{L}} = \exp\left(\frac{\sum_h w_h \ln \mathrm{RR}_h}{\sum w_h} \right)$$

and the corresponding $100(1-\alpha)$% confidence limits are

$$\left(\mathrm{RR}_{\mathrm{L}} \exp\left(\frac{-z}{\sqrt{\sum_h w_h}} \right), \mathrm{RR}_{\mathrm{L}} \exp\left(\frac{z}{\sqrt{\sum_h w_h}} \right) \right)$$

where RR_h is the column 1 relative risk estimate for stratum h, and

$$w_h = \frac{1}{var(\ln \mathrm{RR}_h)}$$

If n_{h11} or n_{h21} is zero, then PROC FREQ adds 0.5 to each cell of the stratum before computing RR_h and w_h, and prints a warning. Refer to Kleinbaum, Kupper, and Morgenstern (1982, Sections 17.4 and 17.5).

Breslow-Day Test for Homogeneity of the Odds Ratios

When you specify the CMH option, PROC FREQ computes the Breslow-Day test for stratified analysis of 2×2 tables. It tests the null hypothesis that the odds ratios for the q strata are all equal. When the null hypothesis is true, the statistic has approximately a chi-square distribution with $q - 1$ degrees of freedom. Refer to Breslow and Day (1980) and Agresti (1996).

The Breslow-Day statistic is computed as

$$Q_{\mathrm{BD}} = \sum_h \frac{(n_{h11} - E(n_{h11} \mid \mathrm{OR}_{\mathrm{MH}}))^2}{var(n_{h11} \mid \mathrm{OR}_{\mathrm{MH}})}$$

where E and var denote expected value and variance, respectively. The summation does not include any table with a zero row or column. If $\mathrm{OR}_{\mathrm{MH}}$ equals zero or if it is undefined, then PROC FREQ does not compute the statistic and prints a warning message.

For the Breslow-Day test to be valid, the sample size should be relatively large in each stratum, and at least 80% of the expected cell counts should be greater than 5. Note that this is a stricter sample size requirement than the requirement for the Cochran-Mantel-Haenszel test for $q \times 2 \times 2$ tables, in that each stratum sample size (not just the overall sample size) must be relatively large. Even when the Breslow-Day test is valid, it may not be very powerful against certain alternatives, as discussed in Breslow and Day (1980).

If you specify the BDT option, PROC FREQ computes the Breslow-Day test with Tarone's adjustment, which subtracts an adjustment factor from Q_{BD} to make the resulting statistic asymptotically chi-square.

$$Q_{\mathrm{BDT}} = Q_{\mathrm{BD}} - \frac{(\sum_h (n_{h11} - E(n_{h11} \mid \mathrm{OR}_{\mathrm{MH}})))^2}{\sum_h var(n_{h11} \mid \mathrm{OR}_{\mathrm{MH}})}$$

Refer to Tarone (1985), Jones et al. (1989), and Breslow (1996).

Exact Statistics

Exact statistics can be useful in situations where the asymptotic assumptions are not met, and so the asymptotic *p*-values are not close approximations for the true *p*-values. Standard asymptotic methods involve the assumption that the test statistic follows a particular distribution when the sample size is sufficiently large. When the sample size is not large, asymptotic results may not be valid, with the asymptotic *p*-values differing perhaps substantially from the exact *p*-values. Asymptotic results may also be unreliable when the distribution of the data is sparse, skewed, or heavily tied. Refer to Agresti (1996) and Bishop, Fienberg, and Holland (1975). Exact computations are based on the statistical theory of exact conditional inference for contingency tables, reviewed by Agresti (1992).

In addition to computation of exact *p*-values, PROC FREQ provides the option of estimating exact *p*-values by Monte Carlo simulation. This can be useful for problems that are so large that exact computations require a great amount of time and memory, but for which asymptotic approximations may not be sufficient.

PROC FREQ provides exact *p*-values for the following tests for two-way tables: Pearson chi-square, likelihood-ratio chi-square, Mantel-Haenszel chi-square, Fisher's exact test, Jonckheere-Terpstra test, Cochran-Armitage test for trend, and McNemar's test. PROC FREQ also computes exact *p*-values for tests of hypotheses that the following statistics equal zero: Pearson correlation coefficient, Spearman correlation coefficient, simple kappa coefficient, and weighted kappa coefficient. Additionally, PROC FREQ computes exact confidence limits for the odds ratio for 2×2 tables. For stratified 2×2 tables, PROC FREQ computes exact confidence limits for the common odds ratio, as well as an exact test that the common odds ratio equals one. For one-way frequency tables, PROC FREQ provides the exact chi-square goodness-of-fit test (for equal proportions or for proportions or frequencies that you specify). Also for one-way tables, PROC FREQ provides exact confidence limits for the binomial proportion and an exact test for the binomial proportion value.

The following sections summarize the exact computational algorithms, define the exact *p*-values that PROC FREQ computes, discuss the computational resource requirements, and describe the Monte Carlo estimation option.

Computational Algorithms

PROC FREQ computes exact *p*-values for general $R \times C$ tables using the network algorithm developed by Mehta and Patel (1983). This algorithm provides a substantial advantage over direct enumeration, which can be very time-consuming and feasible only for small problems. Refer to Agresti (1992) for a review of algorithms for computation of exact *p*-values, and refer to Mehta, Patel, and Tsiatis (1984) and Mehta, Patel, and Senchaudhuri (1991) for information on the performance of the network algorithm.

The reference set for a given contingency table is the set of all contingency tables with the observed marginal row and column sums. Corresponding to this reference set, the network algorithm forms a directed acyclic network consisting of nodes in a number of stages. A path through the network corresponds to a distinct table in the reference set. The distances between nodes are defined so that the total distance of a path through the network is the corresponding value of the test statistic. At each node, the algorithm computes the shortest and longest path distances for all the paths that pass through that node. For statistics that can be expressed as a linear combination of cell frequencies multiplied by increasing row and column scores, PROC FREQ computes shortest and longest path distances using the algorithm given in Agresti, Mehta, and Patel (1990). For statistics of other forms, PROC FREQ computes an upper bound for the longest path and a lower bound for the shortest path, following the approach of Valz and Thompson (1994).

The longest and shortest path distances or bounds for a node are compared to the value of the test statistic to determine whether all paths through the node contribute to the *p*-value, none of the paths through the node contribute to the *p*-value, or neither of these situations occur. If all paths through the node contribute, the *p*-value is incremented accordingly, and these paths are eliminated from further analysis. If no paths contribute, these paths are eliminated from the analysis. Otherwise, the algorithm continues, still processing this node and the associated paths. The algorithm finishes when all nodes have been accounted for, incrementing the *p*-value accordingly, or eliminated.

In applying the network algorithm, PROC FREQ uses full precision to represent all statistics, row and column scores, and other quantities involved in the computations. Although it is possible to use rounding to improve the speed and memory requirements of the algorithm, PROC FREQ does not do this since it can result in reduced accuracy of the *p*-values.

For one-way tables, PROC FREQ computes the exact chi-square goodness-of-fit test by the method of Radlow and Alf (1975). PROC FREQ generates all possible one-way tables with the observed total sample size and number of categories. For each possible table, PROC FREQ compares its chi-square value with the value for the observed table. If the table's chi-square value is greater than or equal to the observed chi-square, PROC FREQ increments the exact *p*-value by the probability of that table, which is calculated under the null hypothesis using the multinomial frequency distribution. By default, the null hypothesis states that all categories have equal proportions. If you specify null hypothesis proportions or frequencies using the TESTP= or TESTF= option in the TABLES statement, then PROC FREQ calculates the exact chi-square test based on that null hypothesis.

For binomial proportions in one-way tables, PROC FREQ computes exact confidence limits using the F distribution method given in Collett (1991) and also described by Leemis and Trivedi (1996). PROC FREQ computes the exact test for a binomial proportion ($H_0\colon p = p_0$) by summing binomial probabilities over all alternatives. See the section "Binomial Proportion" on page 118 for details. By default, PROC FREQ uses $p_0 = 0.5$ as the null hypothesis proportion. Alternatively, you can specify the null hypothesis proportion with the P= option in the TABLES statement.

See the section "Odds Ratio and Relative Risks for 2 x 2 Tables" on page 122 for details on computation of exact confidence limits for the odds ratio for 2×2 tables. See the section "***Exact Confidence Limits for the Common Odds Ratio***" on page 138 for details on computation of exact confidence limits for the common odds ratio for stratified 2×2 tables.

Definition of *p*-Values

For several tests in PROC FREQ, the test statistic is nonnegative, and large values of the test statistic indicate a departure from the null hypothesis. Such tests include the Pearson chi-square, the likelihood-ratio chi-square, the Mantel-Haenszel chi-square, Fisher's exact test for tables larger than 2×2 tables, McNemar's test, and the one-way chi-square goodness-of-fit test. The exact *p*-value for these nondirectional tests is the sum of probabilities for those tables having a test statistic greater than or equal to the value of the observed test statistic.

There are other tests where it may be appropriate to test against either a one-sided or a two-sided alternative hypothesis. For example, when you test the null hypothesis that the true parameter value equals 0 ($T = 0$), the alternative of interest may be one-sided ($T \leq 0$, or $T \geq 0$) or two-sided ($T \neq 0$). Such tests include the Pearson correlation coefficient, Spearman correlation coefficient, Jonckheere-Terpstra test, Cochran-Armitage test for trend, simple kappa coefficient, and weighted kappa coefficient. For these tests, PROC FREQ outputs the right-sided *p*-value when the observed value of the test statistic is greater than its expected value. The right-sided *p*-value is the sum of probabilities for those tables having a test statistic greater than or equal to the observed test statistic. Otherwise, when the test statistic is less than or equal to its expected value, PROC FREQ outputs the left-sided *p*-value. The left-sided *p*-value is the sum of probabilities for those tables having a test statistic less than or equal to the one observed. The one-sided *p*-value P_1 can be expressed as

$$P_1 = \text{Prob (Test Statistic} \geq t) \qquad \text{if} \;\; t > E_0(T)$$

$$P_1 = \text{Prob (Test Statistic} \leq t) \qquad \text{if} \;\; t \leq E_0(T)$$

where t is the observed value of the test statistic and $E_0(T)$ is the expected value of the test statistic under the null hypothesis. PROC FREQ computes the two-sided *p*-value as the sum of the one-sided *p*-value and the corresponding area in the opposite tail of the distribution of the statistic, equidistant from the expected value. The two-sided *p*-value P_2 can be expressed as

$$P_2 = \text{Prob} \; (\, | \, \text{Test Statistic} - E_0(T) \, | \geq | \, t - E_0(T) \, | \,)$$

If you specify the POINT option in the EXACT statement, PROC FREQ also displays exact point probabilities for the test statistics. The exact point probability is the exact probability that the test statistic equals the observed value.

Computational Resources

PROC FREQ uses relatively fast and efficient algorithms for exact computations. These recently developed algorithms, together with improvements in computer power, make it feasible now to perform exact computations for data sets where previously only asymptotic methods could be applied. Nevertheless, there are still large problems that may require a prohibitive amount of time and memory for exact computations, depending on the speed and memory available on your computer. For large problems, consider whether exact methods are really needed or whether asymptotic methods might give results quite close to the exact results, while requiring much less computer time and memory. When asymptotic methods may not be sufficient for such large problems, consider using Monte Carlo estimation of exact *p*-values, as described in the section "Monte Carlo Estimation" on page 146.

A formula does not exist that can predict in advance how much time and memory are needed to compute an exact *p*-value for a certain problem. The time and memory required depend on several factors, including which test is being performed, the total sample size, the number of rows and columns, and the specific arrangement of the observations into table cells. Generally, larger problems (in terms of total sample size, number of rows, and number of columns) tend to require more time and memory. Additionally, for a fixed total sample size, time and memory requirements tend to increase as the number of rows and columns increases, since this corresponds to an increase in the number of tables in the reference set. Also for a fixed sample size, time and memory requirements increase as the marginal row and column totals become more homogeneous. Refer to Agresti, Mehta, and Patel (1990) and Gail and Mantel (1977).

At any time while PROC FREQ is computing exact *p*-values, you can terminate the computations by pressing the system interrupt key sequence (refer to the *SAS Companion* for your system) and choosing to stop computations. After you terminate exact computations, PROC FREQ completes all other remaining tasks. The procedure produces the requested output and reports missing values for any exact *p*-values that were not computed by the time of termination.

You can also use the MAXTIME= option in the EXACT statement to limit the amount of time PROC FREQ uses for exact computations. You specify a MAXTIME= value that is the maximum amount of clock time (in seconds) that PROC FREQ can use to compute an exact *p*-value. If PROC FREQ does not finish computing an exact *p*-value within that time, it terminates the computation and completes all other remaining tasks.

Monte Carlo Estimation

If you specify the option MC in the EXACT statement, PROC FREQ computes Monte Carlo estimates of the exact *p*-values instead of directly computing the exact *p*-values. Monte Carlo estimation can be useful for large problems that require a great amount of time and memory for exact computations but for which asymptotic approximations may not be sufficient. To describe the precision of each Monte Carlo estimate, PROC FREQ provides the asymptotic standard error and $100(1-\alpha)$% confidence limits. The confidence level α is determined by the ALPHA= option in the EXACT statement, which, by default, equals 0.01, and produces 99% confidence limits. The N=*n* option in the EXACT statement specifies the number of samples that PROC FREQ uses for Monte Carlo estimation; the default is 10000 samples. You can specify a larger value for n to improve the precision of the Monte Carlo estimates. Because larger values of n generate more samples, the computation time increases. Alternatively, you can specify a smaller value of n to reduce the computation time.

To compute a Monte Carlo estimate of an exact *p*-value, PROC FREQ generates a random sample of tables with the same total sample size, row totals, and column totals as the observed table. PROC FREQ uses the algorithm of Agresti, Wackerly, and Boyett (1979), which generates tables in proportion to their hypergeometric probabilities conditional on the marginal frequencies. For each sample table, PROC FREQ computes the value of the test statistic and compares it to the value for the observed table. When estimating a right-sided *p*-value, PROC FREQ counts all sample tables for which the test statistic is greater than or equal to the observed test statistic. Then the *p*-value estimate equals the number of these tables divided by the total number of tables sampled.

$$\begin{aligned} \hat{P}_{\mathrm{MC}} &= M \,/\, N \\ M &= \text{number of samples with } (\text{Test Statistic} \geq t) \\ N &= \text{total number of samples} \\ t &= \text{observed Test Statistic} \end{aligned}$$

PROC FREQ computes left-sided and two-sided *p*-value estimates in a similar manner. For left-sided *p*-values, PROC FREQ evaluates whether the test statistic for each sampled table is less than or equal to the observed test statistic. For two-sided *p*-values, PROC FREQ examines the sample test statistics according to the expression for P_2 given in the section "Asymptotic Tests" on page 109. The variable M is a binomially distributed variable with N trials and success probability p. It follows that the asymptotic standard error of the Monte Carlo estimate is

$$se(\hat{P}_{\mathrm{MC}}) = \sqrt{\hat{P}_{\mathrm{MC}}(1 - \hat{P}_{\mathrm{MC}})/(N-1)}$$

PROC FREQ constructs asymptotic confidence limits for the *p*-values according to

$$\hat{P}_{\mathrm{MC}} \pm z_{\alpha/2} \cdot se(\hat{P}_{\mathrm{MC}})$$

where $z_{\alpha/2}$ is the $100(1-\alpha/2)$ percentile of the standard normal distribution, and the confidence level α is determined by the ALPHA= option in the EXACT statement.

When the Monte Carlo estimate $\hat{P}_{\mathrm{MC}}$ equals 0, then PROC FREQ computes the confidence limits for the *p*-value as

$$(\, 0, \;\; 1 - \alpha^{(1/N)} \,)$$

When the Monte Carlo estimate $\hat{P}_{MC}$ equals 1, then PROC FREQ computes the confidence limits as

$$(\, \alpha^{(1/N)}, \;\; 1 \,)$$

Computational Resources

For each variable in a table request, PROC FREQ stores all of the levels in memory. If all variables are numeric and not formatted, this requires about 84 bytes for each variable level. When there are character variables or formatted numeric variables, the memory that is required depends on the formatted variable lengths, with longer formatted lengths requiring more memory. The number of levels for each variable is limited only by the largest integer that your operating environment can store.

For any single crosstabulation table requested, PROC FREQ builds the entire table in memory, regardless of whether the table has zero cell counts. Thus, if the numeric variables A, B, and C each have 10 levels, PROC FREQ requires 2520 bytes to store the variable levels for the table request A*B*C, as follows:

```
3 variables * 10 levels/variable * 84 bytes/level
```

In addition, PROC FREQ requires 8000 bytes to store the table cell frequencies

```
1000 cells * 8 bytes/cell
```

even though there may be only 10 observations.

When the variables have many levels or when there are many multiway tables, your computer may not have enough memory to construct the tables. If PROC FREQ runs out of memory while constructing tables, it stops collecting levels for the variable with the most levels and returns the memory that is used by that variable. The procedure then builds the tables that do not contain the disabled variables.

If there is not enough memory for your table request and if increasing the available memory is impractical, you can reduce the number of multiway tables or variable levels. If you are not using the CMH or AGREE option in the TABLES statement to compute statistics across strata, reduce the number of multiway tables by using PROC SORT to sort the data set by one or more of the variables or by using the DATA step to create an index for the variables. Then remove the sorted or indexed variables from the TABLES statement and include a BY statement that uses these variables. You can also reduce memory requirements by using a FORMAT statement in the PROC FREQ step to reduce the number of levels. Additionally, reducing the formatted variable lengths reduces the amount of memory that is needed to store the variable levels. For more information on using formats, see the "Grouping with Formats" section on page 99.

Output Data Sets

PROC FREQ produces two types of output data sets that you can use with other statistical and reporting procedures. These data sets are produced as follows:

- Specifying a TABLES statement with an OUT= option creates an output data set that contains frequency or crosstabulation table counts and percentages.
- Specifying an OUTPUT statement creates an output data set that contains statistics.

PROC FREQ does not display the output data sets. Use PROC PRINT, PROC REPORT, or any other SAS reporting tool to display an output data set.

Contents of the TABLES Statement Output Data Set

The OUT= option in the TABLES statement creates an output data set that contains one observation for each combination of the variable values (or table cell) in the last table request. By default, each observation contains the frequency and percentage for the table cell. When the input data set contains missing values, the output data set also contains an observation with the frequency of missing values. The output data set includes the following variables:

- BY variables
- table request variables, such as A, B, C, and D in the table request A*B*C*D
- COUNT, a variable containing the cell frequency
- PERCENT, a variable containing the cell percentage

If you specify the OUTEXPECT and OUTPCT options in the TABLES statement, the output data set also contains expected frequencies and row, column, and table percentages, respectively. The additional variables are

- EXPECTED, a variable containing the expected frequency
- PCT_TABL, a variable containing the percentage of two-way table frequency, for *n*-way tables where $n > 2$
- PCT_ROW, a variable containing the percentage of row frequency
- PCT_COL, a variable containing the percentage of column frequency

If you specify the OUTCUM option in the TABLES statement, the output data set also contains cumulative frequencies and cumulative percentages for one-way tables. The additional variables are

- CUM_FREQ, a variable containing the cumulative frequency
- CUM_PCT, a variable containing the cumulative percentage

The OUTCUM option has no effect for two-way or multiway tables.

When you submit the following statements

```
proc freq;
   tables A A*B / out=D;
run;
```

the output data set D contains frequencies and percentages for the last table request, A*B. If A has two levels (1 and 2), B has three levels (1,2, and 3), and no table cell count is zero or missing, the output data set D includes six observations, one for each combination of A and B. The first observation corresponds to A=1 and B=1; the second observation corresponds to A=1 and B=2; and so on. The data set includes the variables COUNT and PERCENT. The value of COUNT is the number of observations with the given combination of A and B values. The value of PERCENT is the percent of the total number of observations having that A and B combination.

When PROC FREQ combines different variable values into the same formatted level, the output data set contains the smallest internal value for the formatted level. For example, suppose a variable X has the values 1.1., 1.4, 1.7, 2.1, and 2.3. When you submit the statement

```
format X 1.;
```

in a PROC FREQ step, the formatted levels listed in the frequency table for X are 1 and 2. If you create an output data set with the frequency counts, the internal values of X are 1.1 and 1.7. To report the internal values of X when you display the output data set, use a format of 3.1 with X.

Contents of the OUTPUT Statement Output Data Set

The OUTPUT statement creates a SAS data set containing the statistics that PROC FREQ computes for the last table request. You specify which statistics to store in the output data set. There is an observation with the specified statistics for each stratum or two-way table. If PROC FREQ computes summary statistics for a stratified table, the output data set also contains a summary observation with those statistics.

The OUTPUT data set can include the following variables.

- BY variables
- variables that identify the stratum, such as A and B in the table request A*B*C*D
- variables that contain the specified statistics

The output data set also includes variables with the *p*-values and degrees of freedom, asymptotic standard error (ASE), or confidence limits when PROC FREQ computes these values for a specified statistic.

The variable names for the specified statistics in the output data set are the names of the options enclosed in underscores. PROC FREQ forms variable names for the corresponding *p*-values, degrees of freedom, or confidence limits by combining the name of the option with the appropriate prefix from the following list:

DF_	degrees of freedom
E_	asymptotic standard error (ASE)
L_	lower confidence limit
U_	upper confidence limit
E0_	ASE under the null hypothesis
Z_	standardized value
P_	*p*-value
P2_	two-sided *p*-value
PL_	left-sided *p*-value
PR_	right-sided *p*-value
XP_	exact *p*-value
XP2_	exact two-sided *p*-value
XPL_	exact left-sided *p*-value
XPR_	exact right-sided *p*-value
XPT_	exact point probability
XL_	exact lower confidence limit
XR_	exact upper confidence limit

For example, variable names created for the Pearson chi-square, its degrees of freedom, its p-values are _PCHI_, DF_PCHI, and P_PCHI, respectively.

If the length of the prefix plus the statistic option exceeds eight characters, PROC FREQ truncates the option so that the name of the new variable is eight characters long.

Displayed Output

Number of Variable Levels Table

If you specify the NLEVELS option in the PROC FREQ statement, PROC FREQ displays the "Number of Variable Levels" table. This table provides the number of levels for all variables named in the TABLES statements. PROC FREQ determines the variable levels from the formatted variable values. See "Grouping with Formats" for details. The "Number of Variable Levels" table contains the following information:

- Variable name
- Levels, which is the total number of levels of the variable
- Number of Nonmissing Levels, if there are missing levels for any of the variables
- Number of Missing Levels, if there are missing levels for any of the variables

One-Way Frequency Tables

PROC FREQ displays one-way frequency tables for all one-way table requests in the TABLES statements, unless you specify the NOPRINT option in the PROC statement or the NOPRINT option in the TABLES statement. For a one-way table showing the frequency distribution of a single variable, PROC FREQ displays the following information:

- the name of the variable and its values
- Frequency counts, giving the number of observations that have each value
- specified Test Frequency counts, if you specify the CHISQ and TESTF= options to request a chi-square goodness-of-fit test for specified frequencies
- Percent, giving the percentage of the total number of observations with that value. (The NOPERCENT option suppresses this information.)
- specified Test Percents, if you specify the CHISQ and TESTP= options to request a chi-square goodness-of-fit test for specified percents. (The NOPERCENT option suppresses this information.)
- Cumulative Frequency counts, giving the sum of the frequency counts of that value and all other values listed above it in the table. The last cumulative frequency is the total number of nonmissing observations. (The NOCUM option suppresses this information.)
- Cumulative Percent values, giving the percentage of the total number of observations with that value and all others previously listed in the table. (The NOCUM or the NOPERCENT option suppresses this information.)

The one-way table also displays the Frequency Missing, or the number of observations with missing values.

Statistics for One-Way Frequency Tables

For one-way tables, two statistical options are available in the TABLES statement. The CHISQ option provides a chi-square goodness-of-fit test, and the BINOMIAL option provides binomial proportion statistics. PROC FREQ displays the following information, unless you specify the NOPRINT option in the PROC statement:

- If you specify the CHISQ option for a one-way table, PROC FREQ provides a chi-square goodness-of-fit test, displaying the Chi-Square statistic, the degrees of freedom (DF), and the probability value (Pr > ChiSq). If you specify the CHISQ option in the EXACT statement, PROC FREQ also displays the exact probability value for this test. If you specify the POINT option with the CHISQ option in the EXACT statement, PROC FREQ displays the exact point probability for the test statistic.
- If you specify the BINOMIAL option for a one-way table, PROC FREQ displays the estimate of the binomial Proportion, which is the proportion of observations in the first class listed in the one-way table. PROC FREQ also displays the asymptotic standard error (ASE) and the asymptotic and exact confidence limits for this estimate. For the binomial proportion test, PROC FREQ displays the asymptotic standard error under the null hypothesis (ASE Under H0), the standardized test statistic (Z), and the one-sided and two-sided probability values. If you specify the BINOMIAL option in the EXACT statement, PROC FREQ also displays the exact one-sided and two-sided probability values for this test. If you specify the POINT option with the BINOMIAL option in the EXACT statement, PROC FREQ displays the exact point probability for the test.

Multiway Tables

PROC FREQ displays all multiway table requests in the TABLES statements, unless you specify the NOPRINT option in the PROC statement or the NOPRINT option in the TABLES statement.

For two-way to multiway crosstabulation tables, the values of the last variable in the table request form the table columns. The values of the next-to-last variable form the rows. Each level (or combination of levels) of the other variables forms one stratum.

There are three ways to display multiway tables in PROC FREQ. By default, PROC FREQ displays multiway tables as separate two-way crosstabulation tables for each stratum of the multiway table. Also by default, PROC FREQ displays these two-way crosstabulation tables in table cell format. Alternatively, if you specify the CROSSLIST option, PROC FREQ displays the two-way crosstabulation tables in ODS column format. If you specify the LIST option, PROC FREQ displays multiway tables in list format.

Crosstabulation Tables

By default, PROC FREQ displays two-way crosstabulation tables in table cell format. The row variable values are listed down the side of the table, the column variable values are listed across the top of the table, and each row and column variable level combination forms a table cell.

Each cell of a crosstabulation table may contain the following information:

- Frequency, giving the number of observations that have the indicated values of the two variables. (The NOFREQ option suppresses this information.)
- the Expected cell frequency under the hypothesis of independence, if you specify the EXPECTED option
- the Deviation of the cell frequency from the expected value, if you specify the DEVIATION option
- Cell Chi-Square, which is the cell's contribution to the total chi-square statistic, if you specify the CELLCHI2 option
- Tot Pct, or the cell's percentage of the total frequency, for *n*-way tables when $n > 2$, if you specify the TOTPCT option
- Percent, the cell's percentage of the total frequency. (The NOPERCENT option suppresses this information.)
- Row Pct, or the row percentage, the cell's percentage of the total frequency count for that cell's row. (The NOROW option suppresses this information.)
- Col Pct, or column percentage, the cell's percentage of the total frequency count for that cell's column. (The NOCOL option suppresses this information.)
- Cumulative Col%, or cumulative column percent, if you specify the CUMCOL option

The table also displays the Frequency Missing, or the number of observations with missing values.

CROSSLIST Tables

If you specify the CROSSLIST option, PROC FREQ displays two-way crosstabulation tables with ODS column format. Using column format, a CROSSLIST table provides the same information (frequencies, percentages, and other statistics) as the default crosstabulation table with cell format. But unlike the default crosstabulation table, a CROSSLIST table has a table definition that you can customize with PROC TEMPLATE. For more information, refer to the chapter titled "The TEMPLATE Procedure" in the *SAS Output Delivery System User's Guide*.

In the CROSSLIST table format, the rows of the display correspond to the crosstabulation table cells, and the columns of the display correspond to descriptive statistics such as frequencies and percentages. Each table cell is identified by the values of its TABLES row and column variable levels, with all column variable levels listed within each row variable level. The CROSSLIST table also provides row totals, column totals, and overall table totals.

For a crosstabulation table in the CROSSLIST format, PROC FREQ displays the following information:

- the row variable name and values
- the column variable name and values
- Frequency, giving the number of observations that have the indicated values of the two variables. (The NOFREQ option suppresses this information.)
- the Expected cell frequency under the hypothesis of independence, if you specify the EXPECTED option
- the Deviation of the cell frequency from the expected value, if you specify the DEVIATION option
- Cell Chi-Square, which is the cell's contribution to the total chi-square statistic, if you specify the CELLCHI2 option
- Total Percent, or the cell's percentage of the total frequency, for *n*-way tables when $n > 2$, if you specify the TOTPCT option
- Percent, the cell's percentage of the total frequency. (The NOPERCENT option suppresses this information.)
- Row Percent, the cell's percentage of the total frequency count for that cell's row. (The NOROW option suppresses this information.)
- Column Percent, the cell's percentage of the total frequency count for that cell's column. (The NOCOL option suppresses this information.)

The table also displays the Frequency Missing, or the number of observations with missing values.

LIST Tables

If you specify the LIST option in the TABLES statement, PROC FREQ displays multiway tables in a list format rather than as crosstabulation tables. The LIST option displays the entire multiway table in one table, instead of displaying a separate two-way table for each stratum. The LIST option is not available when you also request statistical options. Unlike the default crosstabulation output, the LIST output does not display row percentages, column percentages, and optional information such as expected frequencies and cell chi-squares.

For a multiway table in list format, PROC FREQ displays the following information:

- the variable names and values
- Frequency counts, giving the number of observations with the indicated combination of variable values
- Percent, the cell's percentage of the total number of observations. (The NOPERCENT option suppresses this information.)
- Cumulative Frequency counts, giving the sum of the frequency counts of that cell and all other cells listed above it in the table. The last cumulative frequency is the total number of nonmissing observations. (The NOCUM option suppresses this information.)

- Cumulative Percent values, giving the percentage of the total number of observations for that cell and all others previously listed in the table. (The NOCUM or the NOPERCENT option suppresses this information.)

The table also displays the Frequency Missing, or the number of observations with missing values.

Statistics for Multiway Tables

PROC FREQ computes statistical tests and measures for crosstabulation tables, depending on which statements and options you specify. You can suppress the display of all these results by specifying the NOPRINT option in the PROC statement. With any of the following information, PROC FREQ also displays the Sample Size and the Frequency Missing.

- If you specify the SCOROUT option, PROC FREQ displays the Row Scores and Column Scores that it uses for statistical computations. The Row Scores table displays the row variable values and the Score corresponding to each value. The Column Scores table displays the column variable values and the corresponding Scores. PROC FREQ also identifies the score type used to compute the row and column scores. You can specify the score type with the SCORES= option in the TABLES statement.
- If you specify the CHISQ option, PROC FREQ displays the following statistics for each two-way table: Pearson Chi-Square, Likelihood-Ratio Chi-Square, Continuity-Adjusted Chi-Square (for 2×2 tables), Mantel-Haenszel Chi-Square, the Phi Coefficient, the Contingency Coefficient, and Cramer's V. For each test statistic, PROC FREQ also displays the degrees of freedom (DF) and the probability value (Prob).
- If you specify the CHISQ option for 2×2 tables, PROC FREQ also displays Fisher's exact test. The test output includes the cell (1,1) frequency (F), the exact left-sided and right-sided probability values, the table probability (P), and the exact two-sided probability value.
- If you specify the FISHER option in the TABLES statement (or, equivalently, the FISHER option in the EXACT statement), PROC FREQ displays Fisher's exact test for tables larger than 2×2. The test output includes the table probability (P) and the probability value. In addition, PROC FREQ displays the CHISQ output listed earlier, even if you do not also specify the CHISQ option.
- If you specify the PCHI, LRCHI, or MHCHI option in the EXACT statement, PROC FREQ also displays the corresponding exact test: Pearson Chi-Square, Likelihood-Ratio Chi-Square, or Mantel-Haenszel Chi-Square, respectively. The test output includes the test statistic, the degrees of freedom (DF), and the asymptotic and exact probability values. If you also specify the POINT option in the EXACT statement, PROC FREQ displays the point probability for each exact test requested. If you specify the CHISQ option in the EXACT statement, PROC FREQ displays exact probability values for all three of these chi-square tests.

- If you specify the MEASURES option, PROC FREQ displays the following statistics and their asymptotic standard errors (ASE) for each two-way table: Gamma, Kendall's Tau-*b*, Stuart's Tau-*c*, Somers' $D(C|R)$, Somers' $D(R|C)$, Pearson Correlation, Spearman Correlation, Lambda Asymmetric $(C|R)$, Lambda Asymmetric $(R|C)$, Lambda Symmetric, Uncertainty Coefficient $(C|R)$, Uncertainty Coefficient $(R|C)$, and Uncertainty Coefficient Symmetric. If you specify the CL option, PROC FREQ also displays confidence limits for these measures.
- If you specify the PLCORR option, PROC FREQ displays the tetrachoric correlation for 2×2 tables or the polychoric correlation for larger tables. In addition, PROC FREQ displays the MEASURES output listed earlier, even if you do not also specify the MEASURES option.
- If you specify the option GAMMA, KENTB, STUTC, SMDCR, SMDRC, PCORR, or SCORR in the TEST statement, PROC FREQ displays asymptotic tests for Gamma, Kendall's Tau-*b*, Stuart's Tau-*c*, Somers' $D(C|R)$, Somers' $D(R|C)$, the Pearson Correlation, or the Spearman Correlation, respectively. If you specify the MEASURES option in the TEST statement, PROC FREQ displays all these asymptotic tests. The test output includes the statistic, its asymptotic standard error (ASE), Confidence Limits, the ASE under the null hypothesis H0, the standardized test statistic (Z), and the one-sided and two-sided probability values.
- If you specify the PCORR or SCORR option in the EXACT statement, PROC FREQ displays asymptotic and exact tests for the Pearson Correlation or the Spearman Correlation, respectively. The test output includes the correlation, its asymptotic standard error (ASE), Confidence Limits, the ASE under the null hypothesis H0, the standardized test statistic (Z), and the asymptotic and exact one-sided and two-sided probability values. If you also specify the POINT option in the EXACT statement, PROC FREQ displays the point probability for each exact test requested.
- If you specify the RISKDIFF option for 2×2 tables, PROC FREQ displays the Column 1 and Column 2 Risk Estimates. For each column, PROC FREQ displays Row 1 Risk, Row 2 Risk, Total Risk, and Risk Difference, together with their asymptotic standard errors (ASE), Asymptotic Confidence Limits, and Exact Confidence Limits. Exact confidence limits are not available for the risk difference.
- If you specify the MEASURES option or the RELRISK option for 2×2 tables, PROC FREQ displays Estimates of the Relative Risk for Case-Control and Cohort studies, together with their Confidence Limits. These measures are also known as the Odds Ratio and the Column 1 and 2 Relative Risks. If you specify the OR option in the EXACT statement, PROC FREQ also displays Exact Confidence Limits for the Odds Ratio.
- If you specify the TREND option, PROC FREQ displays the Cochran-Armitage Trend Test for tables that are $2 \times C$ or $R \times 2$. For this test, PROC FREQ gives the Statistic (Z) and the one-sided and two-sided probability values. If you specify the TREND option in the EXACT statement, PROC FREQ also displays the exact one-sided and two-sided probability values for this test.

If you specify the POINT option with the TREND option in the EXACT statement, PROC FREQ displays the exact point probability for the test statistic.

- If you specify the JT option, PROC FREQ displays the Jonckheere-Terpstra Test, showing the Statistic (JT), the standardized test statistic (Z), and the one-sided and two-sided probability values. If you specify the JT option in the EXACT statement, PROC FREQ also displays the exact one-sided and two-sided probability values for this test. If you specify the POINT option with the JT option in the EXACT statement, PROC FREQ displays the exact point probability for the test statistic.
- If you specify the AGREE option and the PRINTKWT option, PROC FREQ displays the Kappa Coefficient Weights for square tables greater than 2×2.
- If you specify the AGREE option, for two-way tables PROC FREQ displays McNemar's Test and the Simple Kappa Coefficient for 2×2 tables. For square tables larger than 2×2, PROC FREQ displays Bowker's Test of Symmetry, the Simple Kappa Coefficient, and the Weighted Kappa Coefficient. For McNemar's Test and Bowker's Test of Symmetry, PROC FREQ displays the Statistic (S), the degrees of freedom (DF), and the probability value (Pr > S). If you specify the MCNEM option in the EXACT statement, PROC FREQ also displays the exact probability value for McNemar's test. If you specify the POINT option with the MCNEM option in the EXACT statement, PROC FREQ displays the exact point probability for the test statistic. For the simple and weighted kappa coefficients, PROC FREQ displays the kappa values, asymptotic standard errors (ASE), and Confidence Limits.
- If you specify the KAPPA or WTKAP option in the TEST statement, PROC FREQ displays asymptotic tests for the simple kappa coefficient or the weighted kappa coefficient, respectively. If you specify the AGREE option in the TEST statement, PROC FREQ displays both these asymptotic tests. The test output includes the kappa coefficient, its asymptotic standard error (ASE), Confidence Limits, the ASE under the null hypothesis H0, the standardized test statistic (Z), and the one-sided and two-sided probability values.
- If you specify the KAPPA or WTKAP option in the EXACT statement, PROC FREQ displays asymptotic and exact tests for the simple kappa coefficient or the weighted kappa coefficient, respectively. The test output includes the kappa coefficient, its asymptotic standard error (ASE), Confidence Limits, the ASE under the null hypothesis H0, the standardized test statistic (Z), and the asymptotic and exact one-sided and two-sided probability values. If you specify the POINT option in the EXACT statement, PROC FREQ displays the point probability for each exact test requested.
- If you specify the MC option in the EXACT statement, PROC FREQ displays Monte Carlo estimates for all exact *p*-values requested by *statistic-options* in the EXACT statement. The Monte Carlo output includes the *p*-value Estimate, its Confidence Limits, the Number of Samples used to compute the Monte Carlo estimate, and the Initial Seed for random number generation.
- If you specify the AGREE option, for multiple strata PROC FREQ displays Overall Simple and Weighted Kappa Coefficients, with their asymptotic standard errors (ASE) and Confidence Limits. PROC FREQ also displays Tests for

Equal Kappa Coefficients, giving the Chi-Squares, degrees of freedom (DF), and probability values (Pr > ChiSq) for the Simple Kappa and Weighted Kappa. For multiple strata of 2×2 tables, PROC FREQ displays Cochran's Q, giving the Statistic (Q), the degrees of freedom (DF), and the probability value (Pr > Q).

- If you specify the CMH option, PROC FREQ displays Cochran-Mantel-Haenszel Statistics for the following three alternative hypotheses: Nonzero Correlation, Row Mean Scores Differ (ANOVA Statistic), and General Association. For each of these statistics, PROC FREQ gives the degrees of freedom (DF) and the probability value (Prob). For 2×2 tables, PROC FREQ also displays Estimates of the Common Relative Risk for Case-Control and Cohort studies, together with their confidence limits. These include both Mantel-Haenszel and Logit stratum-adjusted estimates of the common Odds Ratio, Column 1 Relative Risk, and Column 2 Relative Risk. Also for 2×2 tables, PROC FREQ displays the Breslow-Day Test for Homogeneity of the Odds Ratios. For this test, PROC FREQ gives the Chi-Square, the degrees of freedom (DF), and the probability value (Pr > ChiSq).
- If you specify the CMH option in the TABLES statement and also specify the COMOR option in the EXACT statement, PROC FREQ displays exact confidence limits for the Common Odds Ratio for multiple strata of 2×2 tables. PROC FREQ also displays the Exact Test of H0: Common Odds Ratio = 1. The test output includes the Cell (1,1) Sum (S), Mean of S Under H0, One-sided Pr <= S, and Point Pr = S. PROC FREQ also provides exact two-sided probability values for the test, computed according to the following three methods: 2 * One-sided, Sum of probabilities <= Point probability, and Pr >= |S - Mean|.

ODS Table Names

PROC FREQ assigns a name to each table 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 on ODS, see Chapter 14, "Using the Output Delivery System." (*SAS/STAT User's Guide*)

Table 2.11 lists the ODS table names together with their descriptions and the options required to produce the tables. Note that the ALL option in the TABLES statement invokes the CHISQ, MEASURES, and CMH options.

Table 2.11. ODS Tables Produced in PROC FREQ

ODS Table Name	Description	Statement	Option
BinomialProp	Binomial proportion	TABLES	BINOMIAL (one-way tables)
BinomialPropTest	Binomial proportion test	TABLES	BINOMIAL (one-way tables)
BreslowDayTest	Breslow-Day test	TABLES	CMH ($h \times 2 \times 2$ tables)
CMH	Cochran-Mantel-Haenszel statistics	TABLES	CMH
ChiSq	Chi-square tests	TABLES	CHISQ
CochransQ	Cochran's Q	TABLES	AGREE ($h \times 2 \times 2$ tables)
ColScores	Column scores	TABLES	SCOROUT
CommonOddsRatioCL	Exact confidence limits for the common odds ratio	EXACT	COMOR
CommonOddsRatioTest	Common odds ratio exact test	EXACT	COMOR
CommonRelRisks	Common relative risks	TABLES	CMH ($h \times 2 \times 2$ tables)
CrossList	Column format crosstabulation table	TABLES	CROSSLIST (n-way table request, $n > 1$)
CrossTabFreqs	Crosstabulation table	TABLES	(n-way table request, $n > 1$)
EqualKappaTest	Test for equal simple kappas	TABLES	AGREE ($h \times 2 \times 2$ tables)
EqualKappaTests	Tests for equal kappas	TABLES	AGREE ($h{\times}r{\times}r$ tables, $r > 2$)
FishersExact	Fisher's exact test	EXACT or TABLES or TABLES	FISHER FISHER or EXACT CHISQ (2×2 tables)
FishersExactMC	Monte Carlo estimates for Fisher's exact test	EXACT	FISHER / MC
Gamma	Gamma	TEST	GAMMA
GammaTest	Gamma test	TEST	GAMMA
JTTest	Jonckheere-Terpstra test	TABLES	JT
JTTestMC	Monte Carlo estimates for the JT exact test	EXACT	JT / MC
KappaStatistics	Kappa statistics	TABLES	AGREE ($r \times r$ tables, $r > 2$, and no TEST or EXACT KAPPA)
KappaWeights	Kappa weights	TABLES	AGREE and PRINTKWT
List	List format multiway table	TABLES	LIST
LRChiSq	Likelihood-ratio chi-square exact test	EXACT	LRCHI
LRChiSqMC	Monte Carlo exact test for likelihood-ratio chi-square	EXACT	LRCHI / MC
McNemarsTest	McNemar's test	TABLES	AGREE (2×2 tables)
Measures	Measures of association	TABLES	MEASURES
MHChiSq	Mantel-Haenszel chi-square exact test	EXACT	MHCHI
MHChiSqMC	Monte Carlo exact test for Mantel-Haenszel chi-square	EXACT	MHCHI / MC
NLevels	Number of variable levels	PROC	NLEVELS
OddsRatioCL	Exact confidence limits for the odds ratio	EXACT	OR
OneWayChiSq	One-way chi-square test	TABLES	CHISQ (one-way tables)

Table 2.11. (continued)

ODS Table Name	Description	Statement	Option
OneWayChiSqMC	Monte Carlo exact test for one-way chi-square	EXACT	CHISQ / MC (one-way tables)
OneWayFreqs	One-way frequencies	PROC or TABLES	(with no TABLES stmt) (one-way table request)
OverallKappa	Overall simple kappa	TABLES	AGREE ($h \times 2 \times 2$ tables)
OverallKappas	Overall kappa coefficients	TABLES	AGREE ($h{\times}r{\times}r$ tables, $r > 2$)
PearsonChiSq	Pearson chi-square exact test	EXACT	PCHI
PearsonChiSqMC	Monte Carlo exact test for Pearson chi-square	EXACT	PCHI / MC
PearsonCorr	Pearson correlation	TEST or EXACT	PCORR PCORR
PearsonCorrMC	Monte Carlo exact test for Pearson correlation	EXACT	PCORR / MC
PearsonCorrTest	Pearson correlation test	TEST or EXACT	PCORR PCORR
RelativeRisks	Relative risk estimates	TABLES	RELRISK or MEASURES (2×2 tables)
RiskDiffCol1	Column 1 risk estimates	TABLES	RISKDIFF (2×2 tables)
RiskDiffCol2	Column 2 risk estimates	TABLES	RISKDIFF (2×2 tables)
RowScores	Row scores	TABLES	SCOROUT
SimpleKappa	Simple kappa coefficient	TEST or EXACT	KAPPA KAPPA
SimpleKappaMC	Monte Carlo exact test for simple kappa	EXACT	KAPPA / MC
SimpleKappaTest	Simple kappa test	TEST or EXACT	KAPPA KAPPA
SomersDCR	Somers' $D(C\|R)$	TEST	SMDCR
SomersDCRTest	Somers' $D(C\|R)$ test	TEST	SMDCR
SomersDRC	Somers' $D(R\|C)$	TEST	SMDRC
SomersDRCTest	Somers' $D(R\|C)$ test	TEST	SMDRC
SpearmanCorr	Spearman correlation	TEST or EXACT	SCORR SCORR
SpearmanCorrMC	Monte Carlo exact test for Spearman correlation	EXACT	SCORR / MC
SpearmanCorrTest	Spearman correlation test	TEST or EXACT	SCORR SCORR
SymmetryTest	Test of symmetry	TABLES	AGREE
TauB	Kendall's tau-b	TEST	KENTB
TauBTest	Kendall's tau-b test	TEST	KENTB
TauC	Stuart's tau-c	TEST	STUTC
TauCTest	Stuart's tau-c test	TEST	STUTC
TrendTest	Cochran-Armitage test for trend	TABLES	TREND

Table 2.11. (continued)

ODS Table Name	Description	Statement	Option
TrendTestMC	Monte Carlo exact test for trend	EXACT	TREND / MC
WeightedKappa	Weighted kappa	TEST or EXACT	WTKAP WTKAP
WeightedKappaMC	Monte Carlo exact test for weighted kappa	EXACT	WTKAP / MC
WeightedKappaTest	Weighted kappa test	TEST or EXACT	WTKAP WTKAP

Examples

Example 2.1. Creating an Output Data Set with Table Cell Frequencies

The eye and hair color of children from two different regions of Europe are recorded in the data set Color. Instead of recording one observation per child, the data are recorded as cell counts, where the variable Count contains the number of children exhibiting each of the 15 eye and hair color combinations. The data set does not include missing combinations.

```
data Color;
   input Region Eyes $ Hair $ Count @@;
      label Eyes  ='Eye Color'
            Hair  ='Hair Color'
            Region='Geographic Region';
      datalines;
1 blue  fair   23  1 blue  red     7  1 blue  medium 24
1 blue  dark   11  1 green fair   19  1 green red     7
1 green medium 18  1 green dark   14  1 brown fair   34
1 brown red     5  1 brown medium 41  1 brown dark   40
1 brown black   3  2 blue  fair   46  2 blue  red    21
2 blue  medium 44  2 blue  dark   40  2 blue  black   6
2 green fair   50  2 green red    31  2 green medium 37
2 green dark   23  2 brown fair   56  2 brown red    42
2 brown medium 53  2 brown dark   54  2 brown black  13
;
```

The following statements read the Color data set and create an output data set containing the frequencies, percentages, and expected cell frequencies of the Eyes by Hair two-way table. The TABLES statement requests three tables: Eyes and Hair frequency tables and an Eyes by Hair crosstabulation table. The OUT= option creates the FreqCnt data set, which contains the crosstabulation table frequencies. The OUTEXPECT option outputs the expected cell frequencies to FreqCnt, and the SPARSE option includes the zero cell counts. The WEIGHT statement specifies that Count contains the observation weights. These statements create Output 2.1.1 through Output 2.1.3.

```
proc freq data=Color;
   weight Count;
   tables Eyes Hair Eyes*Hair / out=FreqCnt outexpect sparse;
   title1 'Eye and Hair Color of European Children';
run;
proc print data=FreqCnt noobs;
   title2 'Output Data Set from PROC FREQ';
run;
```

Output 2.1.1 displays the two frequency tables produced, one showing the distribution of eye color, and one showing the distribution of hair color. By default, PROC FREQ lists the variables values in alphabetical order. The 'Eyes*Hair' specification produces a crosstabulation table, shown in Output 2.1.2, with eye color defining the table rows and hair color defining the table columns. A zero cell count for green eyes and black hair indicates that this eye and hair color combination does not occur in the data.

The output data set (Output 2.1.3) contains frequency counts and percentages for the last table. The data set also includes an observation for the zero cell count (SPARSE) and a variable with the expected cell frequency for each table cell (OUTEXPECT).

Output 2.1.1. Frequency Tables

```
                    Eye and Hair Color of European Children

                               The FREQ Procedure

                                   Eye Color

                                                 Cumulative    Cumulative
         Eyes     Frequency      Percent      Frequency       Percent
         ----------------------------------------------------------
         blue          222        29.13           222          29.13
         brown         341        44.75           563          73.88
         green         199        26.12           762         100.00


                                  Hair Color

                                                 Cumulative    Cumulative
        Hair      Frequency      Percent      Frequency       Percent
        -----------------------------------------------------------
        black           22         2.89            22           2.89
        dark           182        23.88           204          26.77
        fair           228        29.92           432          56.69
        medium         217        28.48           649          85.17
        red            113        14.83           762         100.00
```

Output 2.1.2. Crosstabulation Table

```
                    Eye and Hair Color of European Children

                             Table of Eyes by Hair

          Eyes(Eye Color)     Hair(Hair Color)

          Frequency|
          Percent  |
          Row Pct  |
          Col Pct  |black   |dark    |fair    |medium  |red     |  Total
          ---------+--------+--------+--------+--------+--------+
          blue     |      6 |     51 |     69 |     68 |     28 |    222
                   |   0.79 |   6.69 |   9.06 |   8.92 |   3.67 |  29.13
                   |   2.70 |  22.97 |  31.08 |  30.63 |  12.61 |
                   |  27.27 |  28.02 |  30.26 |  31.34 |  24.78 |
          ---------+--------+--------+--------+--------+--------+
          brown    |     16 |     94 |     90 |     94 |     47 |    341
                   |   2.10 |  12.34 |  11.81 |  12.34 |   6.17 |  44.75
                   |   4.69 |  27.57 |  26.39 |  27.57 |  13.78 |
                   |  72.73 |  51.65 |  39.47 |  43.32 |  41.59 |
          ---------+--------+--------+--------+--------+--------+
          green    |      0 |     37 |     69 |     55 |     38 |    199
                   |   0.00 |   4.86 |   9.06 |   7.22 |   4.99 |  26.12
                   |   0.00 |  18.59 |  34.67 |  27.64 |  19.10 |
                   |   0.00 |  20.33 |  30.26 |  25.35 |  33.63 |
          ---------+--------+--------+--------+--------+--------+
          Total          22      182      228      217      113      762
                       2.89    23.88    29.92    28.48    14.83   100.00
```

Output 2.1.3. OUT= Data Set

```
                         Output Data Set from PROC FREQ

                 Eyes     Hair      COUNT    EXPECTED    PERCENT

                 blue     black        6       6.409      0.7874
                 blue     dark        51      53.024      6.6929
                 blue     fair        69      66.425      9.0551
                 blue     medium      68      63.220      8.9239
                 blue     red         28      32.921      3.6745
                 brown    black       16       9.845      2.0997
                 brown    dark        94      81.446     12.3360
                 brown    fair        90     102.031     11.8110
                 brown    medium      94      97.109     12.3360
                 brown    red         47      50.568      6.1680
                 green    black        0       5.745      0.0000
                 green    dark        37      47.530      4.8556
                 green    fair        69      59.543      9.0551
                 green    medium      55      56.671      7.2178
                 green    red         38      29.510      4.9869
```

Example 2.2. Computing Chi-Square Tests for One-Way Frequency Tables

This example examines whether the children's hair color (from Example 2.1 on page 161) has a specified multinomial distribution for the two regions. The hypothesized distribution for hair color is 30% fair, 12% red, 30% medium, 25% dark, and 3% black.

In order to test the hypothesis for each region, the data are first sorted by Region. Then the FREQ procedure uses a BY statement to produce a separate table for each BY group (Region). The option ORDER=DATA orders the frequency table values (hair color) by their order in the data set. The TABLES statement requests a frequency table for hair color, and the option NOCUM suppresses the display of the cumulative frequencies and percentages. The TESTP= option specifies the hypothesized percentages for the chi-square test; the number of percentages specified equals the number of table levels, and the percentages sum to 100. The following statements produce Output 2.2.1.

```
proc sort data=Color;
   by Region;
run;
proc freq data=Color order=data;
   weight Count;
   tables Hair / nocum testp=(30 12 30 25 3);
   by Region;
   title 'Hair Color of European Children';
run;
```

The frequency tables in Output 2.2.1 list the variable values (hair color) in the order in which they appear in the data set. The "Test Percent" column lists the hypothesized percentages for the chi-square test. Always check that you have ordered the TESTP= percentages to correctly match the order of the variable levels.

PROC FREQ computes a chi-square statistic for each region. The chi-square statistic is significant at the 0.05 level for Region 2 (p=0.0003) but not for Region 1. This indicates a significant departure from the hypothesized percentages in Region 2.

Output 2.2.1. One-Way Frequency Table with BY Groups

```
                   Hair Color of European Children

----------------------------- Geographic Region=1 ------------------------------

                          The FREQ Procedure

                             Hair Color

                                                Test
                  Hair      Frequency    Percent     Percent
                  -------------------------------------------
                  fair            76       30.89       30.00
                  red             19        7.72       12.00
                  medium          83       33.74       30.00
                  dark            65       26.42       25.00
                  black            3        1.22        3.00


                           Chi-Square Test
                       for Specified Proportions
                       -------------------------
                       Chi-Square         7.7602
                       DF                      4
                       Pr > ChiSq         0.1008

                           Sample Size = 246
```

```
                   Hair Color of European Children

----------------------------- Geographic Region=2 ------------------------------

                             Hair Color

                                                Test
                  Hair      Frequency    Percent     Percent
                  -------------------------------------------
                  fair           152       29.46       30.00
                  red             94       18.22       12.00
                  medium         134       25.97       30.00
                  dark           117       22.67       25.00
                  black           19        3.68        3.00


                           Chi-Square Test
                       for Specified Proportions
                       -------------------------
                       Chi-Square        21.3824
                       DF                      4
                       Pr > ChiSq         0.0003

                           Sample Size = 516
```

Example 2.3. Computing Binomial Proportions for One-Way Frequency Tables

The binomial proportion is computed as the proportion of observations for the first level of the variable that you are studying. The following statements compute the proportion of children with brown eyes (from the data set in Example 2.1 on page 161) and test this value against the hypothesis that the proportion is 50%. Also, these statements test whether the proportion of children with fair hair is 28%.

```
proc freq data=Color order=freq;
   weight Count;
   tables Eyes / binomial alpha=.1;
   tables Hair / binomial(p=.28);
   title 'Hair and Eye Color of European Children';
run;
```

The first TABLES statement produces a frequency table for eye color. The BINOMIAL option computes the binomial proportion and confidence limits, and it tests the hypothesis that the proportion for the first eye color level (brown) is 0.5. The option ALPHA=.1 specifies that 90% confidence limits should be computed. The second TABLES statement creates a frequency table for hair color and computes the binomial proportion and confidence limits, but it tests that the proportion for the first hair color (fair) is 0.28. These statements produce Output 2.3.1 and Output 2.3.2.

The frequency table in Output 2.3.1 displays the variable values in order of descending frequency count. Since the first variable level is 'brown', PROC FREQ computes the binomial proportion of children with brown eyes. PROC FREQ also computes its asymptotic standard error (ASE), and asymptotic and exact 90% confidence limits. If you do not specify the ALPHA= option, then PROC FREQ computes the default 95% confidence limits.

Because the value of Z is less than zero, PROC FREQ computes a left-sided p-value (0.0019). This small p-value supports the alternative hypothesis that the true value of the proportion of children with brown eyes is less than 50%.

Output 2.3.2 displays the results from the second TABLES statement. PROC FREQ computes the default 95% confidence limits since the ALPHA= option is not specified. The value of Z is greater than zero, so PROC FREQ computes a right-sided p-value (0.1188). This large p-value provides insufficient evidence to reject the null hypothesis that the proportion of children with fair hair is 28%.

Output 2.3.1. Binomial Proportion for Eye Color

```
                 Hair and Eye Color of European Children

                           The FREQ Procedure

                               Eye Color

                                              Cumulative    Cumulative
    Eyes     Frequency     Percent     Frequency      Percent
    ----------------------------------------------------------
    brown         341       44.75           341        44.75
    blue          222       29.13           563        73.88
    green         199       26.12           762       100.00


                          Binomial Proportion
                           for Eyes = brown
                 --------------------------------
                 Proportion                0.4475
                 ASE                       0.0180
                 90% Lower Conf Limit      0.4179
                 90% Upper Conf Limit      0.4771

                 Exact Conf Limits
                 90% Lower Conf Limit      0.4174
                 90% Upper Conf Limit      0.4779

                   Test of H0: Proportion = 0.5

                 ASE under H0              0.0181
                 Z                        -2.8981
                 One-sided Pr <  Z         0.0019
                 Two-sided Pr > |Z|        0.0038

                        Sample Size = 762
```

Output 2.3.2. Binomial Proportion for Hair Color

```
                 Hair and Eye Color of European Children

                               Hair Color

                                              Cumulative    Cumulative
           Hair      Frequency     Percent     Frequency      Percent
           ---------------------------------------------------------
           fair           228       29.92           228        29.92
           medium         217       28.48           445        58.40
           dark           182       23.88           627        82.28
           red            113       14.83           740        97.11
           black           22        2.89           762       100.00


                           Binomial Proportion
                             for Hair = fair
                         --------------------------------
                         Proportion                0.2992
                         ASE                       0.0166
                         95% Lower Conf Limit      0.2667
                         95% Upper Conf Limit      0.3317

                         Exact Conf Limits
                         95% Lower Conf Limit      0.2669
                         95% Upper Conf Limit      0.3331

                          Test of H0: Proportion = 0.28

                         ASE under H0              0.0163
                         Z                         1.1812
                         One-sided Pr >  Z         0.1188
                         Two-sided Pr > |Z|        0.2375

                                Sample Size = 762
```

Example 2.4. Analyzing a 2x2 Contingency Table

This example computes chi-square tests and Fisher's exact test to compare the probability of coronary heart disease for two types of diet. It also estimates the relative risks and computes exact confidence limits for the odds ratio.

The data set FatComp contains hypothetical data for a case-control study of high fat diet and the risk of coronary heart disease. The data are recorded as cell counts, where the variable Count contains the frequencies for each exposure and response combination. The data set is sorted in descending order by the variables Exposure and Response, so that the first cell of the 2×2 table contains the frequency of positive exposure and positive response. The FORMAT procedure creates formats to identify the type of exposure and response with character values.

```
proc format;
   value ExpFmt 1='High Cholesterol Diet'
                0='Low Cholesterol Diet';
   value RspFmt 1='Yes'
                0='No';
run;

data FatComp;
   input Exposure Response Count;
   label Response='Heart Disease';
   datalines;
0 0  6
0 1  2
1 0  4
1 1 11
;

proc sort data=FatComp;
   by descending Exposure descending Response;
run;
```

In the following statements, the TABLES statement creates a two-way table, and the option ORDER=DATA orders the contingency table values by their order in the data set. The CHISQ option produces several chi-square tests, while the RELRISK option produces relative risk measures. The EXACT statement creates the exact Pearson chi-square test and exact confidence limits for the odds ratio. These statements produce Output 2.4.1 through Output 2.4.3.

```
proc freq data=FatComp order=data;
   weight Count;
   tables Exposure*Response / chisq relrisk;
   exact pchi or;
   format Exposure ExpFmt. Response RspFmt.;
   title 'Case-Control Study of High Fat/Cholesterol Diet';
run;
```

Output 2.4.1. Contingency Table

```
                 Case-Control Study of High Fat/Cholesterol Diet

                              The FREQ Procedure

                        Table of Exposure by Response

                   Exposure          Response(Heart Disease)

                   Frequency        |
                   Percent          |
                   Row Pct          |
                   Col Pct          |Yes     |No      |  Total
                   -----------------+--------+--------+
                   High Cholesterol |     11 |      4 |     15
                    Diet            |  47.83 |  17.39 |  65.22
                                    |  73.33 |  26.67 |
                                    |  84.62 |  40.00 |
                   -----------------+--------+--------+
                   Low Cholesterol  |      2 |      6 |      8
                   Diet             |   8.70 |  26.09 |  34.78
                                    |  25.00 |  75.00 |
                                    |  15.38 |  60.00 |
                   -----------------+--------+--------+
                   Total                  13       10       23
                                       56.52    43.48   100.00
```

The contingency table in Output 2.4.1 displays the variable values so that the first table cell contains the frequency for the first cell in the data set, the frequency of positive exposure and positive response.

Output 2.4.2. Chi-Square Statistics

```
                 Case-Control Study of High Fat/Cholesterol Diet

                   Statistics for Table of Exposure by Response

              Statistic                     DF       Value      Prob
              ------------------------------------------------------
              Chi-Square                     1      4.9597    0.0259
              Likelihood Ratio Chi-Square    1      5.0975    0.0240
              Continuity Adj. Chi-Square     1      3.1879    0.0742
              Mantel-Haenszel Chi-Square     1      4.7441    0.0294
              Phi Coefficient                       0.4644
              Contingency Coefficient               0.4212
              Cramer's V                            0.4644

            WARNING: 50% of the cells have expected counts less than 5.
                     (Asymptotic) Chi-Square may not be a valid test.


                             Pearson Chi-Square Test
                        ----------------------------------
                        Chi-Square                  4.9597
                        DF                               1
                        Asymptotic Pr >  ChiSq      0.0259
                        Exact      Pr >= ChiSq      0.0393


                               Fisher's Exact Test
                        ----------------------------------
                        Cell (1,1) Frequency (F)        11
                        Left-sided Pr <= F          0.9967
                        Right-sided Pr >= F         0.0367

                        Table Probability (P)       0.0334
                        Two-sided Pr <= P           0.0393

                                 Sample Size = 23
```

Output 2.4.2 displays the chi-square statistics. Since the expected counts in some of the table cells are small, PROC FREQ gives a warning that the asymptotic chi-square tests may not be appropriate. In this case, the exact tests are appropriate. The alternative hypothesis for this analysis states that coronary heart disease is more likely to be associated with a high fat diet, so a one-sided test is desired. Fisher's exact right-sided test analyzes whether the probability of heart disease in the high fat group exceeds the probability of heart disease in the low fat group; since this p-value is small, the alternative hypothesis is supported.

Output 2.4.3. Relative Risk

```
             Case-Control Study of High Fat/Cholesterol Diet

               Statistics for Table of Exposure by Response

               Estimates of the Relative Risk (Row1/Row2)

 Type of Study                 Value       95% Confidence Limits
 -----------------------------------------------------------------
 Case-Control (Odds Ratio)    8.2500        1.1535       59.0029
 Cohort (Col1 Risk)           2.9333        0.8502       10.1204
 Cohort (Col2 Risk)           0.3556        0.1403        0.9009


                     Odds Ratio (Case-Control Study)
                   -----------------------------------
                   Odds Ratio                   8.2500

                   Asymptotic Conf Limits
                   95% Lower Conf Limit         1.1535
                   95% Upper Conf Limit        59.0029

                   Exact Conf Limits
                   95% Lower Conf Limit         0.8677
                   95% Upper Conf Limit       105.5488

                             Sample Size = 23
```

The odds ratio, displayed in Output 2.4.3, provides an estimate of the relative risk when an event is rare. This estimate indicates that the odds of heart disease is 8.25 times higher in the high fat diet group; however, the wide confidence limits indicate that this estimate has low precision.

Example 2.5. Creating an Output Data Set Containing Chi-Square Statistics

This example uses the Color data from Example 2.1 (page 161) to output the Pearson chi-square and the likelihood-ratio chi-square statistics to a SAS data set. The following statements create a two-way table of eye color versus hair color.

```
proc freq data=Color order=data;
   weight Count;
   tables Eyes*Hair / chisq expected cellchi2 norow nocol;
   output out=ChiSqData pchi lrchi n nmiss;
   title 'Chi-Square Tests for 3 by 5 Table of Eye and Hair Color';
run;
proc print data=ChiSqData noobs;
   title1 'Chi-Square Statistics for Eye and Hair Color';
   title2 'Output Data Set from the FREQ Procedure';
run;
```

The CHISQ option produces chi-square tests, the EXPECTED option displays expected cell frequencies in the table, and the CELLCHI2 option displays the cell contribution to the chi-square. The NOROW and NOCOL options suppress the display of row and column percents in the table.

The OUTPUT statement creates the ChiSqData data set with eight variables: the N option stores the number of nonmissing observations, the NMISS option stores the number of missing observations, and the PCHI and LRCHI options store Pearson and likelihood-ratio chi-square statistics, respectively, together with their degrees of freedom and p-values.

The preceding statements produce Output 2.5.1 and Output 2.5.2.

Output 2.5.1. Contingency Table

```
             Chi-Square Tests for 3 by 5 Table of Eye and Hair Color

                               The FREQ Procedure

                              Table of Eyes by Hair

      Eyes(Eye Color)     Hair(Hair Color)

      Frequency      |
      Expected       |
      Cell Chi-Square|
      Percent        |fair    |red     |medium  |dark    |black   |  Total
      ---------------+--------+--------+--------+--------+--------+
      blue           |     69 |     28 |     68 |     51 |      6 |    222
                     | 66.425 | 32.921 |  63.22 | 53.024 | 6.4094 |
                     | 0.0998 | 0.7357 | 0.3613 | 0.0772 | 0.0262 |
                     |   9.06 |   3.67 |   8.92 |   6.69 |   0.79 |  29.13
      ---------------+--------+--------+--------+--------+--------+
      green          |     69 |     38 |     55 |     37 |      0 |    199
                     | 59.543 |  29.51 | 56.671 |  47.53 | 5.7454 |
                     | 1.5019 | 2.4422 | 0.0492 | 2.3329 | 5.7454 |
                     |   9.06 |   4.99 |   7.22 |   4.86 |   0.00 |  26.12
      ---------------+--------+--------+--------+--------+--------+
      brown          |     90 |     47 |     94 |     94 |     16 |    341
                     | 102.03 | 50.568 | 97.109 | 81.446 | 9.8451 |
                     | 1.4187 | 0.2518 | 0.0995 |  1.935 | 3.8478 |
                     |  11.81 |   6.17 |  12.34 |  12.34 |   2.10 |  44.75
      ---------------+--------+--------+--------+--------+--------+
      Total               228      113      217      182       22      762
                        29.92    14.83    28.48    23.88     2.89   100.00
```

The contingency table in Output 2.5.1 displays eye and hair color in the order in which they appear in the Color data set. The Pearson chi-square statistic in Output 2.5.2 provides evidence of an association between eye and hair color (p=0.0073). The cell chi-square values show that most of the association is due to more green-eyed children with fair or red hair and fewer with dark or black hair. The opposite occurs with the brown-eyed children.

Output 2.5.2. Chi-Square Statistics

```
          Chi-Square Tests for 3 by 5 Table of Eye and Hair Color

                    Statistics for Table of Eyes by Hair

           Statistic                      DF       Value      Prob
           ------------------------------------------------------
           Chi-Square                      8     20.9248    0.0073
           Likelihood Ratio Chi-Square     8     25.9733    0.0011
           Mantel-Haenszel Chi-Square      1      3.7838    0.0518
           Phi Coefficient                        0.1657
           Contingency Coefficient                0.1635
           Cramer's V                             0.1172

                             Sample Size = 762
```

Output 2.5.3. Output Data Set

```
                Chi-Square Statistics for Eye and Hair Color
                  Output Data Set from the FREQ Procedure

 N    NMISS    _PCHI_   DF_PCHI      P_PCHI   _LRCHI_  DF_LRCHI     P_LRCHI

762     0     20.9248      8     .007349898  25.9733       8     .001061424
```

The OUT= data set is displayed in Output 2.5.3. It contains one observation with the sample size, the number of missing values, and the chi-square statistics and corresponding degrees of freedom and p-values as in Output 2.5.2.

Example 2.6. Computing Cochran-Mantel-Haenszel Statistics for a Stratified Table

The data set Migraine contains hypothetical data for a clinical trial of migraine treatment. Subjects of both genders receive either a new drug therapy or a placebo. Their response to treatment is coded as 'Better' or 'Same'. The data are recorded as cell counts, and the number of subjects for each treatment and response combination is recorded in the variable Count.

```
data Migraine;
   input Gender $ Treatment $ Response $ Count @@;
   datalines;
female Active  Better 16   female Active  Same 11
female Placebo Better  5   female Placebo Same 20
male   Active  Better 12   male   Active  Same 16
male   Placebo Better  7   male   Placebo Same 19
;
```

The following statements create a three-way table stratified by Gender, where Treatment forms the rows and Response forms the columns. The CMH option produces the Cochran-Mantel-Haenszel statistics. For this stratified 2×2 table, estimates of the common relative risk and the Breslow-Day test for homogeneity of the odds ratios are also displayed. The NOPRINT option suppresses the display of the contingency tables. These statements produce Output 2.6.1 through Output 2.6.3.

```
proc freq data=Migraine;
   weight Count;
   tables Gender*Treatment*Response / cmh noprint;
   title 'Clinical Trial for Treatment of Migraine Headaches';
run;
```

Output 2.6.1. Cochran-Mantel-Haenszel Statistics

```
                 Clinical Trial for Treatment of Migraine Headaches

                                The FREQ Procedure

                    Summary Statistics for Treatment by Response
                               Controlling for Gender

             Cochran-Mantel-Haenszel Statistics (Based on Table Scores)

           Statistic    Alternative Hypothesis    DF       Value      Prob
           ---------------------------------------------------------------
               1        Nonzero Correlation        1      8.3052    0.0040
               2        Row Mean Scores Differ     1      8.3052    0.0040
               3        General Association        1      8.3052    0.0040

                              Total Sample Size = 106
```

For a stratified 2×2 table, the three CMH statistics displayed in Output 2.6.1 test the same hypothesis. The significant p-value (0.004) indicates that the association between treatment and response remains strong after adjusting for gender.

Output 2.6.2. CMH Option: Relative Risks

```
               Clinical Trial for Treatment of Migraine Headaches

                  Summary Statistics for Treatment by Response
                             Controlling for Gender

               Estimates of the Common Relative Risk (Row1/Row2)

 Type of Study     Method                  Value     95% Confidence Limits
 -------------------------------------------------------------------------
 Case-Control      Mantel-Haenszel        3.3132       1.4456       7.5934
   (Odds Ratio)    Logit                  3.2941       1.4182       7.6515

 Cohort            Mantel-Haenszel        2.1636       1.2336       3.7948
   (Col1 Risk)     Logit                  2.1059       1.1951       3.7108

 Cohort            Mantel-Haenszel        0.6420       0.4705       0.8761
   (Col2 Risk)     Logit                  0.6613       0.4852       0.9013

                            Total Sample Size = 106
```

The CMH option also produces a table of relative risks, as shown in Output 2.6.2. Because this is a prospective study, the relative risk estimate assesses the effectiveness of the new drug; the "Cohort (Col1 Risk)" values are the appropriate estimates for the first column, or the risk of improvement. The probability of migraine improvement with the new drug is just over two times the probability of improvement with the placebo.

Output 2.6.3. CMH Option: Breslow-Day Test

```
               Clinical Trial for Treatment of Migraine Headaches

                  Summary Statistics for Treatment by Response
                             Controlling for Gender

                              Breslow-Day Test for
                          Homogeneity of the Odds Ratios
                          ------------------------------
                          Chi-Square              1.4929
                          DF                           1
                          Pr > ChiSq              0.2218

                            Total Sample Size = 106
```

The large p-value for the Breslow-Day test (0.2218) in Output 2.6.3 indicates no significant gender difference in the odds ratios.

Example 2.7. Computing the Cochran-Armitage Trend Test

The data set Pain contains hypothetical data for a clinical trial of a drug therapy to control pain. The clinical trial investigates whether adverse responses increase with larger drug doses. Subjects receive either a placebo or one of four drug doses. An adverse response is recorded as Adverse='Yes'; otherwise, it is recorded as Adverse='No'. The number of subjects for each drug dose and response combination is contained in the variable Count.

```
data pain;
   input Dose Adverse $ Count @@;
   datalines;
0 No 26   0 Yes  6
1 No 26   1 Yes  7
2 No 23   2 Yes  9
3 No 18   3 Yes 14
4 No  9   4 Yes 23
;
```

The TABLES statement in the following program produces a two-way table. The MEASURES option produces measures of association, and the CL option produces confidence limits for these measures. The TREND option tests for a trend across the ordinal values of the Dose variable with the Cochran-Armitage test. The EXACT statement produces exact p-values for this test, and the MAXTIME= option terminates the exact computations if they do not complete within 60 seconds. The TEST statement computes an asymptotic test for Somer's $D(C|R)$. These statements produce Output 2.7.1 through Output 2.7.3.

```
proc freq data=Pain;
   weight Count;
   tables Dose*Adverse / trend measures cl;
   test smdcr;
   exact trend / maxtime=60;
   title1 'Clinical Trial for Treatment of Pain';
run;
```

Output 2.7.1. Contingency Table

```
                  Clinical Trial for Treatment of Pain

                           The FREQ Procedure

                       Table of Dose by Adverse

                  Dose      Adverse

                  Frequency|
                  Percent  |
                  Row Pct  |
                  Col Pct  |No      |Yes     |  Total
                  ---------+--------+--------+
                         0 |     26 |      6 |     32
                           |  16.15 |   3.73 |  19.88
                           |  81.25 |  18.75 |
                           |  25.49 |  10.17 |
                  ---------+--------+--------+
                         1 |     26 |      7 |     33
                           |  16.15 |   4.35 |  20.50
                           |  78.79 |  21.21 |
                           |  25.49 |  11.86 |
                  ---------+--------+--------+
                         2 |     23 |      9 |     32
                           |  14.29 |   5.59 |  19.88
                           |  71.88 |  28.13 |
                           |  22.55 |  15.25 |
                  ---------+--------+--------+
                         3 |     18 |     14 |     32
                           |  11.18 |   8.70 |  19.88
                           |  56.25 |  43.75 |
                           |  17.65 |  23.73 |
                  ---------+--------+--------+
                         4 |      9 |     23 |     32
                           |   5.59 |  14.29 |  19.88
                           |  28.13 |  71.88 |
                           |   8.82 |  38.98 |
                  ---------+--------+--------+
                  Total         102       59      161
                              63.35    36.65   100.00
```

The "Row Pct" values in Output 2.7.1 show the expected increasing trend in the proportion of adverse effects due to increasing dosage (from 18.75% to 71.88%).

Output 2.7.2. Measures of Association

```
                    Clinical Trial for Treatment of Pain

                  Statistics for Table of Dose by Adverse

                                                                  95%
Statistic                              Value       ASE     Confidence Limits
-----------------------------------------------------------------------------
Gamma                                 0.5313    0.0935     0.3480     0.7146
Kendall's Tau-b                       0.3373    0.0642     0.2114     0.4631
Stuart's Tau-c                        0.4111    0.0798     0.2547     0.5675

Somers' D C|R                         0.2569    0.0499     0.1592     0.3547
Somers' D R|C                         0.4427    0.0837     0.2786     0.6068

Pearson Correlation                   0.3776    0.0714     0.2378     0.5175
Spearman Correlation                  0.3771    0.0718     0.2363     0.5178

Lambda Asymmetric C|R                 0.2373    0.0837     0.0732     0.4014
Lambda Asymmetric R|C                 0.1250    0.0662     0.0000     0.2547
Lambda Symmetric                      0.1604    0.0621     0.0388     0.2821

Uncertainty Coefficient C|R           0.1261    0.0467     0.0346     0.2175
Uncertainty Coefficient R|C           0.0515    0.0191     0.0140     0.0890
Uncertainty Coefficient Symmetric     0.0731    0.0271     0.0199     0.1262


                                  Somers' D C|R
                      --------------------------------
                      Somers' D C|R              0.2569
                      ASE                        0.0499
                      95% Lower Conf Limit       0.1592
                      95% Upper Conf Limit       0.3547

                       Test of H0: Somers' D C|R = 0

                      ASE under H0               0.0499
                      Z                          5.1511
                      One-sided Pr >  Z          <.0001
                      Two-sided Pr > |Z|         <.0001

                             Sample Size = 161
```

Output 2.7.2 displays the measures of association produced by the MEASURES option. Somer's $D(C|R)$ measures the association treating the column variable (Adverse) as the response and the row variable (Dose) as a predictor. Because the asymptotic 95% confidence limits do not contain zero, this indicates a strong positive association. Similarly, the Pearson and Spearman correlation coefficients show evidence of a strong positive association, as hypothesized.

Output 2.7.3. Trend Test

```
                 Clinical Trial for Treatment of Pain

               Statistics for Table of Dose by Adverse

                     Cochran-Armitage Trend Test
                    ------------------------------
                    Statistic (Z)           -4.7918

                    Asymptotic Test
                    One-sided Pr <  Z        <.0001
                    Two-sided Pr > |Z|       <.0001

                    Exact Test
                    One-sided Pr <=  Z    7.237E-07
                    Two-sided Pr >= |Z|   1.324E-06

                           Sample Size = 161
```

The Cochran-Armitage test (Output 2.7.3) supports the trend hypothesis. The small left-sided p-values for the Cochran-Armitage test indicate that the probability of the Column 1 level (Adverse='No') decreases as Dose increases or, equivalently, that the probability of the Column 2 level (Adverse='Yes') increases as Dose increases. The two-sided p-value tests against either an increasing or decreasing alternative. This is an appropriate hypothesis when you want to determine whether the drug has progressive effects on the probability of adverse effects but the direction is unknown.

Example 2.8. Computing Friedman's Chi-Square Statistic

Friedman's test is a nonparametric test for treatment differences in a randomized complete block design. Each block of the design may be a subject or a homogeneous group of subjects. If blocks are groups of subjects, the number of subjects in each block must equal the number of treatments. Treatments are randomly assigned to subjects within each block. If there is one subject per block, then the subjects are repeatedly measured once under each treatment. The order of treatments is randomized for each subject.

In this setting, Friedman's test is identical to the ANOVA (row means scores) CMH statistic when the analysis uses rank scores (SCORES=RANK). The three-way table uses subject (or subject group) as the stratifying variable, treatment as the row variable, and response as the column variable. PROC FREQ handles ties by assigning midranks to tied response values. If there are multiple subjects per treatment in each block, the ANOVA CMH statistic is a generalization of Friedman's test.

The data set Hypnosis contains data from a study investigating whether hypnosis has the same effect on skin potential (measured in millivolts) for four emotions (Lehmann 1975, p. 264). Eight subjects are asked to display fear, joy, sadness, and calmness under hypnosis. The data are recorded as one observation per subject for each emotion.

```
data Hypnosis;
   length Emotion $ 10;
   input Subject Emotion $ SkinResponse @@;
   datalines;
1 fear 23.1  1 joy 22.7  1 sadness 22.5  1 calmness 22.6
2 fear 57.6  2 joy 53.2  2 sadness 53.7  2 calmness 53.1
3 fear 10.5  3 joy  9.7  3 sadness 10.8  3 calmness  8.3
4 fear 23.6  4 joy 19.6  4 sadness 21.1  4 calmness 21.6
5 fear 11.9  5 joy 13.8  5 sadness 13.7  5 calmness 13.3
6 fear 54.6  6 joy 47.1  6 sadness 39.2  6 calmness 37.0
7 fear 21.0  7 joy 13.6  7 sadness 13.7  7 calmness 14.8
8 fear 20.3  8 joy 23.6  8 sadness 16.3  8 calmness 14.8
;
```

In the following statements, the TABLES statement creates a three-way table stratified by Subject and a two-way table; the variables Emotion and SkinResponse form the rows and columns of each table. The CMH2 option produces the first two Cochran-Mantel-Haenszel statistics, the option SCORES=RANK specifies that rank scores are used to compute these statistics, and the NOPRINT option suppresses the contingency tables. These statements produce Output 2.8.1 and Output 2.8.2.

```
proc freq data=Hypnosis;
   tables Subject*Emotion*SkinResponse
          / cmh2 scores=rank noprint;
run;
```

Output 2.8.1. CMH Statistics: Stratifying by Subject

```
                          The FREQ Procedure

               Summary Statistics for Emotion by SkinResponse
                          Controlling for Subject

          Cochran-Mantel-Haenszel Statistics (Based on Rank Scores)

       Statistic    Alternative Hypothesis    DF       Value      Prob
       ---------------------------------------------------------------
           1        Nonzero Correlation        1      0.2400    0.6242
           2        Row Mean Scores Differ     3      6.4500    0.0917

                          Total Sample Size = 32
```

Because the CMH statistics in Output 2.8.1 are based on rank scores, the Row Mean Scores Differ statistic is identical to Friedman's chi-square ($Q = 6.45$). The p-value of 0.0917 indicates that differences in skin potential response for different emotions are significant at the 10% level but not at the 5% level.

Output 2.8.2. CMH Statistics: No Stratification

```
                          The FREQ Procedure

              Summary Statistics for Emotion by SkinResponse

         Cochran-Mantel-Haenszel Statistics (Based on Rank Scores)

 Statistic    Alternative Hypothesis    DF       Value      Prob
 ---------------------------------------------------------------
     1        Nonzero Correlation        1      0.0001    0.9933
     2        Row Mean Scores Differ     3      0.5678    0.9038

                        Total Sample Size = 32
```

When you do not stratify by subject, the Row Mean Scores Differ CMH statistic is identical to a Kruskal-Wallis test and is not significant (p=0.9038 in Output 2.8.2). Thus, adjusting for subject is critical to reducing the background variation due to subject differences.

Example 2.9. Testing Marginal Homogeneity with Cochran's Q

When a binary response is measured several times or under different conditions, Cochran's Q tests that the marginal probability of a positive response is unchanged across the times or conditions. When there are more than two response categories, you can use the CATMOD procedure to fit a repeated-measures model.

The data set Drugs contains data for a study of three drugs to treat a chronic disease (Agresti 1990). Forty-six subjects receive drugs A, B, and C. The response to each drug is either favorable ('F') or unfavorable ('U').

```
proc format;
   value $ResponseFmt 'F'='Favorable'
                      'U'='Unfavorable';

data drugs;
   input Drug_A $ Drug_B $ Drug_C $ Count @@;
   datalines;
F F F  6   U F F  2
F F U 16   U F U  4
F U F  2   U U F  6
F U U  4   U U U  6
;
```

The following statements create one-way frequency tables of the responses to each drug. The AGREE option produces Cochran's Q and other measures of agreement for the three-way table. These statements produce Output 2.9.1 through Output 2.9.3.

```
proc freq data=Drugs;
   weight Count;
   tables Drug_A Drug_B Drug_C / nocum;
   tables Drug_A*Drug_B*Drug_C / agree noprint;
   format Drug_A Drug_B Drug_C $ResponseFmt.;
   title 'Study of Three Drug Treatments for a Chronic Disease';
run;
```

Output 2.9.1. One-Way Frequency Tables

```
              Study of Three Drug Treatments for a Chronic Disease

                             The FREQ Procedure

                    Drug_A         Frequency     Percent
                    ------------------------------------
                    Favorable            28       60.87
                    Unfavorable          18       39.13


                    Drug_B         Frequency     Percent
                    ------------------------------------
                    Favorable            28       60.87
                    Unfavorable          18       39.13


                    Drug_C         Frequency     Percent
                    ------------------------------------
                    Favorable            16       34.78
                    Unfavorable          30       65.22
```

The one-way frequency tables in Output 2.9.1 provide the marginal response for each drug. For drugs A and B, 61% of the subjects reported a favorable response while 35% of the subjects reported a favorable response to drug C.

Output 2.9.2. Measures of Agreement

```
           Study of Three Drug Treatments for a Chronic Disease

                 Statistics for Table 1 of Drug_B by Drug_C
                      Controlling for Drug_A=Favorable

                              McNemar's Test
                         ------------------------
                         Statistic (S)    10.8889
                         DF                     1
                         Pr > S            0.0010


                         Simple Kappa Coefficient
                     --------------------------------
                     Kappa                    -0.0328
                     ASE                       0.1167
                     95% Lower Conf Limit     -0.2615
                     95% Upper Conf Limit      0.1960

                             Sample Size = 28


                 Statistics for Table 2 of Drug_B by Drug_C
                     Controlling for Drug_A=Unfavorable

                             McNemar's Test
                         -----------------------
                         Statistic (S)    0.4000
                         DF                    1
                         Pr > S           0.5271


                         Simple Kappa Coefficient
                     --------------------------------
                     Kappa                    -0.1538
                     ASE                       0.2230
                     95% Lower Conf Limit     -0.5909
                     95% Upper Conf Limit      0.2832

                             Sample Size = 18


           Study of Three Drug Treatments for a Chronic Disease

                  Summary Statistics for Drug_B by Drug_C
                           Controlling for Drug_A

                        Overall Kappa Coefficient
                     --------------------------------
                     Kappa                    -0.0588
                     ASE                       0.1034
                     95% Lower Conf Limit     -0.2615
                     95% Upper Conf Limit      0.1439


                           Test for Equal Kappa
                               Coefficients
                           --------------------
                           Chi-Square    0.2314
                           DF                 1
                           Pr > ChiSq    0.6305


                          Total Sample Size = 46
```

McNemar's test (Output 2.9.2) shows strong discordance between drugs B and C when the response to drug A is favorable. The small negative value of the simple kappa indicates no agreement between drug B response and drug C response.

Output 2.9.3. Cochran's Q

```
              Study of Three Drug Treatments for a Chronic Disease

                    Summary Statistics for Drug_B by Drug_C
                             Controlling for Drug_A

                            Cochran's Q, for Drug_A
                              by Drug_B by Drug_C
                            -----------------------
                            Statistic (Q)    8.4706
                            DF                    2
                            Pr > Q           0.0145

                             Total Sample Size = 46
```

Cochran's Q is statistically significant (p=0.0144 in Output 2.9.3), which leads to rejection of the hypothesis that the probability of favorable response is the same for the three drugs.

References

Agresti, A. (1990), *Categorical Data Analysis*, New York: John Wiley & Sons, Inc.

Agresti, A. (1992), "A Survey of Exact Inference for Contingency Tables," *Statistical Science*, 7(1), 131–177.

Agresti, A. (1996), *An Introduction to Categorical Data Analysis*, New York: John Wiley & Sons, Inc.

Agresti, A., Mehta, C.R. and Patel, N.R. (1990), "Exact Inference for Contingency Tables with Ordered Categories," *Journal of the American Statistical Association*, 85, 453–458.

Agresti, A., Wackerly, D., and Boyett, J.M. (1979), "Exact Conditional Tests for Cross-Classifications: Approximation of Attained Significance Levels," *Psychometrika*, 44, 75–83.

Birch, M.W. (1965), "The Detection of Partial Association, II: The General Case," *Journal of the Royal Statistical Society, B*, 27, 111–124.

Bishop, Y., Fienberg, S.E., and Holland, P.W. (1975), *Discrete Multivariate Analysis: Theory and Practice*, Cambridge, MA: MIT Press.

Bowker, A.H. (1948), "Bowker's Test for Symmetry," *Journal of the American Statistical Association*, 43, 572–574.

Breslow, N.E. (1996), "Statistics in Epidemiology: The Case-Control Study," *Journal of the American Statistical Association*, 91, 14–26.

Breslow, N.E. and Day, N.E. (1980), *Statistical Methods in Cancer Research, Volume I: The Analysis of Case-Control Studies*, IARC Scientific Publications, No. 32, Lyon, France: International Agency for Research on Cancer.

Breslow, N.E. and Day, N.E. (1987), *Statistical Methods in Cancer Research, Volume II: The Design and Analysis of Cohort Studies*, IARC Scientific Publications, No. 82, Lyon, France: International Agency for Research on Cancer.

Bross, I.D.J. (1958), "How to Use Ridit Analysis," *Biometrics*, 14, 18–38.

Brown, M.B. and Benedetti, J.K. (1977), "Sampling Behavior of Tests for Correlation in Two-Way Contingency Tables," *Journal of the American Statistical Association*, 72, 309–315.

Cicchetti, D.V. and Allison, T. (1971), "A New Procedure for Assessing Reliability of Scoring EEG Sleep Recordings," *American Journal of EEG Technology*, 11, 101–109.

Cochran, W.G. (1950), "The Comparison of Percentages in Matched Samples," *Biometrika*, 37, 256–266.

Cochran, W.G. (1954), "Some Methods for Strengthening the Common χ^2 Tests," *Biometrics*, 10, 417–451.

Collett, D. (1991), *Modelling Binary Data*, London: Chapman & Hall.

Cohen, J. (1960), "A Coefficient of Agreement for Nominal Scales," *Educational and Psychological Measurement*, 20, 37–46.

Drasgow, F. (1986), "Polychoric and Polyserial Correlations" in *Encyclopedia of Statistical Sciences*, vol. 7, ed. S. Kotz and N. L. Johnson, New York: John Wiley & Sons, Inc., 68–74.

Fienberg, S.E. (1980), *The Analysis of Cross-Classified Data*, Second Edition, Cambridge, MA: MIT Press.

Fleiss, J.L. (1981), *Statistical Methods for Rates and Proportions*, Second Edition, New York: John Wiley & Sons, Inc.

Fleiss, J.L. and Cohen, J. (1973), "The Equivalence of Weighted Kappa and the Intraclass Correlation Coefficient as Measures of Reliability," *Educational and Psychological Measurement*, 33, 613–619.

Fleiss, J.L., Cohen, J., and Everitt, B.S. (1969), "Large-Sample Standard Errors of Kappa and Weighted Kappa," *Psychological Bulletin*, 72, 323–327.

Freeman, G.H. and Halton, J.H. (1951), "Note on an Exact Treatment of Contingency, Goodness of Fit and Other Problems of Significance," *Biometrika*, 38, 141–149.

Gail, M. and Mantel, N. (1977), "Counting the Number of $r \times c$ Contingency Tables with Fixed Margins," *Journal of the American Statistical Association*, 72, 859–862.

Gart, J.J. (1971), "The Comparison of Proportions: A Review of Significance Tests, Confidence Intervals and Adjustments for Stratification," *Review of the International Statistical Institute*, 39(2), 148–169.

Goodman, L.A. and Kruskal, W.H. (1979), *Measures of Association for Cross Classification*, New York: Springer-Verlag.

Greenland, S. and Robins, J.M. (1985), “Estimators of the Mantel-Haenszel Variance Consistent in Both Sparse Data and Large-Strata Limiting Models,” *Biometrics*, 42, 311–323.

Haldane, J.B.S. (1955), “The Estimation and Significance of the Logarithm of a Ratio of Frequencies,” *Annals of Human Genetics*, 20, 309–314.

Hollander, M. and Wolfe, D.A. (1973), *Nonparametric Statistical Methods*, New York: John Wiley & Sons, Inc.

Jones, M.P., O’Gorman, T.W., Lemka, J.H., and Woolson, R.F. (1989), “A Monte Carlo Investigation of Homogeneity Tests of the Odds Ratio Under Various Sample Size Configurations,” *Biometrics*, 45, 171–181.

Kendall, M. (1955), *Rank Correlation Methods*, Second Edition, London: Charles Griffin and Co.

Kendall, M. and Stuart, A. (1979), *The Advanced Theory of Statistics*, vol. 2, New York: Macmillan Publishing Company, Inc.

Kleinbaum, D.G., Kupper, L.L., and Morgenstern, H. (1982), *Epidemiologic Research: Principles and Quantitative Methods*, Research Methods Series, New York: Van Nostrand Reinhold.

Landis, R.J., Heyman, E.R., and Koch, G.G. (1978), “Average Partial Association in Three-way Contingency Tables: A Review and Discussion of Alternative Tests,” *International Statistical Review*, 46, 237–254.

Leemis, L.M. and Trivedi, K.S. (1996), “A Comparison of Approximate Interval Estimators for the Bernoulli Parameter,” *The American Statistician*, 50(1), 63–68.

Lehmann, E.L. (1975), *Nonparametrics: Statistical Methods Based on Ranks*, San Francisco: Holden-Day, Inc.

Liebetrau, A.M. (1983), *Measures of Association*, *Quantitative Application in the Social Sciences*, vol. 32, Beverly Hills: Sage Publications, Inc.

Mack, G.A. and Skillings, J.H. (1980), “A Friedman-Type Rank Test for Main Effects in a Two-Factor ANOVA,” *Journal of the American Statistical Association*, 75, 947–951.

Mantel, N. (1963), “Chi-square Tests with One Degree of Freedom: Extensions of the Mantel-Haenszel Procedure,” *Journal of the American Statistical Association*, 58, 690–700.

Mantel, N. and Haenszel, W. (1959), “Statistical Aspects of the Analysis of Data from Retrospective Studies of Disease,” *Journal of the National Cancer Institute*, 22, 719–748.

Margolin, B.H. (1988), “Test for Trend in Proportions,” in *Encyclopedia of Statistical Sciences*, vol. 9, ed. S. Kotz and N.L. Johnson, New York: John Wiley & Sons, Inc., 334–336.

McNemar, Q. (1947), “Note on the Sampling Error of the Difference between Correlated Proportions or Percentages,” *Psychometrika*, 12, 153–157.

Mehta, C.R. and Patel, N.R. (1983), “A Network Algorithm for Performing Fisher’s Exact Test in $r \times c$ Contingency Tables,” *Journal of the American Statistical Association*, 78, 427–434.

Mehta, C.R., Patel, N.R., and Gray, R. (1985), “On Computing an Exact Confidence Interval for the Common Odds Ratio in Several 2×2 Contingency Tables,” *Journal of the American Statistical Association*, 80, 969–973.

Mehta, C.R., Patel, N.R., and Senchaudhuri, P. (1991), “Exact Stratified Linear Rank Tests for Binary Data,” *Computing Science and Statistics: Proceedings of the 23rd Symposium on the Interface*, ed. E.M. Keramidas, 200–207.

Mehta, C.R., Patel, N.R., and Tsiatis, A.A. (1984), “Exact Significance Testing to Establish Treatment Equivalence with Ordered Categorical Data,” *Biometrics*, 40, 819–825.

Narayanan, A. and Watts, D. (1996), “Exact Methods in the NPAR1WAY Procedure,” in *Proceedings of the Twenty-First Annual SAS Users Group International Conference*, Cary, NC: SAS Institute Inc., 1290–1294.

Olsson, U. (1979), “Maximum Likelihood Estimation of the Polychoric Correlation Coefficient,” *Psychometrika*, 12, 443–460.

Pirie, W. (1983), “Jonckheere Tests for Ordered Alternatives,” in *Encyclopedia of Statistical Sciences*, vol. 4, ed. S. Kotz and N.L. Johnson, New York: John Wiley & Sons, Inc., 315–318.

Radlow, R. and Alf, E.F. (1975), “An Alternate Multinomial Assessment of the Accuracy of the Chi-Square Test of Goodness of Fit,” *Journal of the American Statistical Association*, 70, 811–813.

Robins, J.M., Breslow, N., and Greenland, S. (1986), “Estimators of the Mantel-Haenszel Variance Consistent in Both Sparse Data and Large-Strata Limiting Models,” *Biometrics*, 42, 311–323.

Snedecor, G.W. and Cochran, W.G. (1989), *Statistical Methods*, Eighth Edition, Ames, IA: Iowa State University Press.

Somers, R.H. (1962), “A New Asymmetric Measure of Association for Ordinal Variables,” *American Sociological Review*, 27, 799–811.

Stokes, M.E., Davis, C.S., and Koch, G.G. (1995), *Categorical Data Analysis Using the SAS System*, Cary, NC: SAS Institute Inc.

Tarone, R.E. (1985), “On Heterogeneity Tests Based on Efficient Scores,” *Biometrika*, 72, 1, 91–95.

Theil, H. (1972), *Statistical Decomposition Analysis*, Amsterdam: North-Holland Publishing Company.

Thomas, D.G. (1971), “Algorithm AS-36. Exact Confidence Limits for the Odds Ratio in a 2×2 Table,” *Applied Statistics*, 20, 105–110.

Valz, P.D. and Thompson, M.E. (1994), "Exact Inference for Kendall's S and Spearman's Rho with Extensions to Fisher's Exact Test in $r \times c$ Contingency Tables," *Journal of Computational and Graphical Statistics*, 3(4), 459–472.

van Elteren, P.H. (1960), "On the Combination of Independent Two-Sample Tests of Wilcoxon," *Bulletin of the International Statistical Institute*, 37, 351–361.

Vollset, S.E., Hirji, K.F., and Elashoff, R.M. (1991), "Fast Computation of Exact Confidence Limits for the Common Odds Ratio in a Series of 2×2 Tables," *Journal of the American Statistical Association*, 86, 404–409.

Woolf, B. (1955), "On Estimating the Relationship between Blood Group and Disease," *Annals of Human Genetics*, 19, 251–253.

Chapter 3
The UNIVARIATE Procedure

Chapter Contents

Chapter 3
The UNIVARIATE Procedure

Overview

The UNIVARIATE procedure provides the following:

- descriptive statistics based on moments (including skewness and kurtosis), quantiles or percentiles (such as the median), frequency tables, and extreme values
- histograms and comparative histograms. Optionally, these can be fitted with probability density curves for various distributions and with kernel density estimates.
- quantile-quantile plots (Q-Q plots) and probability plots. These plots facilitate the comparison of a data distribution with various theoretical distributions.
- goodness-of-fit tests for a variety of distributions including the normal
- the ability to inset summary statistics on plots produced on a graphics device
- the ability to analyze data sets with a frequency variable
- the ability to create output data sets containing summary statistics, histogram intervals, and parameters of fitted curves

You can use the PROC UNIVARIATE statement, together with the VAR statement, to compute summary statistics. See "Getting Started" on page 196 for introductory examples. In addition, you can use the following statements to request plots:

- the HISTOGRAM statement for creating histograms, the QQPLOT statement for creating Q-Q plots, and the PROBPLOT statement for creating probability plots
- the CLASS statement together with the HISTOGRAM, QQPLOT, and PROBPLOT statement for creating comparative histograms, Q-Q plots, and probability plots
- the INSET statement with any of the plot statements for enhancing the plot with an inset table of summary statistics. The INSET statement is applicable only to plots produced on graphics devices.

Getting Started

The following examples demonstrate how you can use the UNIVARIATE procedure to analyze the distributions of variables through the use of descriptive statistical measures and graphical displays, such as histograms.

Capabilities of PROC UNIVARIATE

The UNIVARIATE procedure provides a variety of descriptive measures, high-resolution graphical displays, and statistical methods, which you can use to summarize, visualize, analyze, and model the statistical distributions of numeric variables. These tools are appropriate for a broad range of tasks and applications:

- Exploring the distributions of the variables in a data set is an important preliminary step in data analysis, data warehousing, and data mining. With the UNIVARIATE procedure you can use tables and graphical displays, such as histograms and nonparametric density estimates, to find key features of distributions, identify outliers and extreme observations, determine the need for data transformations, and compare distributions.
- Modeling the distributions of data and validating distributional assumptions are basic steps in statistical analysis. You can use the UNIVARIATE procedure to fit parametric distributions (beta, exponential, gamma, lognormal, normal, and Weibull) and to compute probabilities and percentiles from these models. You can assess goodness of fit with hypothesis tests and with graphical displays such as probability plots and quantile-quantile plots. You can also use the UNIVARIATE procedure to validate distributional assumptions for other types of statistical analysis. When standard assumptions are not met, you can use the UNIVARIATE procedure to perform nonparametric tests and compute robust estimates of location and scale.
- Summarizing the distribution of the data is often helpful for creating effective statistical reports and presentations. You can use the UNIVARIATE procedure to create tables of summary measures, such as means and percentiles, together with graphical displays, such as histograms and comparative histograms, which facilitate the interpretation of the report.

The following examples illustrate a few of the tasks that you can carry out with the UNIVARIATE procedure.

Summarizing a Data Distribution

Figure 3.1 shows a table of basic summary measures and a table of extreme observations for the loan-to-value ratios of 5,840 home mortgages. The ratios are saved as values of the variable LoanToValueRatio in a data set named HomeLoans. The following statements request a univariate analysis:

```
ods select BasicMeasures ExtremeObs;
proc univariate data=homeloans;
   var LoanToValueRatio;
run;
```

The ODS SELECT statement restricts the default output to the tables for basic statistical measures and extreme observations.

```
                     The UNIVARIATE Procedure
         Variable:  LoanToValueRatio  (Loan to Value Ratio)

                    Basic Statistical Measures

           Location                    Variability

       Mean     0.292512     Std Deviation           0.16476
       Median   0.248050     Variance                0.02715
       Mode     0.250000     Range                   1.24780
                             Interquartile Range     0.16419


                       Extreme Observations

            -------Lowest------        -----Highest-----

                 Value      Obs            Value      Obs

             0.0651786        1          1.13976     5776
             0.0690157        3          1.14209     5791
             0.0699755       59          1.14286     5801
             0.0702412       84          1.17090     5799
             0.0704787        4          1.31298     5811
```

Figure 3.1. Basic Measures and Extreme Observations

The tables in Figure 3.1 show, in particular, that the average ratio is 0.2925 and the minimum and maximum ratios are 0.06518 and 1.1398, respectively.

Exploring a Data Distribution

Figure 3.2 shows a histogram of the loan-to-value ratios. The histogram reveals features of the ratio distribution, such as its skewness and the peak at 0.175, which are not evident from the tables in the previous example. The following statements create the histogram:

```
title 'Home Loan Analysis';
proc univariate data=homeloans noprint;
   histogram LoanToValueRatio / cfill=ltgray;
   inset n = 'Number of Homes' / position=ne;
run;
```

The NOPRINT option suppresses the display of summary statistics. The INSET statement inserts the total number of analyzed home loans in the northeast corner of the plot.

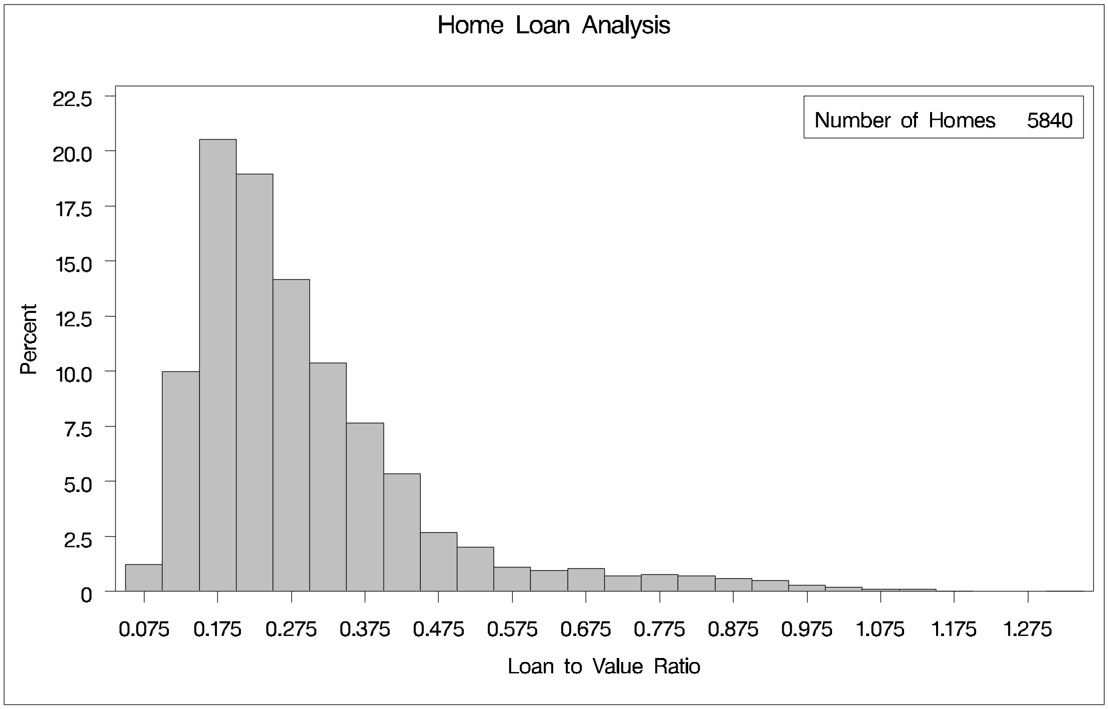


Figure 3.2. Histogram for Loan-to-Value Ratio

The data set HomeLoans contains a variable named LoanType that classifies the loans into two types: Gold and Platinum. It is useful to compare the distributions of LoanToValueRatio for the two types. The following statements request quantiles for each distribution and a comparative histogram, which are shown in Figure 3.3 and Figure 3.4.

```
title 'Comparison of Loan Types';
ods select Quantiles MyHist;
proc univariate data=HomeLoans;
   var LoanToValueRatio;
   class LoanType;
   histogram LoanToValueRatio / cfill=ltgray
                                kernel(color=black)
                                name='MyHist';
   inset n='Number of Homes' median='Median Ratio' (5.3) / position=ne;
   label LoanType = 'Type of Loan';
run;
```

The ODS SELECT statement restricts the default output to the tables of quantiles. The CLASS statement specifies LoanType as a classification variable for the quantile computations and comparative histogram. The KERNEL option adds a smooth nonparametric estimate of the ratio density to each histogram. The INSET statement specifies summary statistics to be displayed directly in the graph.

```
                    Comparison of Loan Types

                    The UNIVARIATE Procedure
       Variable:  LoanToValueRatio  (Loan to Value Ratio)
                        LoanType = Gold

                    Quantiles (Definition 5)

                    Quantile        Estimate

                    100% Max       1.0617647
                    99%            0.8974576
                    95%            0.6385908
                    90%            0.4471369
                    75% Q3         0.2985099
                    50% Median     0.2217033
                    25% Q1         0.1734568
                    10%            0.1411130
                    5%             0.1213079
                    1%             0.0942167
                    0% Min         0.0651786



                    Comparison of Loan Types

                    The UNIVARIATE Procedure
       Variable:  LoanToValueRatio  (Loan to Value Ratio)
                      LoanType = Platinum

                    Quantiles (Definition 5)

                    Quantile       Estimate

                    100% Max       1.312981
                    99%            1.050000
                    95%            0.691803
                    90%            0.549273
                    75% Q3         0.430160
                    50% Median     0.366168
                    25% Q1         0.314452
                    10%            0.273670
                    5%             0.253124
                    1%             0.231114
                    0% Min         0.215504
```

Figure 3.3. Quantiles for Loan-to-Value Ratio

The output in Figure 3.3 shows that the median ratio for Platinum loans (0.366) is greater than the median ratio for Gold loans (0.222). The comparative histogram in Figure 3.4 enables you to compare the two distributions more easily.

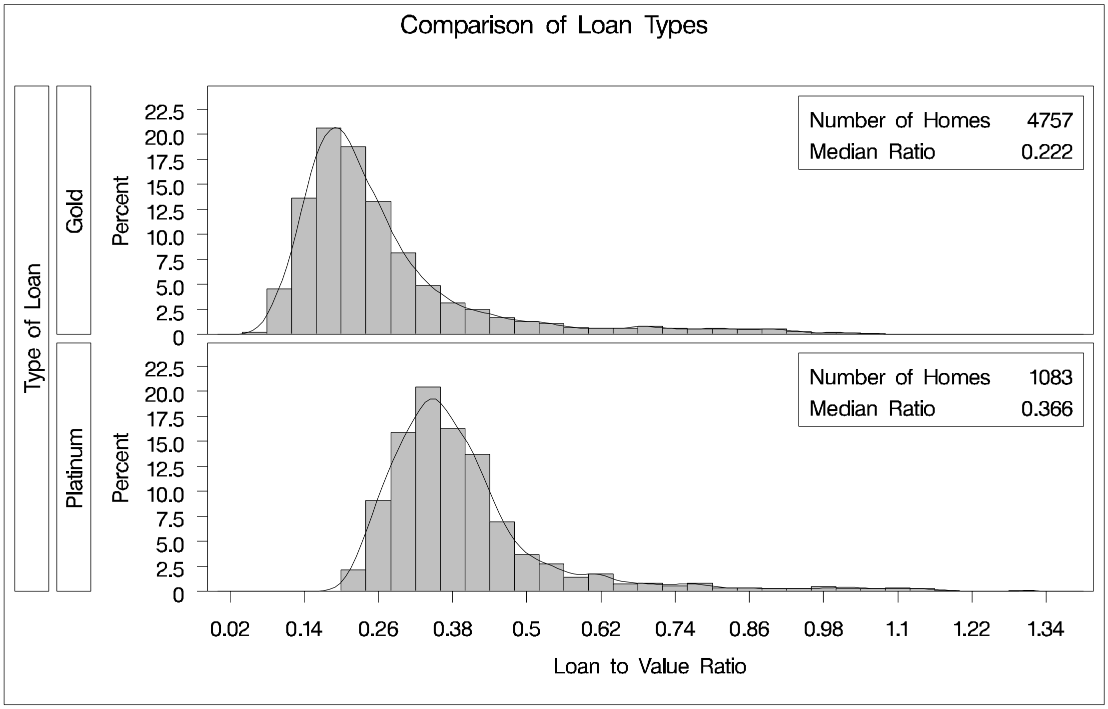


Figure 3.4. Comparative Histogram for Loan-to-Value Ratio

The comparative histogram shows that the ratio distributions are similar except for a shift of about 0.14.

A sample program, univar1.sas, for this example is available in the SAS Sample Library for Base SAS software.

Modeling a Data Distribution

In addition to summarizing a data distribution as in the preceding example, you can use PROC UNIVARIATE to statistically model a distribution based on a random sample of data. The following statements create a data set named Aircraft containing the measurements of a position deviation for a sample of 30 aircraft components.

```
data Aircraft;
   input Deviation @@;
   label Deviation = 'Position Deviation';
   datalines;
-.00653 0.00141 -.00702 -.00734 -.00649 -.00601
-.00631 -.00148 -.00731 -.00764 -.00275 -.00497
-.00741 -.00673 -.00573 -.00629 -.00671 -.00246
-.00222 -.00807 -.00621 -.00785 -.00544 -.00511
-.00138 -.00609 0.00038 -.00758 -.00731 -.00455
;
run;
```

An initial question in the analysis is whether the measurement distribution is normal. The following statements request a table of moments, the tests for normality, and a normal probability plot, which are shown in Figure 3.5 and Figure 3.6:

```
title 'Position Deviation Analysis';
ods select Moments TestsForNormality MyPlot;
proc univariate data=Aircraft normaltest;
   var Deviation;
   probplot Deviation / normal (mu=est sigma=est)
                        square name='MyPlot';
   label Deviation = 'Position Deviation';
   inset  mean std / format=6.4;
run;
```

```
                    Position Deviation Analysis

                     The UNIVARIATE Procedure
              Variable:  Deviation  (Position Deviation)

                              Moments

   N                          30    Sum Weights                 30
   Mean               -0.0053067    Sum Observations       -0.1592
   Std Deviation      0.00254362    Variance            6.47002E-6
   Skewness            1.2562507    Kurtosis            0.69790426
   Uncorrected SS     0.00103245    Corrected SS        0.00018763
   Coeff Variation    -47.932613    Std Error Mean       0.0004644


                        Tests for Normality

      Test                  --Statistic---    -----p Value------

      Shapiro-Wilk          W      0.845364    Pr < W      0.0005
      Kolmogorov-Smirnov    D      0.208921    Pr > D     <0.0100
      Cramer-von Mises      W-Sq   0.329274    Pr > W-Sq  <0.0050
      Anderson-Darling      A-Sq   1.784881    Pr > A-Sq  <0.0050
```

Figure 3.5. Moments and Tests for Normality

All four goodness-of-fit tests in Figure 3.5 reject the hypothesis that the measurements are normally distributed.

Figure 3.6 shows a normal probability plot for the measurements. A linear pattern of points following the diagonal reference line would indicate that the measurements are normally distributed. Instead, the curved point pattern suggests that a skewed distribution, such as the lognormal, is more appropriate than the normal distribution.

A lognormal distribution for Deviation is fitted in Example 3.26.

A sample program, univar2.sas, for this example is available in the SAS Sample Library for Base SAS software.

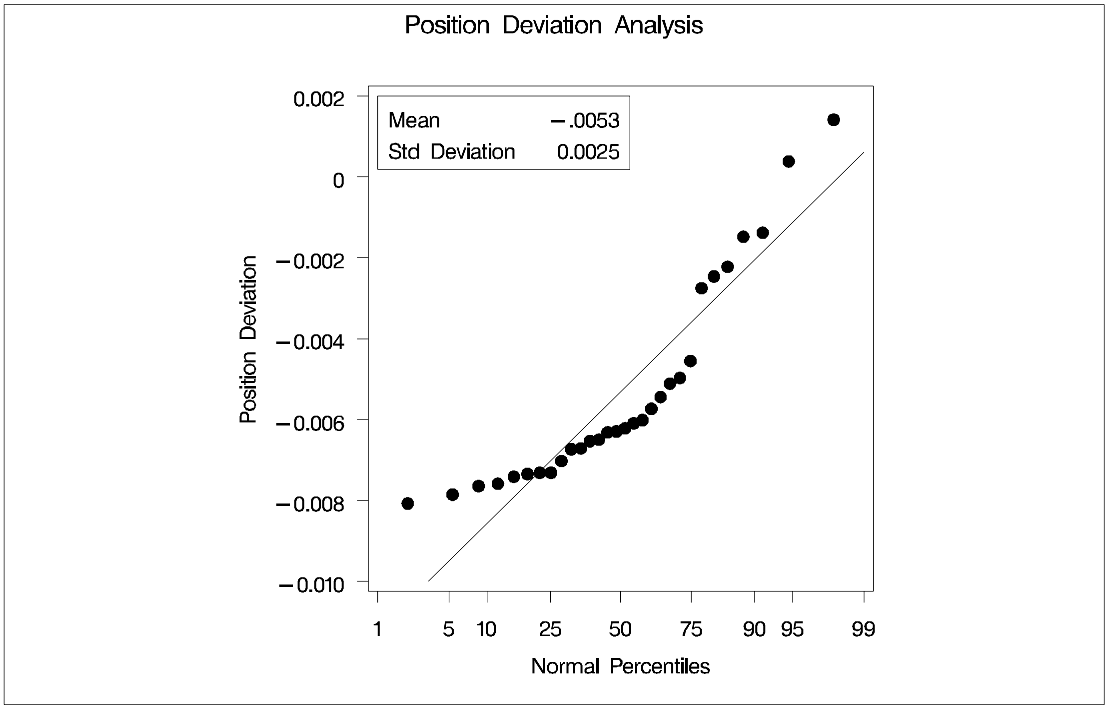


Figure 3.6. Normal Probability Plot

Syntax

PROC UNIVARIATE < *options* > **;**
  BY *variables* **;**
  CLASS *variable-1* <*(v-options)*>< *variable-2* <*(v-options)*>>
    < */ KEYLEVEL= value1* | *(value1 value2)* >**;**
  FREQ *variable* **;**
  HISTOGRAM < *variables* >< */ options* > **;**
  ID *variables* **;**
  INSET *keyword-list* < */ options* > **;**
  OUTPUT < **OUT=***SAS-data-set* >
    < *keyword1=names. . .keywordk=names* >< *percentile-options* >**;**
  PROBPLOT < *variables* >< */ options* > **;**
  QQPLOT < *variables* >< */ options* > **;**
  VAR *variables* **;**
  WEIGHT *variable* **;**

The PROC UNIVARIATE statement invokes the procedure. The VAR statement specifies the numeric variables to be analyzed, and it is required if the OUTPUT statement is used to save summary statistics in an output data set. If you do not use the VAR statement, all numeric variables in the data set are analyzed.

The plot statements HISTOGRAM, PROBPLOT, and QQPLOT create graphical displays, and the INSET statement enhances these displays by adding a table of summary statistics directly on the graph. You can specify one or more of each of the plot statements, the INSET statement, and the OUTPUT statement. If you use a VAR statement, the variables listed in a plot statement must be a subset of the variables listed in the VAR statement.

You can use a CLASS statement to specify one or two variables that group the data into classification levels. The analysis is carried out for each combination of levels, and you can use the CLASS statement with a plot statement to create a comparative display.

You can specify a BY statement to obtain separate analysis for each BY group. The FREQ statement specifies a variable whose values provide the frequency for each observation. The WEIGHT statement specifies a variable whose values are used to weight certain statistics. The ID statement specifies one or more variables to identify the extreme observations.

PROC UNIVARIATE Statement

PROC UNIVARIATE < *options* > **;**

The PROC UNIVARIATE statement is required to invoke the UNIVARIATE procedure. You can use the PROC UNIVARIATE statement by itself to request a variety of statistics for summarizing the data distribution of each analysis variable:

- sample moments
- basic measures of location and variability
- confidence intervals for the mean, standard deviation, and variance
- tests for location
- tests for normality
- trimmed and Winsorized means
- robust estimates of scale
- quantiles and related confidence intervals
- extreme observations and extreme values
- frequency counts for observations
- missing values

In addition, you can use options in the PROC UNIVARIATE statement to

- specify the input data set to be analyzed
- specify a graphics catalog for saving graphical output
- specify rounding units for variable values
- specify the definition used to calculate percentiles
- specify the divisor used to calculate variances and standard deviations
- request that plots be produced on line printers and define special printing characters used for features
- suppress tables

The following are the *options* that can be used with the PROC UNIVARIATE statement:

ALL

requests all statistics and tables that the FREQ, MODES, NEXTRVAL=5, PLOT, and CIBASIC options generate. If the analysis variables are not weighted, this option also requests the statistics and tables generated by the CIPCTLDF, CIPCTLNORMAL, LOCCOUNT, NORMAL, ROBUSTSCALE, TRIMMED=.25, and WINSORIZED=.25 options. PROC UNIVARIATE also uses any values that you specify for ALPHA=, MU0=, NEXTRVAL=, CIBASIC, CIPCTLDF, CIPCTLNORMAL, TRIMMED=, or WINSORIZED= to produce the output.

ALPHA=α

specifies the level of significance α for $100(1-\alpha)\%$ confidence intervals. The value α must be between 0 and 1; the default value is 0.05, which results in 95% confidence intervals.

Note that specialized ALPHA= options are available for a number of confidence interval options. For example, you can specify CIBASIC(ALPHA=0.10) to request a table of basic confidence limits at the 90% level. The default *values* of these options are the value of the ALPHA= option in the PROC statement.

ANNOTATE=*SAS-data-set*
ANNO=*SAS-data-set*

specifies an input data set that contains annotate variables as described in *SAS/GRAPH Reference*. You can use this data set to add features to your high-resolution graphics. PROC UNIVARIATE adds the features in this data set to every high-resolution graph that is produced in the procedure. PROC UNIVARIATE does not use the ANNOTATE= data set unless you create a high-resolution graph with the HISTOGRAM, PROBPLOT, or QQPLOT statement. Use the ANNOTATE= option in the HISTOGRAM, PROBPLOT, or QQPLOT statement if you want to add a feature to a specific graph produced by the statement.

CIBASIC <**(**<**TYPE=***keyword*><**ALPHA=**α >**)**>

requests confidence limits for the mean, standard deviation, and variance based on the assumption that the data are normally distributed. If you use the CIBASIC option, you must use the default value of VARDEF=, which is DF.

TYPE=*keyword*

specifies the type of confidence limit, where *keyword* is LOWER, UPPER, or TWOSIDED. The default value is TWOSIDED.

ALPHA=α

specifies the level of significance α for $100(1-\alpha)\%$ confidence intervals. The value α must be between 0 and 1; the default value is 0.05, which results in 95% confidence intervals. The default value is the value of ALPHA= given in the PROC statement.

CIPCTLDF <**(**<**TYPE=***keyword*><**ALPHA=**α >**)**>
CIQUANTDF <**(**<**TYPE=***keyword*><**ALPHA=**α >**)**>

requests confidence limits for quantiles based on a method that is distribution-free. In other words, no specific parametric distribution such as the normal is assumed for the data. PROC UNIVARIATE uses order statistics (ranks) to compute the confidence limits as described by Hahn and Meeker (1991). This option does not apply if you use a WEIGHT statement.

TYPE=*keyword*
specifies the type of confidence limit, where *keyword* is LOWER, UPPER, SYMMETRIC, or ASYMMETRIC. The default value is SYMMETRIC.

ALPHA=α
specifies the level of significance α for $100(1-\alpha)\%$ confidence intervals. The value α must be between 0 and 1; the default value is 0.05, which results in 95% confidence intervals. The default value is the value of ALPHA= given in the PROC statement.

CIPCTLNORMAL <**(**<**TYPE=***keyword*><**ALPHA=**α >**)**>
CIQUANTNORMAL <**(**<**TYPE=***keyword*><**ALPHA=**α >**)**>
requests confidence limits for quantiles based on the assumption that the data are normally distributed. The computational method is described in Section 4.4.1 of Hahn and Meeker (1991) and uses the noncentral t distribution as given by Odeh and Owen (1980). This option does not apply if you use a WEIGHT statement

TYPE=*keyword*
specifies the type of confidence limit, where *keyword* is LOWER, UPPER, or TWOSIDED. The default is TWOSIDED.

ALPHA=α
specifies the level of significance α for $100(1-\alpha)\%$ confidence intervals. The value α must be between 0 and 1; the default value is 0.05, which results in 95% confidence intervals. The default value is the value of ALPHA= given in the PROC statement.

DATA=*SAS-data-set*
specifies the input SAS data set to be analyzed. If the DATA= option is omitted, the procedure uses the most recently created SAS data set.

EXCLNPWGT
excludes observations with nonpositive weight values (zero or negative) from the analysis. By default, PROC UNIVARIATE treats observations with negative weights like those with zero weights and counts them in the total number of observations. This option applies only when you use a WEIGHT statement.

FREQ
requests a frequency table that consists of the variable values, frequencies, cell percentages, and cumulative percentages.

If you specify the WEIGHT statement, PROC UNIVARIATE includes the weighted count in the table and uses this value to compute the percentages.

GOUT=*graphics-catalog*
specifies the SAS catalog that PROC UNIVARIATE uses to save high-resolution graphics output. If you omit the libref in the name of the *graphics-catalog*, PROC UNIVARIATE looks for the catalog in the temporary library called WORK and creates the catalog if it does not exist.

LOCCOUNT

requests a table that shows the number of observations greater than, not equal to, and less than the value of MU0=. PROC UNIVARIATE uses these values to construct the sign test and the signed rank test. This option does not apply if you use a WEIGHT statement.

MODES|MODE

requests a table of all possible modes. By default, when the data contain multiple modes, PROC UNIVARIATE displays the lowest mode in the table of basic statistical measures. When all the values are unique, PROC UNIVARIATE does not produce a table of modes.

MU0=*values*
LOCATION=*values*

specifies the value of the mean or location parameter (μ_0) in the null hypothesis for tests of location summarized in the table labeled *Tests for Location: Mu0=value*. If you specify one value, PROC UNIVARIATE tests the same null hypothesis for all analysis variables. If you specify multiple values, a VAR statement is required, and PROC UNIVARIATE tests a different null hypothesis for each analysis variable in the corresponding order. The default *value* is 0.

The following statement tests the hypothesis $\mu_0 = 0$ for the first variable and the hypothesis $\mu_0 = 0.5$ for the second variable.

```
proc univariate mu0=0 0.5;
```

NEXTROBS=*n*

specifies the number of extreme observations that PROC UNIVARIATE lists in the table of extreme observations. The table lists the n lowest observations and the n highest observations. The default value is 5, and n can range between 0 and half the maximum number of observations. You can specify NEXTROBS=0 to suppress the table of extreme observations.

NEXTRVAL=*n*

specifies the number of extreme values that PROC UNIVARIATE lists in the table of extreme values. The table lists the n lowest unique values and the n highest unique values. The default value is 0, and n can range between 0 and half the maximum number of observations. By default, $n = 0$ and no table is displayed.

NOBYPLOT

suppresses side-by-side box plots that are created by default when you use the BY statement and the ALL option or the PLOT option in the PROC statement.

NOPRINT

suppresses all the tables of descriptive statistics that the PROC UNIVARIATE statement creates. NOPRINT does not suppress the tables that the HISTOGRAM statement creates. You can use the NOPRINT option in the HISTOGRAM statement to suppress the creation of its tables. Use NOPRINT when you want to create an OUT= output data set only.

NORMAL
NORMALTEST

requests tests for normality that include a series of goodness-of-fit tests based on the empirical distribution function. The table provides test statistics and p-values for the Shapiro-Wilk test (provided the sample size is less than or equal to 2000), the Kolmogorov-Smirnov test, the Anderson-Darling test, and the Cramér-von Mises test. This option does not apply if you use a WEIGHT statement.

PCTLDEF=*value*
DEF=*value*

specifies the definition that PROC UNIVARIATE uses to calculate quantiles. The default value is 5. Values can be 1, 2, 3, 4, or 5. You cannot use PCTLDEF= when you compute weighted quantiles. See the section "Calculating Percentiles" on page 273 for details on quantile definitions.

PLOTS | PLOT

produces a stem-and-leaf plot (or a horizontal bar chart), a box plot, and a normal probability plot in line printer output. If you use a BY statement, side-by-side box plots that are labeled "Schematic Plots" appear after the univariate analysis for the last BY group.

PLOTSIZE=*n*

specifies the approximate number of rows used in line-printer plots requested with the PLOTS option. If n is larger than the value of the SAS system option PAGESIZE=, PROC UNIVARIATE uses the value of PAGESIZE=. If n is less than 8, PROC UNIVARIATE uses eight rows to draw the plots.

ROBUSTSCALE

produces a table with robust estimates of scale. The statistics include the interquartile range, Gini's mean difference, the median absolute deviation about the median (*MAD*), and two statistics proposed by Rousseeuw and Croux (1993), Q_n, and S_n. This option does not apply if you use a WEIGHT statement.

ROUND=*units*

specifies the units to use to round the analysis variables prior to computing statistics. If you specify one unit, PROC UNIVARIATE uses this unit to round all analysis variables. If you specify multiple units, a VAR statement is required, and each unit rounds the values of the corresponding analysis variable. If ROUND=0, no rounding occurs. The ROUND= option reduces the number of unique variable values, thereby reducing memory requirements for the procedure. For example, to make the rounding unit 1 for the first analysis variable and 0.5 for the second analysis variable, submit the statement

```
proc univariate round=1 0.5;
   var yldstren tenstren;
run;
```

When a variable value is midway between the two nearest rounded points, the value is rounded to the nearest even multiple of the roundoff value. For example, with a roundoff value of 1, the variable values of -2.5, -2.2, and -1.5 are rounded to -2; the values of -0.5, 0.2, and 0.5 are rounded to 0; and the values of 0.6, 1.2, and 1.4 are rounded to 1.

TRIMMED=*values* <**(**<**TYPE=***keyword*><**ALPHA=**α >**)**>
TRIM=*values* <**(**<**TYPE=***keyword*><**ALPHA=**α >**)**>

requests a table of trimmed means, where *value* specifies the number or the proportion of observations that PROC UNIVARIATE trims. If the *value* is the number n of trimmed observations, n must be between 0 and half the number of nonmissing observations. If *value* is a proportion p between 0 and $\frac{1}{2}$, the number of observations that PROC UNIVARIATE trims is the smallest integer that is greater than or equal to np, where n is the number of observations. To include confidence limits for the mean and the Student's t test in the table, you must use the default value of VARDEF= which is DF. For details concerning the computation of trimmed means, see the section "Trimmed Means" on page 279.

TYPE= *keyword*

specifies the type of confidence limit for the mean, where *keyword* is LOWER, UPPER, or TWOSIDED. The default value is TWOSIDED.

ALPHA=α

specifies the level of significance α for $100(1-\alpha)\%$ confidence intervals. The value α must be between 0 and 1; the default value is 0.05, which results in 95% confidence intervals.

This option does not apply if you use a WEIGHT statement.

VARDEF=*divisor*

specifies the divisor to use in the calculation of variances and standard deviation. By default, VARDEF=DF. The following table shows the possible values for *divisor* and associated divisors.

Table 3.1. 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 $\frac{CSS}{\text{divisor}}$ where CSS is the corrected sums of squares and equals $\sum_{i=1}^{n}(x_i - \bar{x})^2$. When you weight the analysis variables, $CSS = \sum_{i=1}^{n}(w_i x_i - \bar{x}_w)^2$ where $\bar{x}_w$ is the weighted mean.

The default value is DF. To compute the standard error of the mean, confidence limits, and Student's t test, use the default value of VARDEF=.

When you use the WEIGHT statement and VARDEF=DF, the variance is an estimate of s^2 where the variance of the ith observation is $var(x_i) = \frac{s^2}{w_i}$ and w_i is the weight for the ith observation. This yields an estimate of the variance of an observation with unit weight.

When you use the WEIGHT statement and VARDEF=WGT, the computed variance is asymptotically (for large n) an estimate of $\frac{s^2}{\bar{w}}$ where $\bar{w}$ is the average weight. This yields an asymptotic estimate of the variance of an observation with average weight.

WINSORIZED=*values* <**(**<**TYPE=***keyword*><**ALPHA=**α >**)**>
WINSOR=*values* <**(**<**TYPE=***keyword*><**ALPHA=**α**>)>**

requests of a table of Winsorized means, where *value* is the number or the proportion of observations that PROC UNIVARIATE uses to compute the Winsorized mean. If the *value* is the number n of winsorized observations, n must be between 0 and half the number of nonmissing observations. If *value* is a proportion p between 0 and $\frac{1}{2}$, the number of observations that PROC UNIVARIATE uses is equal to the smallest integer that is greater than or equal to np, where n is the number of observations. To include confidence limits for the mean and the student t test in the table, you must use the default value of VARDEF=, which is DF. For details concerning the computation of Winsorized means, see the section "Winsorized Means" on page 278.

TYPE=*keyword*

specifies the type of confidence limit for the mean, where *keyword* is LOWER, UPPER, or TWOSIDED. The default is TWOSIDED.

ALPHA=α

specifies the level of significance α for $100(1-\alpha)\%$ confidence intervals. The value α must be between 0 and 1; the default value is 0.05, which results in 95% confidence intervals.

This option does not apply if you use a WEIGHT statement.

BY Statement

BY *variables* **;**

You can specify a BY statement with PROC UNIVARIATE to obtain separate analyses for each BY group. The BY statement specifies the *variables* 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.

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, 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, the procedure treats each contiguous set as a separate BY group.

CLASS Statement

CLASS *variable-1* <*(v-options)*>< *variable-2* <*(v-options)*>>
< */ KEYLEVEL= value1* | *(value1 value2)* >**;**

The CLASS statement specifies one or two variables that the procedure uses to group the data into classification levels. Variables in a CLASS statement are referred to as *class variables*. Class variables can be numeric or character. Class variables can have floating point 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. PROC UNIVARIATE uses the formatted values of the class variables to determine the classification levels.

You can specify the following *v-options* enclosed in parentheses after the class variable:

MISSING

specifies that missing values for the CLASS variable are to be treated as valid classification levels. Special missing values that represent numeric values (the letters A through Z and the underscore (_) character) are each considered as a separate value. If you omit MISSING, PROC UNIVARIATE excludes the observations with a missing class variable value from the analysis. Enclose this option in parentheses after the class variable.

ORDER=DATA | FORMATTED | FREQ | INTERNAL

specifies the display order for the class variable values. The default value is INTERNAL. You can specify the following values with the ORDER=*option*:

DATA

orders values according to their order in the input data set. When you use a HISTOGRAM, PROBPLOT, or QQPLOT statement, PROC UNIVARIATE displays the rows (columns) of the comparative plot from top to bottom (left to right) in the order that the class variable values first appear in the input data set.

FORMATTED

orders values by their ascending formatted values. This order may depend on your operating environment. When you use a HISTOGRAM, PROBPLOT, or QQPLOT statement, PROC UNIVARIATE displays the rows (columns) of the comparative plot from top to bottom (left to right) in increasing order of the formatted class variable values. For example, suppose a numeric class variable DAY (with values 1, 2, and 3) has a user-defined format that assigns Wednesday to the value 1, Thursday to the value 2, and Friday to the value 3. The rows of the comparative plot will appear in alphabetical order (Friday, Thursday, Wednesday) from top to bottom.

If there are two or more distinct internal values with the same formatted value, then PROC UNIVARIATE determines the order by the internal value that occurs first in the input data set. For numerical variables without an explicit format, the levels are ordered by their internal values.

FREQ

orders values by descending frequency count so that levels with the most observations are listed first. If two or more values have the same frequency count, PROC UNIVARIATE uses the formatted values to determine the order.

When you use a HISTOGRAM, PROBPLOT, or QQPLOT statement, PROC UNIVARIATE displays the rows (columns) of the comparative plot from top to bottom (left to right) in order of decreasing frequency count for the class variable values.

INTERNAL

orders values by their unformatted values, which yields the same order as PROC SORT. This order may depend on your operating environment.

When you use a HISTOGRAM, PROBPLOT, or QQPLOT statement, PROC UNIVARIATE displays the rows (columns) of the comparative plot from top to bottom (left to right) in increasing order of the internal (unformatted) values of the class variable. The first class variable is used to label the rows of the comparative plots (top to bottom). The second class variable is used to label the columns of the comparative plots (left to right). For example, suppose a numeric class variable DAY (with values 1, 2, and 3) has a user-defined format that assigns Wednesday to the value 1, Thursday to the value 2, and Friday to the value 3. The rows of the comparative plot will appear in day-of-the-week order (Wednesday, Thursday, Friday) from top to bottom.

You can specify the following *option* after the slash (/) in the CLASS statement.

KEYLEVEL=*value1* | (*value1 value2*)

specifies the *key cell* in a comparative plot. PROC UNIVARIATE first determines the bin size and midpoints for the key cell, and then extends the midpoint list to accommodate the data ranges for the remaining cells. Thus, the choice of the key cell determines the uniform horizontal axis that PROC UNIVARIATE uses for all cells. If you specify only one class variable and use a HISTOGRAM statement, KEYLEVEL= *value* identifies the key cell as the level for which variable is equal to value. By default, PROC UNIVARIATE sorts the levels in the order that is determined by the ORDER= option. Then, the key cell is the first occurrence of a level in this order. The cells display in order from top to bottom or left to right. Consequently, the key cell appears at the top (or left). When you specify a different key cell with the KEYLEVEL= option, this cell appears at the top (or left).

Likewise, with the PROBPLOT and QQPLOT statements, the key cell determines uniform axis scaling. If you specify two class variables, use KEYLEVEL= *value1 value2* to identify the key cell as the level for which variable-*n* is equal to value-*n*.

By default, PROC UNIVARIATE sorts the levels of the first CLASS variable in the order that is determined by its ORDER= option and, within each of these levels, it sorts the levels of the second CLASS variable in the order that is determined by its ORDER= option. Then, the default key cell is the first occurrence of a combination of levels for the two variables in this order. The cells display in the order of the first CLASS variable from top to bottom and in the order of the second CLASS variable from left to right. Consequently, the default key cell appears at the upper left corner.

When you specify a different key cell with the KEYLEVEL= option, this cell appears at the upper left corner.

The length of the KEYLEVEL= value cannot exceed 16 characters and you must specify a formatted value.

The KEYLEVEL= option does not apply unless you specify a HISTOGRAM, PROBPLOT, or QQPLOT statement.

FREQ Statement

FREQ *variable* **;**

The FREQ statement 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 the variable is not an integer, the SAS System truncates it. If the variable is less than 1 or is missing, the procedure excludes that observation from the analysis. See Example 3.6.

Note: The FREQ statement affects the degrees of freedom, but the WEIGHT statement does not.

HISTOGRAM Statement

HISTOGRAM < *variables* >< */ options* >**;**

The HISTOGRAM statement creates histograms and optionally superimposes estimated parametric and nonparametric probability density curves. You cannot use the WEIGHT statement with the HISTOGRAM statement. You can use any number of HISTOGRAM statements after a PROC UNIVARIATE statement. The components of the HISTOGRAM statement are described as follows.

variables

are the variables for which histograms are to be created. If you specify a VAR statement, the *variables* must also be listed in the VAR statement. Otherwise, the *variables* can be any numeric variables in the input data set. If you do not specify *variables* in a VAR statement or in the HISTOGRAM statement, then by default, a histogram is created for each numeric variable in the DATA= data set. If you use a VAR statement and do not specify any *variables* in the HISTOGRAM statement, then by default, a histogram is created for each variable listed in the VAR statement.

For example, suppose a data set named Steel contains exactly two numeric variables named Length and Width. The following statements create two histograms, one for Length and one for Width:

```
proc univariate data=Steel;
   histogram;
run;
```

Likewise, the following statements create histograms for Length and Width:

```
proc univariate data=Steel;
   var Length Width;
   histogram;
run;
```

The following statements create a histogram for Length only:

```
proc univariate data=Steel;
   var Length Width;
   histogram Length;
run;
```

options

add features to the histogram. Specify all *options* after the slash (/) in the HISTOGRAM statement. *Options* can be one of the following:

- *primary options* for fitted parametric distributions and kernel density estimates
- *secondary options* for fitted parametric distributions and kernel density estimates
- *general options* for graphics and output data sets

For example, in the following statements, the NORMAL option displays a fitted normal curve on the histogram, the MIDPOINTS= option specifies midpoints for the histogram, and the CTEXT= option specifies the color of the text:

```
proc univariate data=Steel;
   histogram Length / normal
                      midpoints = 5.6 5.8 6.0 6.2 6.4
                      ctext     = blue;
run;
```

Table 3.2 through Table 3.12 list the HISTOGRAM *options* by function. For complete descriptions, see the the section "Dictionary of Options" on page 217.

Parametric Density Estimation Options

Table 3.2 lists *primary options* that display a parametric density estimate on the histogram.

Table 3.2. Primary Options for Parametric Fitted Distributions

BETA(*beta-options*)	Fits beta distribution with threshold parameter θ, scale parameter σ, and shape parameters α and β
EXPONENTIAL(*exponential-options*)	Fits exponential distribution with threshold parameter θ and scale parameter σ
GAMMA(*gamma-options*)	Fits gamma distribution with threshold parameter θ, scale parameter σ, and shape parameter α
LOGNORMAL(*lognormal-options*)	Fits lognormal distribution with threshold parameter θ, scale parameter ζ, and shape parameter σ
NORMAL(*normal-options*)	Fits normal distribution with mean μ and standard deviation σ
WEIBULL(*Weibull-options*)	Fits Weibull distribution with threshold parameter θ, scale parameter σ, and shape parameter c

Table 3.3 through Table 3.9 list *secondary options* that specify parameters for fitted parametric distributions and that control the display of fitted curves. Specify these *secondary options* in parentheses after the *primary distribution option*. For example, you can fit a normal curve by specifying the NORMAL option as follows:

```
proc univariate;
   histogram / normal(color=red mu=10 sigma=0.5);
run;
```

The COLOR= *normal-option* draws the curve in red, and the MU= and SIGMA= *normal-options* specify the parameters $\mu = 10$ and $\sigma = 0.5$ for the curve. Note that the sample mean and sample standard deviation are used to estimate μ and σ, respectively, when the MU= and SIGMA= *normal-options* are not specified.

Table 3.3. Secondary Options Used with All Parametric Distribution Options

COLOR=*color*	Specifies color of density curve
FILL	Fills area under density curve
L=*linetype*	Specifies line type of curve
MIDPERCENTS	Prints table of midpoints of histogram intervals
NOPRINT	Suppresses tables summarizing curve
PERCENTS=*value-list*	Lists percents for which quantiles calculated from data and quantiles estimated from curve are tabulated
W=n	Specifies width of density curve

Table 3.4. Secondary Beta-Options

ALPHA=*value*	Specifies first shape parameter α for beta curve
BETA=*value*	Specifies second shape parameter β for beta curve
SIGMA=*value* | EST	Specifies scale parameter σ for beta curve
THETA=*value* | EST	Specifies lower threshold parameter θ for beta curve

Table 3.5. Secondary Exponential-Options

SIGMA=*value*	Specifies scale parameter σ for exponential curve
THETA=*value* | EST	Specifies threshold parameter θ for exponential curve

Table 3.6. Secondary Gamma-Options

ALPHA=*value*	Specifies shape parameter α for gamma curve
SIGMA=*value*	Specifies scale parameter σ for gamma curve
THETA=*value* | EST	Specifies threshold parameter θ for gamma curve

Table 3.7. Secondary Lognormal-Options

SIGMA=*value*	Specifies shape parameter σ for lognormal curve
THETA=*value* | EST	Specifies threshold parameter θ for lognormal curve
ZETA=*value*	Specifies scale parameter ζ for lognormal curve

Table 3.8. Secondary Normal-Options

MU=*value*	Specifies mean μ for normal curve
SIGMA=*value*	Specifies standard deviation σ for normal curve

Table 3.9. Secondary Weibull-Options

C=*value*	Specifies shape parameter *c* for Weibull curve
SIGMA=*value*	Specifies scale parameter σ for Weibull curve
THETA=*value* | EST	Specifies threshold parameter θ for Weibull curve

Nonparametric Density Estimation Options

Use the *option* KERNEL(*kernel-options*) to compute kernel density estimates. Specify the following *secondary options* in parentheses after the KERNEL option to control features of density estimates requested with the KERNEL option.

Table 3.10. Kernel-Options

C=*value-list* | MISE	Specifies standardized bandwidth parameter c
COLOR=*color*	Specifies color of the kernel density curve
FILL	Fills area under kernel density curve
K=NORMAL | QUADRATIC | TRIANGULAR	Specifies type of kernel function
L=*linetype*	Specifies line type used for kernel density curve
LOWER=	Specifies lower bound for kernel density curve
UPPER=	Specifies upper bound for kernel density curve
W=n	Specifies line width for kernel density curve

General Options

Table 3.11 summarizes *options* for enhancing histograms, and Table 3.12 summarizes options for requesting output data sets.

Table 3.11. General Graphics Options

Option	Description
ANNOKEY	Applies annotation requested in ANNOTATE= data set to key cell only
ANNOTATE=	Specifies annotate data set
BARWIDTH=	Specifies width for the bars
CAXIS=	Specifies color for axis
CBARLINE=	Specifies color for outlines of histogram bars
CFILL=	Specifies color for filling under curve
CFRAME=	Specifies color for frame
CFRAMESIDE=	Specifies color for filling frame for row labels
CFRAMETOP=	Specifies color for filling frame for column labels
CGRID=	Specifies color for grid lines
CHREF=	Specifies color for HREF= lines
CPROP=	Specifies color for proportion of frequency bar
CTEXT=	Specifies color for text
CTEXTSIDE=	Specifies color for row labels of comparative histograms
CTEXTTOP=	Specifies color for column labels of comparative histograms
CVREF=	Specifies color for VREF= lines
DESCRIPTION=	Specifies description for plot in graphics catalog
ENDPOINTS=	Lists endpoints for histogram intervals
FONT=	Specifies software font for text
FORCEHIST	Forces creation of histogram
GRID	Creates a grid
FRONTREF	Draws reference lines in front of histogram bars
HEIGHT=	Specifies height of text used outside framed areas
HMINOR=	Specifies number of horizontal minor tick marks
HOFFSET=	Specifies offset for horizontal axis
HREF=	Specifies reference lines perpendicular to the horizontal axis
HREFLABELS=	Specifies labels for HREF= lines
HREFLABPOS=	Specifies vertical position of labels for HREF= lines
INFONT=	Specifies software font for text inside framed areas
INHEIGHT=	Specifies height of text inside framed areas
INTERTILE=	Specifies distance between tiles
LGRID=	Specifies a line type for grid lines
LHREF=	Specifies line style for HREF= lines
LVREF=	Specifies line style for VREF= lines
MAXNBIN=	Specifies maximum number of bins to display
MAXSIGMAS=	Limits the number of bins that display to within a specified number of standard deviations above and below mean of data in key cell
MIDPOINTS=	Lists midpoints for histogram intervals
NAME=	Specifies name for plot in graphics catalog
NCOLS=	Specifies number of columns in comparative histogram
NOBARS	Suppresses histogram bars
NOFRAME	Suppresses frame around plotting area

Table 3.11. (continued)

Option	Description
NOHLABEL	Suppresses label for horizontal axis
NOPLOT	Suppresses plot
NOVLABEL	Suppresses label for vertical axis
NOVTICK	Suppresses tick marks and tick mark labels for vertical axis
NROWS=	Specifies number of rows in comparative histogram
PFILL=	Specifies pattern for filling under curve
RTINCLUDE	Includes right endpoint in interval
TURNVLABELS	Turn and vertically string out characters in labels for vertical axis
VAXIS=	Specifies AXIS statement or values for vertical axis
VAXISLABEL=	Specifies label for vertical axis
VMINOR=	Specifies number of vertical minor tick marks
VOFFSET=	Specifies length of offset at upper end of vertical axis
VREF=	Specifies reference lines perpendicular to the vertical axis
VREFLABELS=	Specifies labels for VREF= lines
VREFLABPOS=	Specifies horizontal position of labels for VREF= lines
VSCALE=	Specifies scale for vertical axis
WAXIS=	Specifies line thickness for axes and frame
WBARLINE=	Specifies line thickness for bar outlines
WGRID=	Specifies line thickness for grid

Table 3.12. Options for Requesting Output Data Sets

Option	Description
MIDPERCENTS	Creates table of histogram intervals
OUTHISTOGRAM=	Specifies information on histogram intervals

Dictionary of Options

The following entries provide detailed descriptions of *options* in the HISTOGRAM statement.

ALPHA=*value*

specifies the shape parameter α for fitted curves requested with the BETA and GAMMA options. Enclose the ALPHA= option in parentheses after the BETA or GAMMA options. By default, the procedure calculates a maximum likelihood estimate for α. You can specify A= as an alias for ALPHA= if you use it as a *beta-option*. You can specify SHAPE= as an alias for ALPHA= if you use it as a *gamma-option*.

ANNOKEY

applies the annotation requested with the ANNOTATE= option to the key cell only. By default, the procedure applies annotation to all of the cells. This option is not available unless you use the CLASS statement. You can use the KEYLEVEL= option in the CLASS statement to specify the key cell.

ANNOTATE=*SAS-data-set*
ANNO=*SAS-data-set*

specifies an input data set containing annotate variables as described in *SAS/GRAPH Software: Reference*. The ANNOTATE= data set you specify in the HISTOGRAM statement is used for all plots created by the statement. You can also specify an ANNOTATE= data set in the PROC UNIVARIATE statement to enhance all plots created by the procedure.

BARWIDTH=*value*

specifies the width of the histogram bars in screen percent units.

BETA <**(***beta-options***)**>

displays a fitted beta density curve on the histogram. The BETA option can occur only once in a HISTOGRAM statement. The beta distribution is bounded below by the parameter θ and above by the value $\theta + \sigma$. Use the THETA= and SIGMA= *beta-options* to specify these parameters. By default, THETA=0 and SIGMA=1. You can specify THETA=EST and SIGMA=EST to request maximum likelihood estimates for θ and σ. See Example 3.21.

Note: Three- and four-parameter maximum likelihood estimation may not always converge. The beta distribution has two shape parameters, α and β. If these parameters are known, you can specify their values with the ALPHA= and BETA= *beta-options*. By default, the procedure computes maximum likelihood estimates for α and β. Table 3.3 (page 214) and Table 3.4 (page 215) list options you can specify with the BETA option.

BETA=*value*
B=*value*

specifies the second shape parameter β for beta density curves requested with the BETA option. Enclose the BETA= option in parentheses after the BETA option. By default, the procedure calculates a maximum likelihood estimate for β.

C=*value*

specifies the shape parameter c for Weibull density curves requested with the WEIBULL option. Enclose the C= *Weibull-option* in parentheses after the WEIBULL option. If you do not specify a value for c, the procedure calculates a maximum likelihood estimate. You can specify the SHAPE= *Weibull-option* as an alias for the C= *Weibull-option*.

C=*value-list* **| MISE**

specifies the standardized bandwidth parameter c for kernel density estimates requested with the KERNEL option. Enclose the C= *kernel-option* in parentheses after the KERNEL option. You can specify up to five values to request multiple estimates. You can also specify the C=MISE option, which produces the estimate with a bandwidth that minimizes the approximate mean integrated square error (MISE).

You can also use the C= *kernel-option* with the K= *kernel-option*, which specifies the kernel function, to compute multiple estimates. If you specify more kernel functions than bandwidths, the last bandwidth in the list is repeated for the remaining estimates. Likewise, if you specify more bandwidths than kernel functions, the last kernel function is repeated for the remaining estimates. If you do not specify a value for c, the bandwidth that minimizes the approximate MISE is used for all the estimates.

CAXIS=*color*
CAXES=*color*
CA=*color*
specifies the color for the axes and tick marks. This option overrides any COLOR= specifications in an AXIS statement. The default value is the first color in the device color list.

CBARLINE=*color*
specifies the color for the outline of the histogram bars. This option overrides the C= option in the SYMBOL1 statement. The default value is the first color in the device color list.

CFILL=*color*
specifies the color to fill the bars of the histogram (or the area under a fitted density curve if you also specify the FILL option). See the entries for the FILL and PFILL= options for additional details. Refer to *SAS/GRAPH Software: Reference* for a list of colors. By default, bars and curve areas are not filled.

CFRAME=*color*
specifies the color for the area that is enclosed by the axes and frame. The area is not filled by default.

CFRAMESIDE=*color*
specifies the color to fill the frame area for the row labels that display along the left side of the comparative histogram. This color also fills the frame area for the label of the corresponding class variable (if you associate a label with the variable). By default, these areas are not filled. This option is not available unless you use the CLASS statement.

CFRAMETOP=*color*
specifies the color to fill the frame area for the column labels that display across the top of the comparative histogram. This color also fills the frame area for the label of the corresponding class variable (if you associate a label with the variable). By default, these areas are not filled. This option is not available unless you use the CLASS statement.

CGRID=*color*
specifies the color for grid lines when a grid displays on the histogram. The default *color* is the first color in the device color list. This option also produces a grid.

CHREF=*color*
CH=*color*
specifies the color for horizontal axis reference lines requested by the HREF= option. The default is the first color in the device color list.

COLOR=*color*
specifies the color of the density curve. Enclose the COLOR= option in parentheses after the distribution option or the KERNEL option. If you use the COLOR= option with the KERNEL option, you can specify a list of up to five colors in parentheses

for multiple kernel density estimates. If there are more estimates than colors, the last color specified is used for the remaining estimates.

CPROP=*color* **| EMPTY**

specifies the color for a horizontal bar whose length (relative to the width of the tile) indicates the proportion of the total frequency that is represented by the corresponding cell in a comparative histogram. By default, no bars are displayed. This option is not available unless you use the CLASS statement. You can specify the keyword EMPTY to display empty bars. See Example 3.20.

CTEXT=*color*
CT=*color*

specifies the color for tick mark values and axis labels. The default is the color specified for the CTEXT= option in the GOPTIONS statement. In the absence of a GOPTIONS statement, the default color is the first color in the device color list.

CTEXTSIDE=*color*

specifies the color for the row labels that display along the left side of the comparative histogram. By default, the color specified by the CTEXT= option is used. If you omit the CTEXT= option, the color specified in the GOPTIONS statement is used. If you omit the GOPTIONS statement, the the first color in the device color list is used. This option is not available unless you use the CLASS statement. You can specify the CFRAMESIDE= option to change the background color for the row labels.

CTEXTTOP=*color*

specifies the color for the column labels that display along the left side of the comparative histogram. By default, the color specified by the CTEXT= option is used. If you omit the CTEXT= option, the color specified in the GOPTIONS statement is used. If you omit the GOPTIONS statement, the the first color in the device color list is used. This option is not available unless you specify the CLASS statement. You can use the CFRAMETOP= option to change the background color for the column labels.

CVREF=*color*
CV=*color*

specifies the color for lines requested with the VREF= option. The default is the first color in the device color list.

DESCRIPTION='*string***'**
DES='*string***'**

specifies a description, up to 40 characters long, that appears in the PROC GREPLAY master menu. The default value is the variable name.

ENDPOINTS <**=***values* **| KEY | UNIFORM**>

uses the endpoints as the tick mark values for the horizontal axis and determines how to compute the bin width of the histogram bars, where *values* specifies values for both the left and right endpoint of each histogram interval. The width of the histogram bars is the difference between consecutive endpoints. The procedure uses the same values for all variables.

The range of endpoints must cover the range of the data. For example, if you specify

```
endpoints=2 to 10 by 2
```

then all of the observations must fall in the intervals [2,4) [4,6) [6,8) [8,10]. You also must use evenly spaced endpoints which you list in increasing order.

KEY	determines the endpoints for the data in the key cell. The initial number of endpoints is based on the number of observations in the key cell using the method of Terrell and Scott (1985). The procedure extends the endpoint list for the key cell in either direction as necessary until it spans the data in the remaining cells.
UNIFORM	determines the endpoints by using all the observations as if there were no cells. In other words, the number of endpoints is based on the total sample size by using the method of Terrell and Scott (1985).

Neither KEY nor UNIFORM apply unless you use the CLASS statement.

If you omit ENDPOINTS, the procedure uses the midpoints. If you specify ENDPOINTS, the procedure computes the endpoints by using an algorithm (Terrell and Scott 1985) that is primarily applicable to continuous data that are approximately normally distributed.

If you specify both MIDPOINTS= and ENDPOINTS, the procedure issues a warning message and uses the endpoints.

If you specify RTINCLUDE, the procedure includes the right endpoint of each histogram interval in that interval instead of including the left endpoint.

If you use a CLASS statement and specify ENDPOINTS, the procedure uses ENDPOINTS=KEY as the default. However if the key cell is empty, then the procedure uses ENDPOINTS=UNIFORM.

EXPONENTIAL <**(***exponential-options***)**>
EXP <**(***exponential-options***)**>

displays a fitted exponential density curve on the histogram. The EXPONENTIAL option can occur only once in a HISTOGRAM statement. The parameter θ must be less than or equal to the minimum data value. Use the THETA= *exponential-option* to specify θ. By default, THETA=0. You can specify THETA=EST to request the maximum likelihood estimate for θ. Use the SIGMA= *exponential-option* to specify σ. By default, the procedure computes a maximum likelihood estimate for σ. Table 3.3 (page 214) and Table 3.5 (page 215) list options you can specify with the EXPONENTIAL option.

FILL

fills areas under the fitted density curve or the kernel density estimate with colors and patterns. The FILL option can occur with only one fitted curve. Enclose the FILL option in parentheses after a density curve option or the KERNEL option. The CFILL= and PFILL= options specify the color and pattern for the area under the curve. For a list of available colors and patterns, see *SAS/GRAPH Reference*.

FONT=*font*

specifies a software font for reference line and axis labels. You can also specify fonts for axis labels in an AXIS statement. The FONT= font takes precedence over the FTEXT= font specified in the GOPTIONS statement. Hardware characters are used by default.

FORCEHIST

forces the creation of a histogram if there is only one unique observation. By default, a histogram is not created if the standard deviation of the data is zero.

FRONTREF

draws reference lines requested with the HREF= and VREF= options in front of the histogram bars. By default, reference lines are drawn behind the histogram bars and can be obscured by them.

GAMMA $<$**(***gamma-options***)**$>$

displays a fitted gamma density curve on the histogram. The GAMMA option can occur only once in a HISTOGRAM statement. The parameter θ must be less than the minimum data value. Use the THETA= *gamma-option* to specify θ. By default, THETA=0. You can specify THETA=EST to request the maximum likelihood estimate for θ. Use the ALPHA= and the SIGMA= *gamma-options* to specify the shape parameter α and the scale parameter σ. By default, PROC UNIVARIATE computes maximum likelihood estimates for α and σ. The procedure calculates the maximum likelihood estimate of α iteratively using the Newton-Raphson approximation. Table 3.3 (page 214) and Table 3.6 (page 215) list options you can specify with the GAMMA option. See Example 3.22.

GRID

displays a grid on the histogram. Grid lines are horizontal lines that are positioned at major tick marks on the vertical axis.

HEIGHT=*value*

specifies the height, in percentage screen units, of text for axis labels, tick mark labels, and legends. This option takes precedence over the HTEXT= option in the GOPTIONS statement.

HMINOR=*n*
HM=*n*

specifies the number of minor tick marks between each major tick mark on the horizontal axis. Minor tick marks are not labeled. By default, HMINOR=0.

HOFFSET=*value*

specifies the offset, in percentage screen units, at both ends of the horizontal axis. You can use HOFFSET=0 to eliminate the default offset.

HREF=*values*

draws reference lines that are perpendicular to the horizontal axis at the values that you specify. If a reference line is almost completely obscured, then use the FRONTREF option to draw the reference lines in front of the histogram bars. Also see the CHREF=, HREFCHAR=, and LHREF= options.

HREFLABELS='*label1***' . . . '** *labeln'*
HREFLABEL='*label1***' . . . '** *labeln'*
HREFLAB='*label1***' . . . '** *labeln'*

specifies labels for the lines requested by the HREF= option. The number of labels must equal the number of lines. Enclose each label in quotes. Labels can have up to 16 characters.

HREFLABPOS=1 | 2 | 3

specifies the vertical position of HREFLABELS= labels. If you specify HREFLABPOS=1, the labels are positioned along the top of the histogram. If you specify HREFLABPOS=2, the labels are staggered from top to bottom of the histogram. If you specify HREFLABPOS=3, the labels are positioned along the bottom of the histogram. By default, HREFLABPOS=1.

INFONT=*font*

specifies a software font to use for text inside the framed areas of the histogram. The INFONT= option takes precedence over the FTEXT= option in the GOPTIONS statement. For a list of fonts, see *SAS/GRAPH Reference*.

INHEIGHT=*value*

specifies the height, in percentage screen units, of text used inside the framed areas of the histogram. By default, the height specified by the HEIGHT= option is used. If you do not specify the HEIGHT= option, the height specified with the HTEXT= option in the GOPTIONS statement is used.

INTERTILE=*value*

specifies the distance, in horizontal percentage screen units, between the framed areas, which are called *tiles*. By default, INTERTILE=0.75 percentage screen units. This option is not available unless you use the CLASS statement. You can specify INTERTILE=0 to create contiguous tiles.

K=NORMAL | QUADRATIC | TRIANGULAR

specifies the kernel function (normal, quadratic, or triangular) used to compute a kernel density estimate. You can specify up to five values to request multiple estimates. You must enclose this option in parentheses after the KERNEL option. You can also use the K= *kernel-option* with the C= *kernel-option*, which specifies standardized bandwidths. If you specify more kernel functions than bandwidths, the procedure repeats the last bandwidth in the list for the remaining estimates. Likewise, if you specify more bandwidths than kernel functions, the procedure repeats the last kernel function for the remaining estimates. By default, K=NORMAL.

KERNEL<**(***kernel-options***)**>

superimposes up to five kernel density estimates on the histogram. By default, the procedure uses the AMISE method to compute kernel density estimates. To request multiple kernel density estimates on the same histogram, specify a list of values for either the C= *kernel-option* or K= *kernel-option*. Table 3.10 (page 215) lists options you can specify with the KERNEL option. See Example 3.23.

L=*linetype*

specifies the line type used for fitted density curves. Enclose the L= option in parentheses after the distribution option or the KERNEL option. If you use the L= option with the KERNEL option, you can specify a list of up to five line types for multiple kernel density estimates. See the entries for the C= and K= options for details on specifying multiple kernel density estimates. By default, L=1, which produces a solid line.

LGRID=*linetype*

specifies the line type for the grid when a grid displays on the histogram. By default, LGRID=1, which produces a solid line. This option also creates a grid.

LHREF=*linetype*
LH=*linetype*

specifies the line type for the reference lines that you request with the HREF= option. By default, LHREF=2, which produces a dashed line.

LOGNORMAL<**(***lognormal-options***)**>

displays a fitted lognormal density curve on the histogram. The LOGNORMAL option can occur only once in a HISTOGRAM statement. The parameter θ must be less than the minimum data value. Use the THETA= *lognormal-option* to specify θ. By default, THETA=0. You can specify THETA=EST to request the maximum likelihood estimate for θ. Use the SIGMA= and ZETA= *lognormal-options* to specify σ and ζ. By default, the procedure computes maximum likelihood estimates for σ and ζ. Table 3.3 (page 214) and Table 3.7 (page 215) list options you can specify with the LOGNORMAL option. See Example 3.22 and Example 3.24.

LOWER=*value-list*

specifies lower bounds for kernel density estimates requested with the KERNEL option. Enclose the LOWER= option in parentheses after the KERNEL option. You can specify up to five lower bounds for multiple kernel density estimates. If you specify more kernel estimates than lower bounds, the last lower bound is repeated for the remaining estimates. The default is a missing value, indicating no lower bounds for fitted kernel density curves.

LVREF=*linetype*
LV=*linetype*

specifies the line type for lines requested with the VREF= option. By default, LVREF=2, which produces a dashed line.

MAXNBIN=*n*

specifies the maximum number of bins displayed in the comparative histogram. This option is useful when the scales or ranges of the data distributions differ greatly from cell to cell. By default, the bin size and midpoints are determined for the key cell, and then the midpoint list is extended to accommodate the data ranges for the remaining cells. However, if the cell scales differ considerably, the resulting number of bins may be so great that each cell histogram is scaled into a narrow region. By using MAXNBIN= to limit the number of bins, you can narrow the window about the data distribution in the key cell. This option is not available unless you specify the CLASS statement. The MAXNBIN= option is an alternative to the MAXSIGMAS= option.

MAXSIGMAS=*value*

limits the number of bins displayed in the comparative histogram to a range of *value* standard deviations (of the data in the key cell) above and below the mean of the data in the key cell. This option is useful when the scales or ranges of the data distributions differ greatly from cell to cell. By default, the bin size and midpoints are determined for the key cell, and then the midpoint list is extended to accommodate the data ranges for the remaining cells. However, if the cell scales differ considerably, the resulting number of bins may be so great that each cell histogram is scaled into a narrow region. By using MAXSIGMAS= to limit the number of bins, you can narrow the window that surrounds the data distribution in the key cell. This option is not available unless you specify the CLASS statement.

MIDPERCENTS

requests a table listing the midpoints and percentage of observations in each histogram interval. If you specify MIDPERCENTS in parentheses after a density estimate option, the procedure displays a table that lists the midpoints, the observed percentage of observations, and the estimated percentage of the population in each interval (estimated from the fitted distribution). See Example 3.18.

MIDPOINTS=*values* | KEY | UNIFORM

specifies how to determine the midpoints for the histogram intervals, where *values* determines the width of the histogram bars as the difference between consecutive midpoints. The procedure uses the same values for all variables.

The range of midpoints, extended at each end by half of the bar width, must cover the range of the data. For example, if you specify

```
midpoints=2 to 10 by 0.5
```

then all of the observations should fall between 1.75 and 10.25. You must use evenly spaced midpoints listed in increasing order.

KEY	determines the midpoints for the data in the key cell. The initial number of midpoints is based on the number of observations in the key cell that use the method of Terrell and Scott (1985). The procedure extends the midpoint list for the key cell in either direction as necessary until it spans the data in the remaining cells.
UNIFORM	determines the midpoints by using all the observations as if there were no cells. In other words, the number of midpoints is based on the total sample size by using the method of Terrell and Scott (1985).

Neither KEY nor UNIFORM apply unless you use the CLASS statement. By default, if you use a CLASS statement, MIDPOINTS=KEY; however, if the key cell is empty then MIDPOINTS=UNIFORM. Otherwise, the procedure computes the midpoints by using an algorithm (Terrell and Scott 1985) that is primarily applicable to continuous data that are approximately normally distributed.

MU=*value*
specifies the parameter μ for normal density curves requested with the NORMAL option. Enclose the MU= option in parentheses after the NORMAL option. By default, the procedure uses the sample mean for μ.

NAME='*string***'**
specifies a name for the plot, up to eight characters long, that appears in the PROC GREPLAY master menu. The default value is 'UNIVAR'.

NCOLS=*n*
NCOL=*n*
specifies the number of columns in a comparative histogram. By default, NCOLS=1 if you specify only one class variable, and NCOLS=2 if you specify two class variables. This option is not available unless you use the CLASS statement. If you specify two class variables, you can use the NCOLS= option with the NROWS= option.

NOBARS
suppresses drawing of histogram bars, which is useful for viewing fitted curves only.

NOFRAME
suppresses the frame around the subplot area.

NOHLABEL
suppresses the label for the horizontal axis. You can use this option to reduce clutter.

NOPLOT
NOCHART
suppresses the creation of a plot. Use this option when you only want to tabulate summary statistics for a fitted density or create an OUTHISTOGRAM= data set.

NOPRINT
suppresses tables summarizing the fitted curve. Enclose the NOPRINT option in parentheses following the distribution option.

NORMAL<**(***normal-options***)**>
displays a fitted normal density curve on the histogram. The NORMAL option can occur only once in a HISTOGRAM statement. Use the MU= and SIGMA= *normal-options* to specify μ and σ. By default, the procedure uses the sample mean and sample standard deviation for μ and σ. Table 3.3 (page 214) and Table 3.8 (page 215) list options you can specify with the NORMAL option. See Example 3.19.

NOVLABEL
suppresses the label for the vertical axis. You can use this option to reduce clutter.

NOVTICK
suppresses the tick marks and tick mark labels for the vertical axis. This option also suppresses the label for the vertical axis.

NROWS=*n*
NROW=*n*
specifies the number of rows in a comparative histogram. By default, NROWS=2. This option is not available unless you use the CLASS statement. If you specify two class variables, you can use the NCOLS= option with the NROWS= option.

OUTHISTOGRAM=*SAS-data-set*
OUTHIST=*SAS-data-set*

creates a SAS data set that contains information about histogram intervals. Specifically, the data set contains the midpoints of the histogram intervals, the observed percentage of observations in each interval, and the estimated percentage of observations in each interval (estimated from each of the specified fitted curves).

PERCENTS=*values*
PERCENT=*values*

specifies a list of percents for which quantiles calculated from the data and quantiles estimated from the fitted curve are tabulated. The percents must be between 0 and 100. Enclose the PERCENTS= option in parentheses after the curve option. The default percents are 1, 5, 10, 25, 50, 75, 90, 95, and 99.

PFILL=*pattern*

specifies a pattern used to fill the bars of the histograms (or the areas under a fitted curve if you also specify the FILL option). See the entries for the CFILL= and FILL options for additional details. Refer to *SAS/GRAPH Software: Reference* for a list of pattern values. By default, the bars and curve areas are not filled.

RTINCLUDE

includes the right endpoint of each histogram interval in that interval. By default, the left endpoint is included in the histogram interval.

SCALE=*value*

is an alias for the SIGMA= option for curves requested by the BETA, EXPONENTIAL, GAMMA, and WEIBULL options and an alias for the ZETA= option for curves requested by the LOGNORMAL option.

SHAPE=*value*

is an alias for the ALPHA= option for curves requested with the GAMMA option, an alias for the SIGMA= option for curves requested with the LOGNORMAL option, and an alias for the C= option for curves requested with the WEIBULL option.

SIGMA=*value* **| EST**

specifies the parameter σ for the fitted density curve when you request the BETA, EXPONENTIAL, GAMMA, LOGNORMAL, NORMAL, and WEIBULL options.

See Table 3.13 for a summary of how to use the SIGMA= option. You must enclose this option in parentheses after the density curve option. As a *beta-option*, you can specify SIGMA=EST to request a maximum likelihood estimate for σ.

Table 3.13. Uses of the SIGMA= Option

Distribution Keyword	SIGMA= Specifies	Default Value	Alias
BETA	Scale parameter σ	1	SCALE=
EXPONENTIAL	Scale parameter σ	Maximum likelihood estimate	SCALE=
GAMMA	Scale parameter σ	Maximum likelihood estimate	SCALE=
WEIBULL	Scale parameter σ	Maximum likelihood estimate	SCALE=
LOGNORMAL	Shape parameter σ	Maximum likelihood estimate	SCALE=
NORMAL	Scale parameter σ	Standard deviation	SHAPE=
WEIBULL	Scale parameter σ	Maximum likelihood estimate	SCALE=

THETA=*value* **| EST**

specifies the lower threshold parameter θ for curves requested with the BETA, EXPONENTIAL, GAMMA, LOGNORMAL, and WEIBULL options. Enclose the THETA= option in parentheses after the curve option. By default, THETA=0. If you specify THETA=EST, an estimate is computed for θ.

THRESHOLD= *value*

is an alias for the THETA= option. See the preceding entry for the THETA= option.

TURNVLABELS
TURNVLABEL

turns the characters in the vertical axis labels so that they display vertically. This happens by default when you use a hardware font.

UPPER=*value-list*

specifies upper bounds for kernel density estimates requested with the KERNEL option. Enclose the UPPER= option in parentheses after the KERNEL option. You can specify up to five upper bounds for multiple kernel density estimates. If you specify more kernel estimates than upper bounds, the last upper bound is repeated for the remaining estimates. The default is a missing value, indicating no upper bounds for fitted kernel density curves.

VAXIS=*name***|***value-list*

specifies the name of an AXIS statement describing the vertical axis. Alternatively, you can specify a *value-list* for the vertical axis.

VAXISLABEL=*'label'*

specifies a label for the vertical axis. Labels can have up to 40 characters.

VMINOR=*n*
VM=*n*

specifies the number of minor tick marks between each major tick mark on the vertical axis. Minor tick marks are not labeled. The default is zero.

VOFFSET=*value*

specifies the offset, in percentage screen units, at the upper end of the vertical axis.

VREF=*value-list*

draws reference lines perpendicular to the vertical axis at the values specified. Also see the CVREF=, LVREF=, and VREFCHAR= options. If a reference line is almost completely obscured, then use the FRONTREF option to draw the reference lines in front of the histogram bars.

VREFLABELS='*label1***'**. . . **'***labeln***'**
VREFLABEL='*label1***'**. . . **'***labeln***'**
VREFLAB='*label1***'**. . . **'***labeln***'**

specifies labels for the lines requested by the VREF= option. The number of labels must equal the number of lines. Enclose each label in quotes. Labels can have up to 16 characters.

VREFLABPOS=*n*

specifies the horizontal position of VREFLABELS= labels. If you specify VREFLABPOS=1, the labels are positioned at the left of the histogram. If you specify VREFLABPOS=2, the labels are positioned at the right of the histogram. By default, VREFLABPOS=1.

VSCALE=COUNT | PERCENT | PROPORTION

specifies the scale of the vertical axis for a histogram. The value COUNT requests the data be scaled in units of the number of observations per data unit. The value PERCENT requests the data be scaled in units of percent of observations per data unit. The value PROPORTION requests the data be scaled in units of proportion of observations per data unit. The default is PERCENT.

W=*n*

specifies the width, in pixels, of the fitted density curve or the kernel density estimate curve. By default, W=1. You must enclose this option in parentheses after the density curve option or the KERNEL option. As a *kernel-option*, you can specify a list of up to five W= values.

WAXIS=*n*

specifies the line thickness, in pixels, for the axes and frame. By default, WAXIS=1.

WBARLINE=*n*

specifies the width of bar outlines. By default, WBARLINE=1.

WEIBULL<**(***Weibull-options***)**>

displays a fitted Weibull density curve on the histogram. The WEIBULL option can occur only once in a HISTOGRAM statement. The parameter θ must be less than the minimum data value. Use the THETA= *Weibull-option* to specify θ. By default, THETA=0. You can specify THETA=EST to request the maximum likelihood estimate for θ. Use the C= and SIGMA= *Weibull-options* to specify the shape parameter c and the scale parameter σ. By default, the procedure computes the maximum likelihood estimates for c and σ. Table 3.3 (page 214) and Table 3.9 (page 215) list option you can specify with the WEIBULL option. See Example 3.22.

PROC UNIVARIATE calculates the maximum likelihood estimate of a iteratively by using the Newton-Raphson approximation. See also the C=, SIGMA=, and THETA= *Weibull-options*.

WGRID=*n*

specifies the line thickness for the grid.

ZETA= *value*

specifies a value for the scale parameter ζ for lognormal density curves requested with the LOGNORMAL option. Enclose the ZETA= *lognormal-option* in parentheses after the LOGNORMAL option. By default, the procedure calculates a maximum likelihood estimate for ζ. You can specify the SCALE= option as an alias for the ZETA= option.

ID Statement

ID *variables* **;**

The ID statement specifies one or more variables to include in the table of extreme observations. The corresponding values of the ID variables appear beside the n largest and n smallest observations, where n is the value of NEXTROBS= option. See Example 3.3.

INSET Statement

INSET *keyword-list* $<$ */ options* $>$ **;**

The INSET statement places a box or table of summary statistics, called an *inset*, directly in a high-resolution graph created with the HISTOGRAM, PROBPLOT, or QQPLOT statement.

The INSET statement must follow the HISTOGRAM, PROBPLOT, or QQPLOT statement that creates the plot that you want to augment. The inset appears in all the graphs that the preceding plot statement produces.

You can use multiple INSET statements after a plot statement to add multiple insets to a plot. See Example 3.17.

In an INSET statement, you specify one or more *keywords* that identify the information to display in the inset. The information is displayed in the order that you request the *keywords*. *Keywords* can be any of the following:

- *statistical keywords*
- *primary keywords*
- *secondary keywords*

The available *statistical keywords* are:

Table 3.14. Descriptive Statistic Keywords

CSS	Corrected sum of squares
CV	Coefficient of variation
KURTOSIS	Kurtosis
MAX	Largest value
MEAN	Sample mean
MIN	Smallest value
MODE	Most frequent value
N	Sample size
NMISS	Number of missing values
NOBS	Number of observations
RANGE	Range
SKEWNESS	Skewness
STD	Standard deviation
STDMEAN	Standard error of the mean
SUM	Sum of the observations
SUMWGT	Sum of the weights
USS	Uncorrected sum of squares
VAR	Variance

Table 3.15. Percentile Statistic Keywords

P1	1st percentile
P5	5th percentile
P10	10th percentile
Q1	Lower quartile (25th percentile)
MEDIAN	Median (50th percentile)
Q3	Upper quartile (75th percentile)
P90	90th percentile
P95	95th percentile
P99	99th percentile
QRANGE	Interquartile range (Q3 - Q1)

Table 3.16. Robust Statistics Keywords

GINI	Gini's mean difference
MAD	Median absolute difference about the median
QN	Q_n, alternative to MAD
SN	S_n, alternative to MAD
STD_GINI	Gini's standard deviation
STD_MAD	MAD standard deviation
STD_QN	Q_n standard deviation
STD_QRANGE	Interquartile range standard deviation
STD_SN	S_n standard deviation

Table 3.17. Hypothesis Testing Keywords

MSIGN	Sign statistic
NORMALTEST	Test statistic for normality
PNORMAL	Probability value for the test of normality
SIGNRANK	Signed rank statistic
PROBM	Probability of greater absolute value for the sign statistic
PROBN	Probability value for the test of normality
PROBS	Probability value for the signed rank test
PROBT	Probability value for the Student's t test
T	Statistics for Student's t test

A *primary keyword* enables you to specify *secondary keywords* in parentheses immediately after the primary keyword. *Primary keywords* are BETA, EXPONENTIAL, GAMMA, LOGNORMAL, NORMAL, WEIBULL, WEIBULL2, KERNEL, and KERNELn. If you specify a *primary keyword* but omit a *secondary keyword*, the inset displays a colored line and the distribution name as a key for the density curve.

By default, PROC UNIVARIATE identifies inset statistics with appropriate labels and prints numeric values using appropriate formats. To customize the label, specify the *keyword* followed by an equal sign (=) and the desired label in quotes. To customize the format, specify a numeric format in parentheses after the *keyword*. Labels can have up to 24 characters.

If you specify both a label and a format for a statistic, the label must appear before the format. For example,

```
inset n='Sample Size' std='Std Dev' (5.2);
```

requests customized labels for two statistics and displays the standard deviation with a field width of 5 and two decimal places.

The following tables list *primary keywords*:

Table 3.18. Parametric Density Primary Keywords

Keyword	Distribution	Plot Statement Availability
BETA	Beta	All plot statements
EXPONENTIAL	Exponential	All plot statements
GAMMA	Gamma	All plot statements
LOGNORMAL	Lognormal	All plot statements
NORMAL	Normal	All plot statements
WEIBULL	Weibull(3-parameter)	All plot statements
WEIBULL2	Weibull(2-parameter)	PROBPLOT and QQPLOT

Table 3.19. Kernel Density Estimate Primary Keywords

Keyword	Description
KERNEL	Displays statistics for all kernel estimates
KERNELn	Displays statistics for only the nth kernel density estimate $n = 1, 2, 3, 4$, or 5

Table 3.20 through Table 3.28 list the *secondary keywords* available with *primary keywords* in Table 3.18 and Table 3.19.

Table 3.20. Secondary Keywords Available with the BETA Keyword

Secondary Keyword	Alias	Description
ALPHA	SHAPE1	First shape parameter α
BETA	SHAPE2	Second shape parameter β
SIGMA	SCALE	Scale parameter σ
THETA	THRESHOLD	Lower threshold parameter θ
MEAN		Mean of the fitted distribution
STD		Standard deviation of the fitted distribution

Table 3.21. Secondary Keywords Available with the EXP Keyword

Secondary Keyword	Alias	Description
SIGMA	SCALE	Scale parameter σ
THETA	THRESHOLD	Threshold parameter θ
MEAN		Mean of the fitted distribution
STD		Standard deviation of the fitted distribution

Table 3.22. Secondary Keywords Available with the GAMMA Keyword

Secondary Keyword	Alias	Description
ALPHA	SHAPE	Shape parameter α
SIGMA	SCALE	Scale parameter σ
THETA	THRESHOLD	Threshold parameter θ
MEAN		Mean of the fitted distribution
STD		Standard deviation of the fitted distribution

Table 3.23. Secondary Keywords Available with the LOGNORMAL Keyword

Secondary Keyword	Alias	Description
SIGMA	SHAPE	Shape parameter σ
THETA	THRESHOLD	Threshold parameter θ
ZETA	SCALE	Scale parameter ζ
MEAN		Mean of the fitted distribution
STD		Standard deviation of the fitted distribution

Table 3.24. Secondary Keywords Available with the NORMAL Keyword

Secondary Keyword	Alias	Description
MU	MEAN	Mean parameter μ
SIGMA	STD	Scale parameter σ

Table 3.25. Secondary Keywords Available with the WEIBULL Keyword

Secondary Keyword	Alias	Description
C	SHAPE	Shape parameter c
SIGMA	SCALE	Scale parameter σ
THETA	THRESHOLD	Threshold parameter θ
MEAN		Mean of the fitted distribution
STD		Standard deviation of the fitted distribution

Table 3.26. Secondary Keywords Available with the WEIBULL2 Keyword

Secondary Keyword	Alias	Description
C	SHAPE	Shape parameter c
SIGMA	SCALE	Scale parameter σ
THETA	THRESHOLD	Known lower threshold θ_0
MEAN		Mean of the fitted distribution
STD		Standard deviation of the fitted distribution

Table 3.27. Secondary Keywords Available with the KERNEL Keyword

Secondary Keyword	Description
TYPE	Kernel type: normal, quadratic, or triangular
BANDWIDTH	Bandwidth λ for the density estimate
BWIDTH	Alias for BANDWIDTH
C	Standardized bandwidth c for the density estimate: $c = \frac{\lambda}{Q} n^{\frac{1}{5}}$ where $n =$ sample size, $\lambda =$ bandwidth, and $Q =$ interquartile range
AMISE	Approximate mean integrated square error (MISE) for the kernel density

Table 3.28. Goodness-of-Fit Statistics for Fitted Curves

Secondary Keyword	Description
AD	Anderson-Darling EDF test statistic
ADPVAL	Anderson-Darling EDF test p-value
CVM	Cramér-von Mises EDF test statistic
CVMPVAL	Cramér-von Mises EDF test p-value
KSD	Kolmogorov-Smirnov EDF test statistic
KSDPVAL	Kolmogorov-Smirnov EDF test p-value

The inset statistics listed in Table 3.18 through Table 3.28 are not available unless you request a plot statement and options that calculate these statistics. For example,

```
proc univariate data=score;
   histogram final / normal;
   inset mean std normal(ad adpval);
run;
```

The MEAN and STD *keywords* display the sample mean and standard deviation of FINAL. The NORMAL *keyword* with the *secondary keywords* AD and ADPVAL display the Anderson-Darling goodness-of-fit test statistic and p-value. The statistics that are specified with the NORMAL *keyword* are available only because the NORMAL option is requested in the HISTOGRAM statement.

The KERNEL or KERNELn *keyword* is available only if you request a kernel density estimate in a HISTOGRAM statement. The WEIBULL2 *keyword* is available only if you request a two-parameter Weibull distribution in the PROBPLOT or QQPLOT statement.

Summary of Options

The following table lists INSET statement *options*, which are specified after the slash (/) in the INSET statement. For complete descriptions, see the section "Dictionary of Options" on page 235.

Table 3.29. INSET Options

CFILL=*color* | BLANK	Specifies color of inset background
CFILLH=*color*	Specifies color of header background
CFRAME=*color*	Specifies color of frame
CHEADER=*color*	Specifies color of header text
CSHADOW=*color*	Specifies color of drop shadow
CTEXT=*color*	Specifies color of inset text
DATA	Specifies data units for POSITION=(x, y) coordinates
DATA=*SAS-data-set*	Specifies data set for statistics in the inset table
FONT=*font*	Specifies font of text
FORMAT=*format*	Specifies format of values in inset
HEADER=*'quoted string'*	Specifies header text
HEIGHT=*value*	Specifies height of inset text
NOFRAME	Suppresses frame around inset

Table 3.29. (continued)

POSITION=*position*	Specifies position of inset
REFPOINT=BR | BL | TR | TL	Specifies reference point of inset positioned with POSITION=(x, y) coordinates

Dictionary of Options

The following entries provide detailed descriptions of options for the INSET statement.

To specify the same format for all the statistics in the INSET statement, use the FORMAT= option.

To create a completely customized inset, use a DATA= data set. The data set contains the label and the value that you want to display in the inset.

If you specify multiple kernel density estimates, you can request inset statistics for all the estimates with the KERNEL *keyword*. Alternatively, you can display inset statistics for individual curves with the KERNELn *keyword*, where n is the curve number between 1 and 5.

CFILL=*color* | BLANK
specifies the color of the background. If you omit the CFILLH= option the header background is included. By default, the background is empty, which causes items that overlap the inset (such as curves or histogram bars) to show through the inset.

If you specify a value for CFILL= option, then overlapping items no longer show through the inset. Use CFILL=BLANK to leave the background uncolored and to prevent items from showing through the inset.

CFILLH=*color*
specifies the color of the header background. The default value is the CFILL= color.

CFRAME=*color*
specifies the color of the frame. The default value is the same color as the axis of the plot.

CHEADER=*color*
specifies the color of the header text. The default value is the CTEXT= color.

CSHADOW=*color*
specifies the color of the drop shadow. By default, if a CSHADOW= option is not specified, a drop shadow is not displayed.

CTEXT=*color*
specifies the color of the text. The default value is the same color as the other text on the plot.

DATA
specifies that data coordinates are to be used in positioning the inset with the POSITION= option. The DATA option is available only when you specify POSITION=(x,y). You must place DATA immediately after the coordinates (x,y).

DATA=*SAS-data-set*

requests that PROC UNIVARIATE display customized statistics from a SAS data set in the inset table. The data set must contain two variables:

LABEL	a character variable whose values provide labels for inset entries.
VALUE	a variable that is either character or numeric and whose values provide values for inset entries.

The label and value from each observation in the data set occupy one line in the inset. The position of the DATA= keyword in the keyword list determines the position of its lines in the inset.

FONT=*font*

specifies the font of the text. By default, if you locate the inset in the interior of the plot then the font is SIMPLEX. If you locate the inset in the exterior of the plot then the font is the same as the other text on the plot.

FORMAT=*format*

specifies a format for all the values in the inset. If you specify a format for a particular statistic, then this format overrides FORMAT= format. For more information about SAS formats, see *SAS Language Reference: Dictionary*

HEADER=*string*

specifies the header text. The *string* cannot exceed 40 characters. By default, no header line appears in the inset. If all the keywords that you list in the INSET statement are secondary keywords that correspond to a fitted curve on a histogram, PROC UNIVARIATE displays a default header that indicates the distribution and identifies the curve.

HEIGHT=*value*

specifies the height of the text.

NOFRAME

suppresses the frame drawn around the text.

POSITION=*position*

POS=*position*

determines the position of the inset. The position is a compass point keyword, a margin keyword, or a pair of coordinates (x,y). You can specify coordinates in axis percent units or axis data units. The default value is NW, which positions the inset in the upper left (northwest) corner of the display. See the section "Positioning the Inset" on page 285.

REFPOINT=BR | BL | TR | TL

specifies the reference point for an inset that PROC UNIVARIATE positions by a pair of coordinates with the POSITION= option. The REFPOINT= option specifies which corner of the inset frame that you want to position at coordinates (x,y). The *keywords* are BL, BR, TL, and TR, which correspond to bottom left, bottom right, top left, and top right. The default value is BL. You must use REFPOINT= with POSITION=(x,y) coordinates.

OUTPUT Statement

OUTPUT < **OUT=***SAS-data-set* >
< *keyword1=names...keywordk=names* >< *percentile-options* >**;**

The OUTPUT statement saves statistics and BY variables in an output data set. When you use a BY statement, each observation in the OUT= data set corresponds to one of the BY groups. Otherwise, the OUT= data set contains only one observation.

You can use any number of OUTPUT statements in the UNIVARIATE procedure. Each OUTPUT statement creates a new data set containing the statistics specified in that statement. You must use the VAR statement with the OUTPUT statement. The OUTPUT statement must contain a specification of the form *keyword=names* or the PCTLPTS= and PCTLPRE= specifications. See Example 3.7 and Example 3.8.

OUT=*SAS-data-set*

identifies the output data set. If *SAS-data-set* does not exist, PROC UNIVARIATE creates it. If you omit OUT=, the data set is named DATA*n*, where *n* is the smallest integer that makes the name unique. The default *SAS-data-set* is DATA*n*.

*keyword***=***name*

specifies the statistics to include in the output data set and gives names to the new variables that contain the statistics. Specify a *keyword* for each desired statistic, an equal sign, and the *names* of the variables to contain the statistic. In the output data set, the first variable listed after a keyword in the OUTPUT statement contains the statistic for the first variable listed in the VAR statement; the second variable contains the statistic for the second variable in the VAR statement, and so on. If the list of *names* following the equal sign is shorter than the list of variables in the VAR statement, the procedure uses the *names* in the order in which the variables are listed in the VAR statement. The available keywords are listed in the following tables:

Table 3.30. Descriptive Statistic Keywords

CSS	Corrected sum of squares
CV	Coefficient of variation
KURTOSIS	Kurtosis
MAX	Largest value
MEAN	Sample mean
MIN	Smallest value
MODE	Most frequent value
N	Sample size
NMISS	Number of missing values
NOBS	Number of observations
RANGE	Range
SKEWNESS	Skewness
STD	Standard deviation
STDMEAN	Standard error of the mean
SUM	Sum of the observations
SUMWGT	Sum of the weights
USS	Uncorrected sum of squares
VAR	Variance

Table 3.31. Quantile Statistic Keywords

P1	1st percentile
P5	5th percentile
P10	10th percentile
Q1	Lower quartile (25th percentile)
MEDIAN	Median (50th percentile)
Q3	Upper quartile (75th percentile)
P90	90th percentile
P95	95th percentile
P99	99th percentile
QRANGE	Interquartile range (Q3 - Q1)

Table 3.32. Robust Statistics Keywords

GINI	Gini's mean difference
MAD	Median absolute difference about the median
QN	Q_n, alternative to MAD
SN	S_n, alternative to MAD
STD_GINI	Gini's standard deviation
STD_MAD	MAD standard deviation
STD_QN	Q_n standard deviation
STD_QRANGE	Interquartile range standard deviation
STD_SN	S_n standard deviation

Table 3.33. Hypothesis Testing Keywords

MSIGN	Sign statistic
NORMALTEST	Test statistic for normality
SIGNRANK	Signed rank statistic
PROBM	Probability of a greater absolute value for the sign statistic
PROBN	Probability value for the test of normality
PROBS	Probability value for the signed rank test
PROBT	Probability value for the Student's t test
T	Statistic for the Student's t test

To store the same statistic for several analysis variables, specify a list of *names*. The order of the names corresponds to the order of the analysis variables in the VAR statement. PROC UNIVARIATE uses the first name to create a variable that contains the statistic for the first analysis variable, the next name to create a variable that contains the statistic for the second analysis variable, and so on. If you do not want to output statistics for all the analysis variables, specify fewer names than the number of analysis variables.

The UNIVARIATE procedure automatically computes the 1st, 5th, 10th, 25th, 50th, 75th, 90th, 95th, and 99th percentiles for the data. These can be saved in an output data set using *keyword=names* specifications. For additional percentiles, you can use the following *percentile-options*:

PCTLPTS=*percentiles*

specifies one or more percentiles that are not automatically computed by the UNIVARIATE procedure. The PCTLPRE= and PCTLPTS= options must be used together. You can specify percentiles with the expression start TO stop BY increment where start is a starting number, stop is an ending number, and increment is a number to increment by. The PCTLPTS= option generates additional percentiles and outputs them to a data set; these additional percentiles are not printed.

To compute the 50th, 95th, 97.5th, and 100th percentiles, submit the statement

```
output pctlpre=P_ pctlpts=50,95 to 100 by 2.5;
```

You can use PCTLPTS= to output percentiles that are not in the list of quantile statistics. PROC UNIVARIATE computes the requested percentiles based on the method that you specify with the PCTLDEF= option in the PROC UNIVARIATE statement. You must use PCTLPRE=, and optionally PCTLNAME=, to specify variable names for the percentiles. For example, the following statements create an output data set that is named Pctls that contains the 20th and 40th percentiles of the analysis variables PreTest and PostTest:

```
proc univariate data=Score;
   var PreTest PostTest;
   output out=Pctls pctlpts=20 40 pctlpre=PreTest_ PostTest_
               pctlname=P20 P40;
run;
```

PROC UNIVARIATE saves the 20th and 40th percentiles for PreTest and PostTest in the variables PreTest_P20, PostTest_P20, PreTest_P40, and PostTest_P40.

PCTLPRE=*prefixes*

specifies one or more prefixes to create the variable names for the variables that contain the PCTLPTS= percentiles. To save the same percentiles for more than one analysis variable, specify a list of prefixes. The order of the prefixes corresponds to the order of the analysis variables in the VAR statement. The PCTLPRE= and PCTLPTS= options must be used together.

The procedure generates new variable names using the *prefix* and the percentile values. If the specified percentile is an integer, the variable name is simply the *prefix* followed by the value. If the specified value is not an integer, an underscore replaces the decimal point in the variable name, and decimal values are truncated to one decimal place. For example, the following statements create the variables PWID20, PWID33_3, PWID66_6, and PWID80 for the 20th, 33.33rd, 66.67th, and 80th percentiles of Width, respectively:

```
proc univariate noprint;
   var Width;
   output pctlpts=20 33.33 66.67 80 pctlpre=pwid;
run;
```

If you request percentiles for more than one variable, you should list prefixes in the same order in which the variables appear in the VAR statement. If combining the *prefix* and percentile value results in a name longer than 32 characters, the prefix is truncated so that the variable name is 32 characters.

PCTLNAME=*suffixes*

specifies one or more suffixes to create the names for the variables that contain the PCTLPTS= percentiles. PROC UNIVARIATE creates a variable name by combining the PCTLPRE= value and suffix-name. Because the suffix names are associated with the percentiles that are requested, list the suffix names in the same order as the PCTLPTS= percentiles. If you specify n *suffixes* with the PCTLNAME= option and m percentile values with the PCTLPTS= option, where $m > n$, the *suffixes* are used to name the first n percentiles, and the default names are used for the remaining $m-n$ percentiles. For example, consider the following statements:

```
proc univariate;
   var Length Width Height;
   output pctlpts  = 20 40
          pctlpre  = pl pw ph
          pctlname = twenty;
run;
```

The value TWENTY in the PCTLNAME= option is used for only the first percentile in the PCTLPTS= list. This suffix is appended to the values in the PCTLPRE= option to generate the new variable names PLTWENTY, PWTWENTY, and PHTWENTY, which contain the 20th percentiles for Length, Width, and Height, respectively. Since a second PCTLNAME= suffix is not specified, variable names for the 40th percentiles for Length, Width, and Height are generated using the prefixes and percentile values. Thus, the output data set contains the variables PLTWENTY, PL40, PWTWENTY, PW40, PHTWENTY, and PH40.

You must specify PCTLPRE= to supply prefix names for the variables that contain the PCTLPTS= percentiles.

If the number of PCTLNAME= values is fewer than the number of percentiles, or if you omit PCTLNAME=, PROC UNIVARIATE uses the percentile as the suffix to create the name of the variable that contains the percentile. For an integer percentile, PROC UNIVARIATE uses the percentile. Otherwise, PROC UNIVARIATE truncates decimal values of percentiles to two decimal places and replaces the decimal point with an underscore.

If either the prefix and suffix name combination or the prefix and percentile name combination is longer than 32 characters, PROC UNIVARIATE truncates the prefix name so that the variable name is 32 characters.

PROBPLOT Statement

PROBPLOT < *variables* >< / *options* >**;**

The PROBPLOT statement creates a probability plot, which compares ordered variable values with the percentiles of a specified theoretical distribution. If the data distribution matches the theoretical distribution, the points on the plot form a linear pattern. Consequently, you can use a probability plot to determine how well a theoretical distribution models a set of measurements.

Probability plots are similar to Q-Q plots, which you can create with the QQPLOT statement. Probability plots are preferable for graphical estimation of percentiles, whereas Q-Q plots are preferable for graphical estimation of distribution parameters.

You can use any number of PROBPLOT statements in the UNIVARIATE procedure. The components of the PROBPLOT statement are described as follows.

variables

are the variables for which to create probability plots. If you specify a VAR statement, the *variables* must also be listed in the VAR statement. Otherwise, the *variables* can be any numeric variables in the input data set. If you do not specify a list of *variables*, then by default the procedure creates a probability plot for each variable listed in the VAR statement, or for each numeric variable in the DATA= data set if you do not specify a VAR statement. For example, each of the following PROBPLOT statements produces two probability plots, one for Length and one for Width:

```
proc univariate data=Measures;
   var Length Width;
   probplot;

proc univariate data=Measures;
   probplot Length Width;
run;
```

options

specify the theoretical distribution for the plot or add features to the plot. If you specify more than one variable, the *options* apply equally to each variable. Specify all *options* after the slash (/) in the PROBPLOT statement. You can specify only one *option* naming a distribution in each PROBPLOT statement, but you can specify any number of other *options*. The distributions available are the beta, exponential, gamma, lognormal, normal, two-parameter Weibull, and three-parameter Weibull. By default, the procedure produces a plot for the normal distribution.

In the following example, the NORMAL option requests a normal probability plot for each variable, while the MU= and SIGMA= *normal-options* request a distribution reference line corresponding to the normal distribution with $\mu = 10$ and $\sigma = 0.3$. The SQUARE option displays the plot in a square frame, and the CTEXT= option specifies the text color.

```
proc univariate data=Measures;
   probplot Length1 Length2 / normal(mu=10 sigma=0.3)
                              square ctext=blue;
run;
```

Table 3.34 through Table 3.43 list the PROBPLOT *options* by function. For complete descriptions, see the section "Dictionary of Options" on page 245. *Options* can be any of the following:

- *primary options*
- *secondary options*
- *general options*

Distribution Options

Table 3.34 lists *options* for requesting a theoretical distribution.

Table 3.34. Primary Options for Theoretical Distributions

BETA(*beta-options*)	Specifies beta probability plot for shape parameters α and β specified with mandatory ALPHA= and BETA= *beta-options*
EXPONENTIAL(*exponential-options*)	Specifies exponential probability plot
GAMMA(*gamma-options*)	Specifies gamma probability plot for shape parameter α specified with mandatory ALPHA= *gamma-option*
LOGNORMAL(*lognormal-options*)	Specifies lognormal probability plot for shape parameter σ specified with mandatory SIGMA= *lognormal-option*
NORMAL(*normal-options*)	Specifies normal probability plot
WEIBULL(*Weibull-options*)	Specifies three-parameter Weibull probability plot for shape parameter c specified with mandatory C= *Weibull-option*
WEIBULL2(*Weibull2-options*)	Specifies two-parameter Weibull probability plot

Table 3.35 through Table 3.42 list *secondary options* that specify distribution parameters and control the display of a distribution reference line. Specify these options in parentheses after the distribution keyword. For example, you can request a normal probability plot with a distribution reference line by specifying the NORMAL option as follows:

```
proc univariate;
   probplot Length / normal(mu=10 sigma=0.3 color=red);
run;
```

The MU= and SIGMA= *normal-options* display a distribution reference line that corresponds to the normal distribution with mean $\mu_0 = 10$ and standard deviation $\sigma_0 = 0.3$, and the COLOR= *normal-option* specifies the color for the line.

Table 3.35. Secondary Reference Line Options Used with All Distributions

COLOR=*color*	Specifies color of distribution reference line
L=*linetype*	Specifies line type of distribution reference line
W=n	Specifies width of distribution reference line

Table 3.36. Secondary Beta-Options

ALPHA=*value-list* | EST	Specifies mandatory shape parameter α
BETA=*value-list* | EST	Specifies mandatory shape parameter β
SIGMA=*value* | EST	Specifies σ_0 for distribution reference line
THETA=*value* | EST	Specifies θ_0 for distribution reference line

Table 3.37. Secondary Exponential-Options

SIGMA=*value* | EST	Specifies σ_0 for distribution reference line
THETA=*value* | EST	Specifies θ_0 for distribution reference line

Table 3.38. Secondary Gamma-Options

ALPHA=*value-list* | EST	Specifies mandatory shape parameter α
SIGMA=*value* | EST	Specifies σ_0 for distribution reference line
THETA=*value* | EST	Specifies θ_0 for distribution reference line

Table 3.39. Secondary Lognormal-Options

SIGMA=*value*	Specifies mandatory shape parameter σ
SLOPE=*value* | EST	Specifies slope of distribution reference line
THETA=*value*|EST	Specifies θ_0 for distribution reference line
ZETA=*value*	Specifies ζ_0 for distribution reference line (slope is $\exp(\zeta_0)$)

Table 3.40. Secondary Normal-Options

MU=*value* | EST	Specifies μ_0 for distribution reference line
SIGMA=*value* | EST	Specifies σ_0 for distribution reference line

Table 3.41. Secondary Weibull-Options

C=*value-list* | EST	Specifies mandatory shape parameter *c*
SIGMA=*value* | EST	Specifies σ_0 for distribution reference line
THETA=*value* | EST	Specifies θ_0 for distribution reference line

Table 3.42. Secondary Weibull2-Options

C=*value* | EST	Specifies c_0 for distribution reference line (slope is $1/c_0$)
SIGMA=*value* | EST	Specifies σ_0 for distribution reference line (intercept is $\log(\sigma_0)$)
SLOPE=*value* | EST	Specifies slope of distribution reference line
THETA=*value*	Specifies known lower threshold θ_0

General Graphics Options

Table 3.43 summarizes general options for enhancing probability plots.

Table 3.43. General Graphics Options

Option	Description
ANNOKEY	Applies annotation requested in ANNOTATE= data set to key cell only
ANNOTATE=	Specifies annotate data set
CAXIS=	Specifies color for axis
CFRAME=	Specifies color for frame
CFRAMESIDE=	Specifies color for filling frame for row labels
CFRAMETOP=	Specifies color for filling frame for column labels
CGRID=	Specifies color for grid lines
CHREF=	Specifies color for HREF= lines
CTEXT=	Specifies color for text
CVREF=	Specifies color for VREF= lines
DESCRIPTION=	Specifies description for plot in graphics catalog
FONT=	Specifies software font for text
GRID	Creates a grid
HEIGHT=	Specifies height of text used outside framed areas
HMINOR=	Specifies number of horizontal minor tick marks
HREF=	Specifies reference lines perpendicular to the horizontal axis
HREFLABELS=	Specifies labels for HREF= lines
INFONT=	Specifies software font for text inside framed areas
INHEIGHT=	Specifies height of text inside framed areas
INTERTILE=	Specifies distance between tiles
LGRID=	Specifies a line type for grid lines
LHREF=	Specifies line style for HREF= lines
LVREF=	Specifies line style for VREF= lines
NADJ=	Adjusts sample size when computing percentiles
NAME=	Specifies name for plot in graphics catalog
NCOLS=	Specifies number of columns in comparative probability plot
NOFRAME	Suppresses frame around plotting area
NOHLABEL	Suppresses label for horizontal axis
NOVLABEL	Suppresses label for vertical axis
NOVTICK	Suppresses tick marks and tick mark labels for vertical axis
NROWS=	Specifies number of rows in comparative probability plot
PCTLMINOR	Requests minor tick marks for percentile axis
PCTLORDER=	Specifies tick mark labels for percentile axis
RANKADJ=	Adjusts ranks when computing percentiles
SQUARE	Displays plot in square format
VAXISLABEL=	Specifies label for vertical axis
VMINOR=	Specifies number of vertical minor tick marks
VREF=	Specifies reference lines perpendicular to the vertical axis
VREFLABELS=	Specifies labels for VREF= lines
VREFLABPOS=	Specifies horizontal position of labels for VREF= lines
WAXIS=	Specifies line thickness for axes and frame

Dictionary of Options

The following entries provide detailed descriptions of *options* in the PROBPLOT statement.

ALPHA=*value* **| EST**

specifies the mandatory shape parameter α for probability plots requested with the BETA and GAMMA options. Enclose the ALPHA= option in parentheses after the BETA or GAMMA options. If you specify ALPHA=EST, a maximum likelihood estimate is computed for α.

ANNOKEY

applies the annotation requested with the ANNOTATE= option to the key cell only. By default, the procedure applies annotation to all of the cells. This option is not available unless you use the CLASS statement. Specify the KEYLEVEL= option in the CLASS statement to specify the key cell.

ANNOTATE=*SAS-data-set*

ANNO=*SAS-data-set*

specifies an input data set containing annotate variables as described in *SAS/GRAPH Software: Reference*. The ANNOTATE= data set you specify in the HISTOGRAM statement is used for all plots created by the statement. You can also specify an ANNOTATE= data set in the PROC UNIVARIATE statement to enhance all plots created by the procedure.

BETA(ALPHA=*value* **| EST BETA=***value* **| EST** $<$*beta-options*$>$**)**

creates a beta probability plot for each combination of the required shape parameters α and β specified by the required ALPHA= and BETA= *beta-options*. If you specify ALPHA=EST and BETA=EST, the procedure creates a plot based on maximum likelihood estimates for α and β. You can specify the SCALE= *beta-option* as an alias for the SIGMA= *beta-option* and the THRESHOLD= *beta-option* as an alias for the THETA= *beta-option*. To create a plot that is based on maximum likelihood estimates for α and β, specify ALPHA=EST and BETA=EST.

To obtain graphical estimates of α and β, specify lists of values in the ALPHA= and BETA= *beta-options*, and select the combination of α and β that most nearly linearizes the point pattern. To assess the point pattern, you can add a diagonal distribution reference line corresponding to lower threshold parameter θ_0 and scale parameter σ_0 with the THETA= and SIGMA= *beta-options*. Alternatively, you can add a line that corresponds to estimated values of θ_0 and σ_0 with the *beta-options* THETA=EST and SIGMA=EST. Agreement between the reference line and the point pattern indicates that the beta distribution with parameters α, β, θ_0, and σ_0 is a good fit.

BETA=*value* **| EST**

B=*value* **| EST**

specifies the mandatory shape parameter β for probability plots requested with the BETA option. Enclose the BETA= option in parentheses after the BETA option. If you specify BETA=EST, a maximum likelihood estimate is computed for β.

C=*value* **| EST**

specifies the shape parameter c for probability plots requested with the WEIBULL and WEIBULL2 options. Enclose this option in parentheses after the WEIBULL or WEIBULL2 option. C= is a required *Weibull-option* in the WEIBULL option; in this situation, it accepts a list of values, or if you specify C=EST, a maximum likelihood estimate is computed for c. You can optionally specify C=*value* or C=EST as a *Weibull2-option* with the WEIBULL2 option to request a distribution reference line; in this situation, you must also specify *Weibull2-option* SIGMA=*value* or SIGMA=EST.

CAXIS=*color*
CAXES=*color*

specifies the color for the axes. This option overrides any COLOR= specifications in an AXIS statement. The default value is the first color in the device color list.

CFRAME=*color*

specifies the color for the area that is enclosed by the axes and frame. The area is not filled by default.

CFRAMESIDE=*color*

specifies the color to fill the frame area for the row labels that display along the left side of a comparative probability plot. This color also fills the frame area for the label of the corresponding class variable (if you associate a label with the variable). By default, these areas are not filled. This option is not available unless you use the CLASS statement.

CFRAMETOP=*color*

specifies the color to fill the frame area for the column labels that display across the top of a comparative probability plot. This color also fills the frame area for the label of the corresponding class variable (if you associate a label with the variable). By default, these areas are not filled. This option does not apply unless you use the CLASS statement.

CGRID=*color*

specifies the color for grid lines when a grid displays on the plot. The default *color* is the first color in the device color list. This option also produces a grid.

CHREF=*color*
CH=*color*

specifies the color for horizontal axis reference lines requested by the HREF= option. The default *color* is the first color in the device color list.

COLOR=*color*

specifies the color of the diagonal distribution reference line. The default *color* is the first color in the device color list. Enclose the COLOR= option in parentheses after a distribution option keyword.

CTEXT=*color*

specifies the color for tick mark values and axis labels. The default *color* is the color that you specify for the CTEXT= option in the GOPTIONS statement. If you omit the GOPTIONS statement, the default is the first color in the device color list.

CVREF=*color*
CV=*color*

specifies the color for the reference lines requested by the VREF= option. The default *color* is the first color in the device color list.

DESCRIPTION='*string***'**
DES='*string***'**

specifies a description, up to 40 characters long, that appears in the PROC GREPLAY master menu. The default *string* is the variable name.

EXPONENTIAL<**(***exponential-options***)**>
EXP<**(***exponential-options***)**>

creates an exponential probability plot. To assess the point pattern, add a diagonal distribution reference line corresponding to θ_0 and σ_0 with the THETA= and SIGMA= *exponential-options*. Alternatively, you can add a line corresponding to estimated values of the threshold parameter θ_0 and the scale parameter σ with the *exponential-options* THETA=EST and SIGMA=EST. Agreement between the reference line and the point pattern indicates that the exponential distribution with parameters θ_0 and σ_0 is a good fit. You can specify the SCALE= *exponential-option* as an alias for the SIGMA= *exponential-option* and the THRESHOLD= *exponential-option* as an alias for the THETA= *exponential-option*.

FONT=*font*

specifies a software font for the reference lines and axis labels. You can also specify fonts for axis labels in an AXIS statement. The FONT= font takes precedence over the FTEXT= font specified in the GOPTIONS statement. Hardware characters are used by default.

GAMMA(ALPHA=*value* **| EST** <*gamma-options*>**)**

creates a gamma probability plot for each value of the shape parameter α given by the mandatory ALPHA= *gamma-option*. If you specify ALPHA=EST, the procedure creates a plot based on a maximum likelihood estimate for α. To obtain a graphical estimate of α, specify a list of values for the ALPHA= *gamma-option*, and select the value that most nearly linearizes the point pattern. To assess the point pattern, add a diagonal distribution reference line corresponding to θ_0 and σ_0 with the THETA= and SIGMA= *gamma-options*. Alternatively, you can add a line corresponding to estimated values of the threshold parameter θ_0 and the scale parameter σ with the *gamma-options* THETA=EST and SIGMA=EST. Agreement between the reference line and the point pattern indicates that the gamma distribution with parameters α, θ_0 and σ_0 is a good fit. You can specify the SCALE= *gamma-option* as an alias for the SIGMA= *gamma-option* and the THRESHOLD= *gamma-option* as an alias for the THETA= *gamma-option*.

GRID

displays a grid. Grid lines are reference lines that are perpendicular to the percentile axis at major tick marks.

HEIGHT=*value*

specifies the height, in percentage screen units, of text for axis labels, tick mark labels, and legends. This option takes precedence over the HTEXT= option in the GOPTIONS statement.

HMINOR=*n*
HM=*n*

specifies the number of minor tick marks between each major tick mark on the horizontal axis. Minor tick marks are not labeled. By default, HMINOR=0.

HREF=*values*

draws reference lines that are perpendicular to the horizontal axis at the values you specify.

HREFLABELS='*label1***'** . . . **'** *labeln'*
HREFLABEL='*label1***'** . . . **'** *labeln'*
HREFLAB='*label1***'** . . . **'** *labeln'*

specifies labels for the reference lines requested by the HREF= option. The number of labels must equal the number of reference lines. Labels can have up to 16 characters.

HREFLABPOS=*n*

specifies the vertical position of HREFLABELS= labels. If you specify HREFLABPOS=1, the labels are positioned along the top of the plot. If you specify HREFLABPOS=2, the labels are staggered from top to bottom of the plot. If you specify HREFLABPOS=3, the labels are positioned along the bottom of the plot. By default, HREFLABPOS=1.

INFONT=*font*

specifies a software font to use for text inside the framed areas of the plot. The INFONT= option takes precedence over the FTEXT= option in the GOPTIONS statement. For a list of fonts, see *SAS/GRAPH Reference*.

INHEIGHT=*value*

specifies the height, in percentage screen units, of text used inside the framed areas of the plot. By default, the height specified by the HEIGHT= option is used. If you do not specify the HEIGHT= option, the height specified with the HTEXT= option in the GOPTIONS statement is used.

INTERTILE=*value*

specifies the distance, in horizontal percentage screen units, between the framed areas, which are called *tiles*. By default, the tiles are contiguous. This option is not available unless you use the CLASS statement.

L=*linetype*

specifies the line type for a diagonal distribution reference line. Enclose the L= option in parentheses after a distribution option. By default, L=1, which produces a solid line.

LGRID=*linetype*
specifies the line type for the grid requested by the GRID= option. By default, LGRID=1, which produces a solid line.

LHREF=*linetype*
LH=*linetype*
specifies the line type for the reference lines that you request with the HREF= option. By default, LHREF=2, which produces a dashed line.

LOGNORMAL(SIGMA=*value* **| EST** <*lognormal-options*>**)**
LNORM(SIGMA=*value* **| EST** <*lognormal-options*>**)**
creates a lognormal probability plot for each value of the shape parameter σ given by the mandatory SIGMA= *lognormal-option*. If you specify SIGMA=EST, the procedure creates a plot based on a maximum likelihood estimate for σ. To obtain a graphical estimate of σ, specify a list of values for the SIGMA= *lognormal-option*, and select the value that most nearly linearizes the point pattern. To assess the point pattern, add a diagonal distribution reference line corresponding to θ_0 and ζ_0 with the THETA= and ZETA= *lognormal-options*. Alternatively, you can add a line corresponding to estimated values of the threshold parameter θ_0 and the scale parameter ζ_0 with the *lognormal-options* THETA=EST and ZETA=EST. Agreement between the reference line and the point pattern indicates that the lognormal distribution with parameters σ, θ_0 and ζ_0 is a good fit. You can specify the THRESHOLD= *lognormal-option* as an alias for the THETA= *lognormal-option* and the SCALE= *lognormal-option* as an alias for the ZETA= *lognormal-option*. See Example 3.26.

LVREF=*linetype*
specifies the line type for the reference lines requested with the VREF= option. By default, LVREF=2, which produces a dashed line.

MU=*value* **| EST**
specifies the mean μ_0 for a normal probability plot requested with the NORMAL option. Enclose the MU= *normal-option* in parentheses after the NORMAL option. The MU= *normal-option* must be specified with the SIGMA= *normal-option*, and they request a distribution reference line. You can specify MU=EST to request a distribution reference line with μ_0 equal to the sample mean.

NADJ=*value*
specifies the adjustment value added to the sample size in the calculation of theoretical percentiles. By default, NADJ=$\frac{1}{4}$. Refer to Chambers et al. (1983).

NAME='*string***'**
specifies a name for the plot, up to eight characters long, that appears in the PROC GREPLAY master menu. The default value is 'UNIVAR'.

NCOLS=*n*
NCOL=*n*
specifies the number of columns in a comparative probability plot. By default, NCOLS=1 if you specify only one class variable, and NCOLS=2 if you specify two class variables. This option is not available unless you use the CLASS statement. If you specify two class variables, you can use the NCOLS= option with the NROWS= option.

NOFRAME
suppresses the frame around the subplot area.

NOHLABEL
suppresses the label for the horizontal axis. You can use this option to reduce clutter.

NORMAL<**(***normal-options***)**>
creates a normal probability plot. This is the default if you omit a distribution option. To assess the point pattern, you can add a diagonal distribution reference line corresponding to μ_0 and σ_0 with the MU= and SIGMA= *normal-options*. Alternatively, you can add a line corresponding to estimated values of μ_0 and σ_0 with the *normal-options* MU=EST and SIGMA=EST; the estimates of the mean μ_0 and the standard deviation σ_0 are the sample mean and sample standard deviation. Agreement between the reference line and the point pattern indicates that the normal distribution with parameters μ_0 and σ_0 is a good fit.

NOVLABEL
suppresses the label for the vertical axis. You can use this option to reduce clutter.

NOVTICK
suppresses the tick marks and tick mark labels for the vertical axis. This option also suppresses the label for the vertical axis.

NROWS=*n*
NROW=*n*
specifies the number of rows in a comparative probability plot. By default, NROWS=2. This option is not available unless you use the CLASS statement. If you specify two class variables, you can use the NCOLS= option with the NROWS= option.

PCTLMINOR
requests minor tick marks for the percentile axis. The HMINOR option overrides the minor tick marks requested by the PCTLMINOR option.

PCTLORDER=*values*
specifies the tick marks that are labeled on the theoretical percentile axis. Since the values are percentiles, the labels must be between 0 and 100, exclusive. The values must be listed in increasing order and must cover the plotted percentile range. Otherwise, the default values of 1, 5, 10, 25, 50, 75, 90, 95, and 99 are used.

RANKADJ=*value*
specifies the adjustment value added to the ranks in the calculation of theoretical percentiles. By default, RANKADJ=$-\frac{3}{8}$, as recommended by Blom (1958). Refer to Chambers et al. (1983) for additional information.

SCALE=*value* **| EST**
is an alias for the SIGMA= option for plots requested by the BETA, EXPONENTIAL, GAMMA, and WEIBULL options and for the ZETA= option when you request the LOGNORMAL option. See the entries for the SIGMA= and ZETA= options.

SHAPE=*value* | EST

is an alias for the ALPHA= option with the GAMMA option, for the SIGMA= option with the LOGNORMAL option, and for the C= option with the WEIBULL and WEIBULL2 options. See the entries for the ALPHA=, SIGMA=, and C= options.

SIGMA=*value* | EST

specifies the parameter σ, where $\sigma > 0$. Alternatively, you can specify SIGMA=EST to request a maximum likelihood estimate for σ_0. The interpretation and use of the SIGMA= option depend on the distribution option with which it is used. See Table 3.44 for a summary of how to use the SIGMA= option. You must enclose this option in parentheses after the distribution option.

Table 3.44. Uses of the SIGMA= Option

Distribution Option	Use of the SIGMA= Option
BETA EXPONENTIAL GAMMA WEIBULL	THETA=θ_0 and SIGMA=σ_0 request a distribution reference line corresponding to θ_0 and σ_0.
LOGNORMAL	SIGMA=$\sigma_1 \ldots \sigma_n$ requests n probability plots with shape parameters $\sigma_1 \ldots \sigma_n$. The SIGMA= option must be specified.
NORMAL	MU=μ_0 and SIGMA=σ_0 request a distribution reference line corresponding to μ_0 and σ_0. SIGMA=EST requests a line with σ_0 equal to the sample standard deviation.
WEIBULL2	SIGMA=σ_0 and C=c_0 request a distribution reference line corresponding to σ_0 and c_0.

SLOPE=*value* | EST

specifies the slope for a distribution reference line requested with the LOGNORMAL and WEIBULL2 options. Enclose the SLOPE= option in parentheses after the distribution option. When you use the SLOPE= *lognormal-option* with the LOGNORMAL option, you must also specify a threshold parameter value θ_0 with the THETA= *lognormal-option* to request the line. The SLOPE= *lognormal-option* is an alternative to the ZETA= *lognormal-option* for specifying ζ_0, since the slope is equal to $\exp(\zeta_0)$.

When you use the SLOPE= *Weibull2-option* with the WEIBULL2 option, you must also specify a scale parameter value σ_0 with the SIGMA= *Weibull2-option* to request the line. The SLOPE= *Weibull2-option* is an alternative to the C= *Weibull2-option* for specifying c_0, since the slope is equal to $\frac{1}{c_0}$.

For example, the first and second PROBPLOT statements produce the same probability plots and the third and fourth PROBPLOT statements produce the same probability plots:

```
proc univariate data=Measures;
   probplot Width / lognormal(sigma=2 theta=0 zeta=0);
   probplot Width / lognormal(sigma=2 theta=0 slope=1);
   probplot Width / weibull2(sigma=2 theta=0 c=.25);
   probplot Width / weibull2(sigma=2 theta=0 slope=4);
run;
```

SQUARE

displays the probability plot in a square frame. By default, the plot is in a rectangular frame.

THETA=*value* | EST

specifies the lower threshold parameter θ for plots requested with the BETA, EXPONENTIAL, GAMMA, LOGNORMAL, WEIBULL, and WEIBULL2 options. Enclose the THETA= option in parentheses after a distribution option. When used with the WEIBULL2 option, the THETA= option specifies the known lower threshold θ_0, for which the default is 0. When used with the other distribution options, the THETA= option specifies θ_0 for a distribution reference line; alternatively in this situation, you can specify THETA=EST to request a maximum likelihood estimate for θ_0. To request the line, you must also specify a scale parameter.

THRESHOLD=*value* | EST

is an alias for the THETA= option.

VAXISLABEL=*'label'*

specifies a label for the vertical axis. Labels can have up to 40 characters.

VMINOR=*n*
VM=*n*

specifies the number of minor tick marks between each major tick mark on the vertical axis. Minor tick marks are not labeled. The default is zero.

VREF=*values*

draws reference lines perpendicular to the vertical axis at the values specified. Also see the CVREF=, LVREF=, and VREFCHAR= options.

VREFLABELS='*label1*'. . . '*labeln*'
VREFLABEL='*label1*'. . . '*labeln*'
VREFLAB='*label1*'. . . '*labeln*'

specifies labels for the reference lines requested by the VREF= option. The number of labels must equal the number of reference lines. Enclose each label in quotes. Labels can have up to 16 characters.

VREFLABPOS=*n*

specifies the horizontal position of VREFLABELS= labels. If you specify VREFLABPOS=1, the labels are positioned at the left of the histogram. If you specify VREFLABPOS=2, the labels are positioned at the right of the histogram. By default, VREFLABPOS=1.

W=*n*

specifies the width, in pixels, for a diagonal distribution line. Enclose the W= option in parentheses after the distribution option. By default, W=1.

WAXIS=*n*

specifies the line thickness, in pixels, for the axes and frame. By default, WAXIS=1.

WEIBULL(C=*value* | EST <*Weibull-options*>)
WEIB(C=*value* | EST <*Weibull-options*>)

creates a three-parameter Weibull probability plot for each value of the required shape parameter c specified by the mandatory C= *Weibull-option*. To create a plot that is based on a maximum likelihood estimate for c, specify C=EST. To obtain a graphical estimate of c, specify a list of values in the C= *Weibull-option*, and select the value that most nearly linearizes the point pattern. To assess the point pattern, add a diagonal distribution reference line corresponding to θ_0 and σ_0 with the THETA= and SIGMA= *Weibull-options*. Alternatively, you can add a line corresponding to estimated values of θ_0 and σ_0 with the *Weibull-options* THETA=EST and SIGMA=EST. Agreement between the reference line and the point pattern indicates that the Weibull distribution with parameters c, θ_0, and σ_0 is a good fit. You can specify the SCALE= *Weibull-option* as an alias for the SIGMA= *Weibull-option* and the THRESHOLD= *Weibull-option* as an alias for the THETA= *Weibull-option*.

WEIBULL2<(*Weibull2-options*)>
W2<(*Weibull2-options*)>

creates a two-parameter Weibull probability plot. You should use the WEIBULL2 option when your data have a *known* lower threshold θ_0, which is 0 by default. To specify the threshold value θ_0, use the THETA= *Weibull2-option*. By default, THETA=0. An advantage of the two-parameter Weibull plot over the three-parameter Weibull plot is that the parameters c and σ can be estimated from the slope and intercept of the point pattern. A disadvantage is that the two-parameter Weibull distribution applies only in situations where the threshold parameter is known. To obtain a graphical estimate of θ_0, specify a list of values for the THETA= *Weibull2-option*, and select the value that most nearly linearizes the point pattern. To assess the point pattern, add a diagonal distribution reference line corresponding to σ_0 and c_0 with the SIGMA= and C= *Weibull2-options*. Alternatively, you can add a distribution reference line corresponding to estimated values of σ_0 and c_0 with the *Weibull2-options* SIGMA=EST and C=EST. Agreement between the reference line and the point pattern indicates that the Weibull distribution with parameters c_0, θ_0, and σ_0 is a good fit. You can specify the SCALE= *Weibull2-option* as an alias for the SIGMA= *Weibull2-option* and the SHAPE= *Weibull2-option* as an alias for the C= *Weibull2-option*.

ZETA=*value* | EST

specifies a value for the scale parameter ζ for the lognormal probability plots requested with the LOGNORMAL option. Enclose the ZETA= *lognormal-option* in parentheses after the LOGNORMAL option. To request a distribution reference line with intercept θ_0 and slope $\exp(\zeta_0)$, specify the THETA=θ_0 and ZETA=ζ_0.

QQPLOT Statement

QQPLOT < *variables* >< / *options* >**;**

The QQPLOT statement creates quantile-quantile plots (Q-Q plots) using high-resolution graphics and compares ordered variable values with quantiles of a specified theoretical distribution. If the data distribution matches the theoretical distribution, the points on the plot form a linear pattern. Thus, you can use a Q-Q plot to determine how well a theoretical distribution models a set of measurements.

Q-Q plots are similar to probability plots, which you can create with the PROBPLOT statement. Q-Q plots are preferable for graphical estimation of distribution parameters, whereas probability plots are preferable for graphical estimation of percentiles.

You can use any number of QQPLOT statements in the UNIVARIATE procedure. The components of the QQPLOT statement are described as follows.

variables

are the variables for which to create Q-Q plots. If you specify a VAR statement, the *variables* must also be listed in the VAR statement. Otherwise, the *variables* can be any numeric variables in the input data set. If you do not specify a list of *variables*, then by default the procedure creates a Q-Q plot for each variable listed in the VAR statement, or for each numeric variable in the DATA= data set if you do not specify a VAR statement. For example, each of the following QQPLOT statements produces two Q-Q plots, one for Length and one for Width:

```
proc univariate data=Measures;
   var Length Width;
   qqplot;

proc univariate data=Measures;
   qqplot Length Width;
run;
```

options

specify the theoretical distribution for the plot or add features to the plot. If you specify more than one variable, the *options* apply equally to each variable. Specify all *options* after the slash (/) in the QQPLOT statement. You can specify only one *option* naming the distribution in each QQPLOT statement, but you can specify any number of other *options*. The distributions available are the beta, exponential, gamma, lognormal, normal, two-parameter Weibull, and three-parameter Weibull. By default, the procedure produces a plot for the normal distribution.

In the following example, the NORMAL option requests a normal Q-Q plot for each variable. The MU= and SIGMA= *normal-options* request a distribution reference line with intercept 10 and slope 0.3 for each plot, corresponding to a normal distribution with mean $\mu = 10$ and standard deviation $\sigma = 0.3$. The SQUARE option displays the plot in a square frame, and the CTEXT= option specifies the text color.

```
proc univariate data=measures;
   qqplot length1 length2 / normal(mu=10 sigma=0.3)
                            square ctext=blue;
run;
```

Table 3.45 through Table 3.54 list the QQPLOT *options* by function. For complete descriptions, see the section “Dictionary of Options” on page 258.

Options can be any of the following:

- *primary options*
- *secondary options*
- *general options*

Distribution Options

Table 3.45 lists *primary options* for requesting a theoretical distribution.

Table 3.45. Primary Options for Theoretical Distributions

BETA(*beta-options*)	Specifies beta Q-Q plot for shape parameters α and β specified with mandatory ALPHA= and BETA= *beta-options*
EXPONENTIAL(*exponential-options*)	Specifies exponential Q-Q plot
GAMMA(*gamma-options*)	Specifies gamma Q-Q plot for shape parameter α specified with mandatory ALPHA= *gamma-option*
LOGNORMAL(*lognormal-options*)	Specifies lognormal Q-Q plot for shape parameter σ specified with mandatory SIGMA= *lognormal-option*
NORMAL(*normal-options*)	Specifies normal Q-Q plot
WEIBULL(*Weibull-options*)	Specifies three-parameter Weibull Q-Q plot for shape parameter c specified with mandatory C= *Weibull-option*
WEIBULL2(*Weibull2-options*)	Specifies two-parameter Weibull Q-Q plot

Table 3.46 through Table 3.53 list *secondary options* that specify distribution parameters and control the display of a distribution reference line. Specify these options in parentheses after the distribution keyword. For example, you can request a normal Q-Q plot with a distribution reference line by specifying the NORMAL option as follows:

```
proc univariate;
   qqplot Length / normal(mu=10 sigma=0.3 color=red);
run;
```

The MU= and SIGMA= *normal-options* display a distribution reference line that corresponds to the normal distribution with mean $\mu_0 = 10$ and standard deviation $\sigma_0 = 0.3$, and the COLOR= *normal-option* specifies the color for the line.

Table 3.46. Secondary Reference Line Options Used with All Distributions

COLOR=*color*	Specifies color of distribution reference line
L=*linetype*	Specifies line type of distribution reference line
W=n	Specifies width of distribution reference line

Table 3.47. Secondary Beta-Options

ALPHA=*value-list* | EST	Specifies mandatory shape parameter α
BETA=*value-list* | EST	Specifies mandatory shape parameter β
SIGMA=*value* | EST	Specifies σ_0 for distribution reference line
THETA=*value* | EST	Specifies θ_0 for distribution reference line

Table 3.48. Secondary Exponential-Options

SIGMA=*value* | EST	Specifies σ_0 for distribution reference line
THETA=*value* | EST	Specifies θ_0 for distribution reference line

Table 3.49. Secondary Gamma-Options

ALPHA=*value-list* | EST	Specifies mandatory shape parameter α
SIGMA=*value* | EST	Specifies σ_0 for distribution reference line
THETA=*value* | EST	Specifies θ_0 for distribution reference line

Table 3.50. Secondary Lognormal-Options

SIGMA=*value-list* | EST	Specifies mandatory shape parameter σ
SLOPE=*value* | EST	Specifies slope of distribution reference line
THETA=*value*|EST	Specifies θ_0 for distribution reference line
ZETA=*value*	Specifies ζ_0 for distribution reference line (slope is $\exp(\zeta_0)$)

Table 3.51. Secondary Normal-Options

MU=*value* | EST	Specifies μ_0 for distribution reference line
SIGMA=*value* | EST	Specifies σ_0 for distribution reference line

Table 3.52. Secondary Weibull-Options

C=*value-list* | EST	Specifies mandatory shape parameter *c*
SIGMA=*value* | EST	Specifies σ_0 for distribution reference line
THETA=*value* | EST	Specifies θ_0 for distribution reference line

Table 3.53. Secondary Weibull2-Options

C=*value* | EST	Specifies c_0 for distribution reference line (slope is $1/c_0$)
SIGMA=*value* | EST	Specifies σ_0 for distribution reference line (intercept is $\log(\sigma_0)$)
SLOPE=*value* | EST	Specifies slope of distribution reference line
THETA=*value*	Specifies known lower threshold θ_0

General Options

Table 3.54 summarizes *general options* for enhancing Q-Q plots.

Table 3.54. General Graphics Options

Option	Description
ANNOKEY	Applies annotation requested in ANNOTATE= data set to key cell only
ANNOTATE=	Specifies annotate data set

Table 3.54. (continued)

Option	Description
CAXIS=	Specifies color for axis
CFRAME=	Specifies color for frame
CFRAMESIDE=	Specifies color for filling frame for row labels
CFRAMETOP=	Specifies color for filling frame for column labels
CGRID=	Specifies color for grid lines
CHREF=	Specifies color for HREF= lines
CTEXT=	Specifies color for text
CVREF=	Specifies color for VREF= lines
DESCRIPTION=	Specifies description for plot in graphics catalog
FONT=	Specifies software font for text
GRID	Creates a grid
HEIGHT=	Specifies height of text used outside framed areas
HMINOR=	Specifies number of horizontal minor tick marks
HREF=	Specifies reference lines perpendicular to the horizontal axis
HREFLABELS=	Specifies labels for HREF= lines
HREFLABPOS=	Specifies vertical position of labels for HREF= lines
INFONT=	Specifies software font for text inside framed areas
INHEIGHT=	Specifies height of text inside framed areas
INTERTILE=	Specifies distance between tiles
LGRID=	Specifies a line type for grid lines
LHREF=	Specifies line style for HREF= lines
LVREF=	Specifies line style for VREF= lines
NADJ=	Adjusts sample size when computing percentiles
NAME=	Specifies name for plot in graphics catalog
NCOLS=	Specifies number of columns in comparative Q-Q plot
NOFRAME	Suppresses frame around plotting area
NOHLABEL	Suppresses label for horizontal axis
NOVLABEL	Suppresses label for vertical axis
NOVTICK	Suppresses tick marks and tick mark labels for vertical axis
NROWS=	Specifies number of rows in comparative Q-Q plot
PCTLAXIS	Displays a nonlinear percentile axis
PCTLMINOR	Requests minor tick marks for percentile axis
PCTLSCALE	Replaces theoretical quantiles with percentiles
RANKADJ=	Adjusts ranks when computing percentiles
SQUARE	Displays plot in square format
VAXISLABEL=	Specifies label for vertical axis
VMINOR=	Specifies number of vertical minor tick marks
VREF=	Specifies reference lines perpendicular to the vertical axis
VREFLABELS=	Specifies labels for VREF= lines
VREFLABPOS=	Specifies horizontal position of labels for VREF= lines
WAXIS=	Specifies line thickness for axes and frame

Dictionary of Options

The following entries provide detailed descriptions of *options* in the QQPLOT statement.

ALPHA=*value* **| EST**

specifies the mandatory shape parameter α for quantile plots requested with the BETA and GAMMA options. Enclose the ALPHA= option in parentheses after the BETA or GAMMA options. If you specify ALPHA=EST, a maximum likelihood estimate is computed for α.

ANNOKEY

applies the annotation requested with the ANNOTATE= option to the key cell only. By default, the procedure applies annotation to all of the cells. This option is not available unless you use the CLASS statement. Specify the KEYLEVEL= option in the CLASS statement to specify the key cell.

ANNOTATE=*SAS-data-set*
ANNO=*SAS-data-set*

specifies an input data set containing annotate variables as described in *SAS/GRAPH Software: Reference*. The ANNOTATE= data set you specify in the HISTOGRAM statement is used for all plots created by the statement. You can also specify an ANNOTATE= data set in the PROC UNIVARIATE statement to enhance all plots created by the procedure.

BETA(ALPHA=*value* **| EST BETA=***value* **| EST** <*beta-options*>**)**

creates a beta quantile plot for each combination of the required shape parameters α and β specified by the required ALPHA= and BETA= *beta-options*. If you specify ALPHA=EST and BETA=EST, the procedure creates a plot based on maximum likelihood estimates for α and β. You can specify the SCALE= *beta-option* as an alias for the SIGMA= *beta-option* and the THRESHOLD= *beta-option* as an alias for the THETA= *beta-option*. To create a plot that is based on maximum likelihood estimates for α and β, specify ALPHA=EST and BETA=EST.

To obtain graphical estimates of α and β, specify lists of values in the ALPHA= and BETA= *beta-options*, and select the combination of α and β that most nearly linearizes the point pattern. To assess the point pattern, you can add a diagonal distribution reference line corresponding to lower threshold parameter θ_0 and scale parameter σ_0 with the THETA= and SIGMA= *beta-options*. Alternatively, you can add a line that corresponds to estimated values of θ_0 and σ_0 with the *beta-options* THETA=EST and SIGMA=EST. Agreement between the reference line and the point pattern indicates that the beta distribution with parameters α, β, θ_0, and σ_0 is a good fit.

BETA=*value* **| EST**
B=*value* **| EST**

specifies the mandatory shape parameter β for quantile plots requested with the BETA option. Enclose the BETA= option in parentheses after the BETA option. If you specify BETA=EST, a maximum likelihood estimate is computed for β.

C=*value* | EST

specifies the shape parameter c for quantile plots requested with the WEIBULL and WEIBULL2 options. Enclose this option in parentheses after the WEIBULL or WEIBULL2 option. C= is a required *Weibull-option* in the WEIBULL option; in this situation, it accepts a list of values, or if you specify C=EST, a maximum likelihood estimate is computed for c. You can optionally specify C=*value* or C=EST as a *Weibull2-option* with the WEIBULL2 option to request a distribution reference line; in this situation, you must also specify *Weibull2-option* SIGMA=*value* or SIGMA=EST.

CAXIS=*color*
CAXES=*color*

specifies the color for the axes. This option overrides any COLOR= specifications in an AXIS statement. The default value is the first color in the device color list.

CFRAME=*color*

specifies the color for the area that is enclosed by the axes and frame. The area is not filled by default.

CFRAMESIDE=*color*

specifies the color to fill the frame area for the row labels that display along the left side of a comparative quantile plot. This color also fills the frame area for the label of the corresponding class variable (if you associate a label with the variable). By default, these areas are not filled. This option is not available unless you use the CLASS statement.

CFRAMETOP=*color*

specifies the color to fill the frame area for the column labels that display across the top of a comparative quantile plot. This color also fills the frame area for the label of the corresponding class variable (if you associate a label with the variable). By default, these areas are not filled. This option does not apply unless you use the CLASS statement.

CGRID=*color*

specifies the color for grid lines when a grid displays on the plot. The default *color* is the first color in the device color list. This option also produces a grid.

CHREF=*color*
CH=*color*

specifies the color for horizontal axis reference lines requested by the HREF= option. The default *color* is the first color in the device color list.

COLOR=*color*

specifies the color of the diagonal distribution reference line. The default *color* is the first color in the device color list. Enclose the COLOR= option in parentheses after a distribution option keyword.

CTEXT=*color*

specifies the color for tick mark values and axis labels. The default *color* is the color that you specify for the CTEXT= option in the GOPTIONS statement. If you omit the GOPTIONS statement, the default is the first color in the device color list.

CVREF=*color*
CV=*color*
specifies the color for the reference lines requested by the VREF= option. The default *color* is the first color in the device color list.

DESCRIPTION='*string***'**
DES='*string***'**
specifies a description, up to 40 characters long, that appears in the PROC GREPLAY master menu. The default *string* is the variable name.

EXPONENTIAL<**(***exponential-options***)**>
EXP<**(***exponential-options***)**>
creates an exponential quantile plot. To assess the point pattern, add a diagonal distribution reference line corresponding to θ_0 and σ_0 with the THETA= and SIGMA= *exponential-options*. Alternatively, you can add a line corresponding to estimated values of the threshold parameter θ_0 and the scale parameter σ with the *exponential-options* THETA=EST and SIGMA=EST. Agreement between the reference line and the point pattern indicates that the exponential distribution with parameters θ_0 and σ_0 is a good fit. You can specify the SCALE= *exponential-option* as an alias for the SIGMA= *exponential-option* and the THRESHOLD= *exponential-option* as an alias for the THETA= *exponential-option*.

FONT=*font*
specifies a software font for the reference lines and axis labels. You can also specify fonts for axis labels in an AXIS statement. The FONT= font takes precedence over the FTEXT= font specified in the GOPTIONS statement. Hardware characters are used by default.

GAMMA(ALPHA=*value* **| EST** <*gamma-options*>**)**
creates a gamma quantile plot for each value of the shape parameter α given by the mandatory ALPHA= *gamma-option*. If you specify ALPHA=EST, the procedure creates a plot based on a maximum likelihood estimate for α. To obtain a graphical estimate of α, specify a list of values for the ALPHA= *gamma-option*, and select the value that most nearly linearizes the point pattern. To assess the point pattern, add a diagonal distribution reference line corresponding to θ_0 and σ_0 with the THETA= and SIGMA= *gamma-options*. Alternatively, you can add a line corresponding to estimated values of the threshold parameter θ_0 and the scale parameter σ with the *gamma-options* THETA=EST and SIGMA=EST. Agreement between the reference line and the point pattern indicates that the gamma distribution with parameters α, θ_0 and σ_0 is a good fit. You can specify the SCALE= *gamma-option* as an alias for the SIGMA= *gamma-option* and the THRESHOLD= *gamma-option* as an alias for the THETA= *gamma-option*.

GRID
displays a grid of horizontal lines positioned at major tick marks on the vertical axis.

HEIGHT=*value*
specifies the height, in percentage screen units, of text for axis labels, tick mark labels, and legends. This option takes precedence over the HTEXT= option in the GOPTIONS statement.

HMINOR=*n*
HM=*n*
specifies the number of minor tick marks between each major tick mark on the horizontal axis. Minor tick marks are not labeled. By default, HMINOR=0.

HREF=*values*
draws reference lines that are perpendicular to the horizontal axis at specified values. When you use the PCTLAXIS option, HREF= *values* must be in quantile units.

HREFLABELS='*label1***'** . . . **'** *labeln'*
HREFLABEL='*label1***'** . . . **'** *labeln'*
HREFLAB='*label1***'** . . . **'** *labeln'*
specifies labels for the reference lines requested by the HREF= option. The number of labels must equal the number of reference lines. Labels can have up to 16 characters.

HREFLABPOS=*n*
specifies the vertical position of HREFLABELS= labels. If you specify HREFLABPOS=1, the labels are positioned along the top of the plot. If you specify HREFLABPOS=2, the labels are staggered from top to bottom of the plot. If you specify HREFLABPOS=3, the labels are positioned along the bottom of the plot. By default, HREFLABPOS=1.

INFONT=*font*
specifies a software font to use for text inside the framed areas of the plot. The INFONT= option takes precedence over the FTEXT= option in the GOPTIONS statement. For a list of fonts, see *SAS/GRAPH Reference*.

INHEIGHT=*value*
specifies the height, in percentage screen units, of text used inside the framed areas of the plot. By default, the height specified by the HEIGHT= option is used. If you do not specify the HEIGHT= option, the height specified with the HTEXT= option in the GOPTIONS statement is used.

INTERTILE=*value*
specifies the distance, in horizontal percentage screen units, between the framed areas, which are called *tiles*. By default, INTERTILE=0.75 percentage screen units. This option is not available unless you use the CLASS statement. You can specify INTERTILE=0 to create contiguous tiles.

L=*linetype*
specifies the line type for a diagonal distribution reference line. Enclose the L= option in parentheses after a distribution option. By default, L=1, which produces a solid line.

LGRID=*linetype*
specifies the line type for the grid requested by the GRID option. By default, LGRID=1, which produces a solid line. The LGRID= option also produces a grid.

LHREF=*linetype*
LH=*linetype*
specifies the line type for the reference lines that you request with the HREF= option. By default, LHREF=2, which produces a dashed line.

LOGNORMAL(SIGMA=*value* | EST <*lognormal-options*>)
LNORM(SIGMA=*value* | EST <*lognormal-options*>)

creates a lognormal quantile plot for each value of the shape parameter σ given by the mandatory SIGMA= *lognormal-option*. If you specify SIGMA=EST, the procedure creates a plot based on a maximum likelihood estimate for σ. To obtain a graphical estimate of σ, specify a list of values for the SIGMA= *lognormal-option*, and select the value that most nearly linearizes the point pattern. To assess the point pattern, add a diagonal distribution reference line corresponding to θ_0 and ζ_0 with the THETA= and ZETA= *lognormal-options*. Alternatively, you can add a line corresponding to estimated values of the threshold parameter θ_0 and the scale parameter ζ_0 with the *lognormal-options* THETA=EST and ZETA=EST. Agreement between the reference line and the point pattern indicates that the lognormal distribution with parameters σ, θ_0 and ζ_0 is a good fit. You can specify the THRESHOLD= *lognormal-option* as an alias for the THETA= *lognormal-option* and the SCALE= *lognormal-option* as an alias for the ZETA= *lognormal-option*. See Example 3.31 through Example 3.33.

LVREF=*linetype*

specifies the line type for the reference lines requested with the VREF= option. By default, LVREF=2, which produces a dashed line.

MU=*value* | EST

specifies the mean μ_0 for a normal quantile plot requested with the NORMAL option. Enclose the MU= *normal-option* in parentheses after the NORMAL option. The MU= *normal-option* must be specified with the SIGMA= *normal-option*, and they request a distribution reference line. You can specify MU=EST to request a distribution reference line with μ_0 equal to the sample mean.

NADJ=*value*

specifies the adjustment value added to the sample size in the calculation of theoretical percentiles. By default, NADJ=$\frac{1}{4}$. Refer to Chambers et al. (1983) for additional information.

NAME='*string*'

specifies a name for the plot, up to eight characters long, that appears in the PROC GREPLAY master menu. The default value is 'UNIVAR'.

NCOLS=*n*
NCOL=*n*

specifies the number of columns in a comparative quantile plot. By default, NCOLS=1 if you specify only one class variable, and NCOLS=2 if you specify two class variables. This option is not available unless you use the CLASS statement. If you specify two class variables, you can use the NCOLS= option with the NROWS= option.

NOFRAME

suppresses the frame around the subplot area. If you specify the PCTLAXIS option, then you cannot specify the NOFRAME option.

NOHLABEL

suppresses the label for the horizontal axis. You can use this option to reduce clutter.

NORMAL<**(***normal-options***)**>

creates a normal quantile plot. This is the default if you omit a distribution option. To assess the point pattern, you can add a diagonal distribution reference line corresponding to μ_0 and σ_0 with the MU= and SIGMA= *normal-options*. Alternatively, you can add a line corresponding to estimated values of μ_0 and σ_0 with the *normal-options* MU=EST and SIGMA=EST; the estimates of the mean μ_0 and the standard deviation σ_0 are the sample mean and sample standard deviation. Agreement between the reference line and the point pattern indicates that the normal distribution with parameters μ_0 and σ_0 is a good fit. See Example 3.28 and Example 3.30.

NOVLABEL

suppresses the label for the vertical axis. You can use this option to reduce clutter.

NOVTICK

suppresses the tick marks and tick mark labels for the vertical axis. This option also suppresses the label for the vertical axis.

NROWS=*n*
NROW=*n*

specifies the number of rows in a comparative quantile plot. By default, NROWS=2. This option is not available unless you use the CLASS statement. If you specify two class variables, you can use the NCOLS= option with the NROWS= option.

PCTLAXIS<**(***axis-options***)**>

adds a nonlinear percentile axis along the frame of the Q-Q plot opposite the theoretical quantile axis. The added axis is identical to the axis for probability plots produced with the PROBPLOT statement. When using the PCTLAXIS option, you must specify HREF= values in quantile units, and you cannot use the NOFRAME option. You can specify the following *axis-options*:

Table 3.55. Axis Options

GRID	Draws vertical grid lines at major percentiles
GRIDCHAR='*character*'	Specifies grid line plotting character on line printer
LABEL='*string*'	Specifies label for percentile axis
LGRID=*linetype*	Specifies line type for grid

PCTLMINOR

requests minor tick marks for the percentile axis when you specify PCTLAXIS. The HMINOR option overrides the PCTLMINOR option.

PCTLSCALE

requests scale labels for the theoretical quantile axis in percentile units, resulting in a nonlinear axis scale. Tick marks are drawn uniformly across the axis based on the quantile scale. In all other respects, the plot remains the same, and you must specify HREF= values in quantile units. For a true nonlinear axis, use the PCTLAXIS option or use the PROBPLOT statement.

RANKADJ=*value*

specifies the adjustment value added to the ranks in the calculation of theoretical percentiles. By default, RANKADJ=$-\frac{3}{8}$, as recommended by Blom (1958). Refer to Chambers et al. (1983) for additional information.

SCALE=*value* **| EST**

is an alias for the SIGMA= option for plots requested by the BETA, EXPONENTIAL, GAMMA, WEIBULL, and WEIBULL2 options and for the ZETA= option with the LOGNORMAL option. See the entries for the SIGMA= and ZETA= options.

SHAPE=*value* **| EST**

is an alias for the ALPHA= option with the GAMMA option, for the SIGMA= option with the LOGNORMAL option, and for the C= option with the WEIBULL and WEIBULL2 options. See the entries for the ALPHA=, SIGMA=, and C= options.

SIGMA=*value* **| EST**

specifies the parameter σ, where $\sigma > 0$. Alternatively, you can specify SIGMA=EST to request a maximum likelihood estimate for σ_0. The interpretation and use of the SIGMA= option depend on the distribution option with which it is used, as summarized in Table 3.56. Enclose this option in parentheses after the distribution option.

Table 3.56. Uses of the SIGMA= Option

Distribution Option	Use of the SIGMA= Option
BETA EXPONENTIAL GAMMA WEIBULL	THETA=θ_0 and SIGMA=σ_0 request a distribution reference line corresponding to θ_0 and σ_0.
LOGNORMAL	SIGMA=$\sigma_1 \ldots \sigma_n$ requests n quantile plots with shape parameters $\sigma_1 \ldots \sigma_n$. The SIGMA= option must be specified.
NORMAL	MU=μ_0 and SIGMA=σ_0 request a distribution reference line corresponding to μ_0 and σ_0. SIGMA=EST requests a line with σ_0 equal to the sample standard deviation.
WEIBULL2	SIGMA=σ_0 and C=c_0 request a distribution reference line corresponding to σ_0 and c_0.

SLOPE=*value* **| EST**

specifies the slope for a distribution reference line requested with the LOGNORMAL and WEIBULL2 options. Enclose the SLOPE= option in parentheses after the distribution option. When you use the SLOPE= *lognormal-option* with the LOGNORMAL option, you must also specify a threshold parameter value θ_0 with the THETA= *lognormal-option* to request the line. The SLOPE= *lognormal-option* is an alternative to the ZETA= *lognormal-option* for specifying ζ_0, since the slope is equal to $\exp(\zeta_0)$.

When you use the SLOPE= *Weibull2-option* with the WEIBULL2 option, you must also specify a scale parameter value σ_0 with the SIGMA= *Weibull2-option* to request the line. The SLOPE= *Weibull2-option* is an alternative to the C= *Weibull2-option* for specifying c_0, since the slope is equal to $\frac{1}{c_0}$.

For example, the first and second QQPLOT statements produce the same quantile plots and the third and fourth QQPLOT statements produce the same quantile plots:

```
proc univariate data=Measures;
   qqplot Width / lognormal(sigma=2 theta=0 zeta=0);
   qqplot Width / lognormal(sigma=2 theta=0 slope=1);
   qqplot Width / weibull2(sigma=2 theta=0 c=.25);
   qqplot Width / weibull2(sigma=2 theta=0 slope=4);
```

SQUARE

displays the quantile plot in a square frame. By default, the frame is rectangular.

THETA=*value* | EST

specifies the lower threshold parameter θ for plots requested with the BETA, EXPONENTIAL, GAMMA, LOGNORMAL, WEIBULL, and WEIBULL2 options. Enclose the THETA= option in parentheses after a distribution option. When used with the WEIBULL2 option, the THETA= option specifies the known lower threshold θ_0, for which the default is 0. When used with the other distribution options, the THETA= option specifies θ_0 for a distribution reference line; alternatively in this situation, you can specify THETA=EST to request a maximum likelihood estimate for θ_0. To request the line, you must also specify a scale parameter.

THRESHOLD=*value* | EST

is an alias for the THETA= option.

VAXISLABEL=*'label'*

specifies a label for the vertical axis. Labels can have up to 40 characters.

VMINOR=*n*
VM=*n*

specifies the number of minor tick marks between each major tick mark on the vertical axis. Minor tick marks are not labeled. The default is zero.

VREF=*values*

draws reference lines perpendicular to the vertical axis at the values specified. Also see the CVREF=, LVREF=, and VREFCHAR= options.

VREFLABELS='*label1*'. . . '*labeln*'
VREFLABEL='*label1*'. . . '*labeln*'
VREFLAB='*label1*'. . . '*labeln*'

specifies labels for the reference lines requested by the VREF= option. The number of labels must equal the number of reference lines. Enclose each label in quotes. Labels can have up to 16 characters.

VREFLABPOS=*n*

specifies the horizontal position of VREFLABELS= labels. If you specify VREFLABPOS=1, the labels are positioned at the left of the histogram. If you specify VREFLABPOS=2, the labels are positioned at the right of the histogram. By default, VREFLABPOS=1.

W=*n*

specifies the width, in pixels, for a diagonal distribution line. Enclose the W= option in parentheses after the distribution option. By default, W=1.

WAXIS=*n*

specifies the line thickness, in pixels, for the axes and frame. By default, WAXIS=1.

WEIBULL(C=*value* | EST $<$*Weibull-options*$>$**)**
WEIB(C=*value* | EST $<$*Weibull-options*$>$**)**

creates a three-parameter Weibull quantile plot for each value of the required shape parameter c specified by the mandatory C= *Weibull-option*. To create a plot that is based on a maximum likelihood estimate for c, specify C=EST. To obtain a graphical estimate of c, specify a list of values in the C= *Weibull-option*, and select the value that most nearly linearizes the point pattern. To assess the point pattern, add a diagonal distribution reference line corresponding to θ_0 and σ_0 with the THETA= and SIGMA= *Weibull-options*. Alternatively, you can add a line corresponding to estimated values of θ_0 and σ_0 with the *Weibull-options* THETA=EST and SIGMA=EST. Agreement between the reference line and the point pattern indicates that the Weibull distribution with parameters c, θ_0, and σ_0 is a good fit. You can specify the SCALE= *Weibull-option* as an alias for the SIGMA= *Weibull-option* and the THRESHOLD= *Weibull-option* as an alias for the THETA= *Weibull-option*. See Example 3.34.

WEIBULL2$<$**(***Weibull2-options***)**$>$
W2$<$**(***Weibull2-options***)**$>$

creates a two-parameter Weibull quantile plot. You should use the WEIBULL2 option when your data have a *known* lower threshold θ_0, which is 0 by default. To specify the threshold value θ_0, use the THETA= *Weibull2-option*. By default, THETA=0. An advantage of the two-parameter Weibull plot over the three-parameter Weibull plot is that the parameters c and σ can be estimated from the slope and intercept of the point pattern. A disadvantage is that the two-parameter Weibull distribution applies only in situations where the threshold parameter is known. To obtain a graphical estimate of θ_0, specify a list of values for the THETA= *Weibull2-option*, and select the value that most nearly linearizes the point pattern. To assess the point pattern, add a diagonal distribution reference line corresponding to σ_0 and c_0 with the SIGMA= and C= *Weibull2-options*. Alternatively, you can add a distribution reference line corresponding to estimated values of σ_0 and c_0 with the *Weibull2-options* SIGMA=EST and C=EST. Agreement between the reference line and the point pattern indicates that the Weibull distribution with parameters c_0, θ_0, and σ_0 is a good fit. You can specify the SCALE= *Weibull2-option* as an alias for the SIGMA= *Weibull2-option* and the SHAPE= *Weibull2-option* as an alias for the C= *Weibull2-option*. See Example 3.34.

ZETA=*value* | EST

specifies a value for the scale parameter ζ for the lognormal quantile plots requested with the LOGNORMAL option. Enclose the ZETA= *lognormal-option* in parentheses after the LOGNORMAL option. To request a distribution reference line with intercept θ_0 and slope $\exp(\zeta_0)$, specify the THETA=θ_0 and ZETA=ζ_0.

VAR Statement

VAR *variables* **;**

The VAR statement specifies the analysis variables and their order in the results. By default, if you omit the VAR statement, PROC UNIVARIATE analyzes all numeric variables that are not listed in the other statements.

Using the Output Statement with the VAR Statement

You must provide a VAR statement when you use an OUTPUT statement. To store the same statistic for several analysis variables in the OUT= data set, you specify a list of names in the OUTPUT statement. PROC UNIVARIATE makes a one-to-one correspondence between the order of the analysis variables in the VAR statement and the list of names that follow a statistic keyword.

WEIGHT Statement

WEIGHT *variable* **;**

The WEIGHT statement specifies numeric weights for analysis variables in the statistical calculations. The UNIVARIATE procedure uses the values w_i of the WEIGHT variable to modify the computation of a number of summary statistics by assuming that the variance of the ith value x_i of the analysis variable is equal to σ^2/w_i, where σ is an unknown parameter. The values of the WEIGHT variable do not have to be integers and are typically positive. By default, observations with nonpositive or missing values of the WEIGHT variable are handled as follows:*

- If the value is zero, the observation is counted in the total number of observations.
- If the value is negative, it is converted to zero, and the observation is counted in the total number of observations.
- If the value is missing, the observation is excluded from the analysis.

To exclude observations that contain negative and zero weights from the analysis, use EXCLNPWGT. Note that most SAS/STAT procedures, such as PROC GLM, exclude negative and zero weights by default. The weight variable does not change how the procedure determines the range, mode, extreme values, extreme observations, or number of missing values. When you specify a WEIGHT statement, the procedure also computes a weighted standard error and a weighted version of Student's t test. The Student's t test is the only test of location that PROC UNIVARIATE computes when you weight the analysis variables.

When you specify a WEIGHT variable, the procedure uses its values, w_i, to compute weighted versions of the statistics† provided in the Moments table. For example, the procedure computes a weighted mean $\overline{x}_w$ and a weighted variance s_w^2 as

$$\overline{x}_w = \frac{\sum_i w_i x_i}{\sum_i w_i}$$

and

$$s_w^2 = \frac{1}{d} \sum_i w_i (x_i - \overline{x}_w)^2$$

*In Release 6.12 and earlier releases, observations were used in the analysis if and only if the WEIGHT variable value was greater than zero.

†In Release 6.12 and earlier releases, weighted skewness and kurtosis were not computed.

where x_i is the ith variable value. The divisor d is controlled by the VARDEF= option in the PROC UNIVARIATE statement.

The WEIGHT statement does not affect the determination of the mode, extreme values, extreme observations, or the number of missing values of the analysis variables. However, the weights w_i are used to compute weighted percentiles.* The WEIGHT variable has no effect on graphical displays produced with the plot statements.

The CIPCTLDF, CIPCTLNORMAL, LOCCOUNT, NORMAL, ROBUSTSCALE, TRIMMED=, and WINSORIZED= options are not available with the WEIGHT statement.

To compute weighted skewness or kurtosis, use VARDEF=DF or VARDEF=N in the PROC statement.

You cannot specify the HISTOGRAM, PROBPLOT, or QQPLOT statements with the WEIGHT statement.

When you use the WEIGHT statement, consider which value of the VARDEF= option is appropriate. See VARDEF= and the calculation of weighted statistics in for more information.

Details

Missing Values

PROC UNIVARIATE excludes missing values for an analysis variable 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 BY or an ID variable value is missing, PROC UNIVARIATE treats it like any other BY or ID variable value. The missing values form a separate BY group.
- If the FREQ variable value is missing or nonpositive, PROC UNIVARIATE excludes the observation from the analysis.
- If the WEIGHT variable value is missing, PROC UNIVARIATE excludes the observation from the analysis.

PROC UNIVARIATE tabulates the number of missing values and reports this information in the ODS table named Missing Values; see the section “ODS Table Names” on page 309. Before the number of missing values is tabulated, PROC UNIVARIATE excludes observations when

- you use the FREQ statement and the frequencies are nonpositive.
- you use the WEIGHT statement and the weights are missing or nonpositive (you must specify the EXCLNPWGT option).

*In Release 6.12 and earlier releases, the weights did not affect the computation of percentiles and the procedure did not exclude the observations with missing weights from the count of observations.

Rounding

When you specify ROUND=u, PROC UNIVARIATE rounds a variable by using the rounding unit to divide the number line into intervals with midpoints of the form ui, where u is the nonnegative rounding unit and i is an integer. The interval width is u. Any variable value that falls in an interval is rounded to the midpoint of that interval. A variable value that is midway between two midpoints, and is therefore on the boundary of two intervals, rounds to the even midpoint. Even midpoints occur when i is an even integer $(0, \pm 2, \pm 4, \ldots)$.

When ROUND=1 and the analysis variable values are between -2.5 and 2.5, the intervals are as follows:

Table 3.57. Intervals for Rounding When ROUND=1

i	Interval	Midpoint	Left endpt rounds to	Right endpt rounds to
−2	[−2.5,−1.5]	−2	−2	−2
−1	[−1.5,−0.5]	−1	−2	0
0	[−0.5,0.5]	0	0	0
1	[0.5,1.5]	1	0	2
2	[1.5,2.5]	2	2	2

When ROUND=.5 and the analysis variable values are between -1.25 and 1.25, the intervals are as follows:

Table 3.58. Intervals for Rounding When ROUND=0.5

i	Interval	Midpoint	Left endpt rounds to	Right endpt rounds to
−2	[−1.25,−0.75]	−1.0	−1	−1
−1	[−0.75,−0.25]	−0.5	−1	0
0	[−0.25,0.25]	0.0	0	0
1	[0.25,0.75]	0.5	0	1
2	[0.75,1.25]	1.0	1	1

As the rounding unit increases, the interval width also increases. This reduces the number of unique values and decreases the amount of memory that PROC UNIVARIATE needs.

Descriptive Statistics

This section provides computational details for the descriptive statistics that are computed with the PROC UNIVARIATE statement. These statistics can also be saved in the OUT= data set by specifying the keywords listed in Table 3.30 on page 237 in the OUTPUT statement.

Standard algorithms (Fisher 1973) are used to compute the moment statistics. The computational methods used by the UNIVARIATE procedure are consistent with those used by other SAS procedures for calculating descriptive statistics.

The following sections give specific details on a number of statistics calculated by the UNIVARIATE procedure.

Mean

The sample mean is calculated as

$$\bar{x}_w = \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i}$$

where n is the number of nonmissing values for a variable, x_i is the ith value of the variable, and w_i is the weight associated with the ith value of the variable. If there is no WEIGHT variable, the formula reduces to

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

Sum

The sum is calculated as $\sum_{i=1}^{n} w_i x_i$, where n is the number of nonmissing values for a variable, x_i is the ith value of the variable, and w_i is the weight associated with the ith value of the variable. If there is no WEIGHT variable, the formula reduces to $\sum_{i=1}^{n} x_i$.

Sum of the Weights

The sum of the weights is calculated as $\sum_{i=1}^{n} w_i$, where n is the number of nonmissing values for a variable and w_i is the weight associated with the ith value of the variable. If there is no WEIGHT variable, the sum of the weights is n.

Variance

The variance is calculated as

$$\frac{1}{d} \sum_{i=1}^{n} w_i (x_i - \bar{x}_w)^2$$

where n is the number of nonmissing values for a variable, x_i is the ith value of the variable, $\bar{x}_w$ is the weighted mean, w_i is the weight associated with the ith value of the variable, and d is the divisor controlled by the VARDEF= option in the PROC UNIVARIATE statement:

$$d = \begin{cases} n-1 & \text{if VARDEF=DF (default)} \\ n & \text{if VARDEF=N} \\ (\sum_i w_i) - 1 & \text{if VARDEF=WDF} \\ \sum_i w_i & \text{if VARDEF=WEIGHT|WGT} \end{cases}$$

If there is no WEIGHT variable, the formula reduces to

$$\frac{1}{d} \sum_{i=1}^{n} (x_i - \bar{x})^2$$

Standard Deviation

The standard deviation is calculated as

$$s_w = \sqrt{\frac{1}{d}\sum_{i=1}^{n} w_i (x_i - \bar{x}_w)^2}$$

where n is the number of nonmissing values for a variable, x_i is the ith value of the variable, $\bar{x}_w$ is the weighted mean, w_i is the weight associated with the ith value of the variable, and d is the divisor controlled by the VARDEF= option in the PROC UNIVARIATE statement. If there is no WEIGHT variable, the formula reduces to

$$s = \sqrt{\frac{1}{d}\sum_{i=1}^{n} (x_i - \bar{x})^2}$$

Skewness

The sample skewness, which measures the tendency of the deviations to be larger in one direction than in the other, is calculated as follows depending on the VARDEF= option:

Table 3.59. Formulas for Skewness

VARDEF	Formula
DF (default)	$\frac{n}{(n-1)(n-2)} \sum_{i=1}^{n} w_i^{3/2} \left(\frac{x_i - \bar{x}_w}{s_w} \right)^3$
N	$\frac{1}{n} \sum_{i=1}^{n} w_i^{3/2} \left(\frac{x_i - \bar{x}_w}{s_w} \right)^3$
WDF	missing
WEIGHT|WGT	missing

where n is the number of nonmissing values for a variable, x_i is the ith value of the variable, $\bar{x}_w$ is the sample average, s is the sample standard deviation, and w_i is the weight associated with the ith value of the variable. If VARDEF=DF, then n must be greater than 2. If there is no WEIGHT variable, then $w_i = 1$ for all $i = 1, \ldots, n$.

The sample skewness can be positive or negative; it measures the asymmetry of the data distribution and estimates the theoretical skewness $\sqrt{\beta_1} = \mu_3 \mu_2^{-\frac{3}{2}}$, where μ_2 and μ_3 are the second and third central moments. Observations that are normally distributed should have a skewness near zero.

Kurtosis

The sample kurtosis, which measures the heaviness of tails, is calculated as follows depending on the VARDEF= option:

Table 3.60. Formulas for Kurtosis

VARDEF	Formula
DF (default)	$\frac{n(n+1)}{(n-1)(n-2)(n-3)} \sum_{i=1}^{n} w_i^2 \left(\frac{x_i - \bar{x}_w}{s_w} \right)^4 - \frac{3(n-1)^2}{(n-2)(n-3)}$
N	$\frac{1}{n} \sum_{i=1}^{n} w_i^2 \left(\frac{x_i - \bar{x}_w}{s_w} \right)^4 - 3$
WDF	missing
WEIGHT|WGT	missing

where n is the number of nonmissing values for a variable, x_i is the ith value of the variable, $\bar{x}_w$ is the sample average, s_w is the sample standard deviation, and w_i is the weight associated with the ith value of the variable. If VARDEF=DF, then n must be greater than 3. If there is no WEIGHT variable, then $w_i = 1$ for all $i = 1, \ldots, n$.

The sample kurtosis measures the heaviness of the tails of the data distribution. It estimates the adjusted theoretical kurtosis denoted as $\beta_2 - 3$, where $\beta_2 = \frac{\mu_4}{{\mu_2}^2}$, and μ_4 is the fourth central moment. Observations that are normally distributed should have a kurtosis near zero.

Coefficient of Variation (CV)

The coefficient of variation is calculated as

$$CV = \frac{100 \times s_w}{\bar{x}_w}$$

Calculating the Mode

The mode is the value that occurs most often in the data. PROC UNIVARIATE counts repetitions of the values of the analysis variables or, if you specify the ROUND= option, the rounded values. If a tie occurs for the most frequent value, the procedure reports the lowest mode in the table labeled “Basic Statistical Measures” in the statistical output. To list all possible modes, use the MODES option in the PROC UNIVARIATE statement. When no repetitions occur in the data (as with truly continuous data), the procedure does not report the mode. The WEIGHT statement has no effect on the mode. See Example 3.2.

Calculating Percentiles

The UNIVARIATE procedure automatically computes the 1st, 5th, 10th, 25th, 50th, 75th, 90th, 95th, and 99th percentiles (quantiles), as well as the minimum and maximum of each analysis variable. To compute percentiles other than these default percentiles, use the PCTLPTS= and PCTLPRE= options in the OUTPUT statement.

You can specify one of five definitions for computing the percentiles with the PCTLDEF= option. 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. Let the tth percentile be y, set $p = \frac{t}{100}$, and let

$$\begin{array}{rcll} np & = & j+g & \text{when PCTLDEF=1, 2, 3, or 5} \\ (n+1)p & = & j+g & \text{when PCTLDEF=4} \end{array}$$

where j is the integer part of *np*, and g is the fractional part of *np*. Then the PCTLDEF= option defines the tth percentile, y, as described in the following table:

Table 3.61. Percentile Definitions

PCTLDEF	Description	Formula
1	Weighted average at x_{np}	$y = (1-g)x_j + gx_{j+1}$ where x_0 is taken to be x_1
2	Observation numbered closest to *np*	$y = x_j$ if $g < \frac{1}{2}$ $y = x_j$ if $g = \frac{1}{2}$ and *j* is even $y = x_{j+1}$ if $g = \frac{1}{2}$ and *j* is odd $y = x_{j+1}$ if $g > \frac{1}{2}$
3	Empirical distribution function	$y = x_j$ if $g = 0$ $y = x_{j+1}$ if $g > 0$
4	Weighted average aimed at $x_{(n+1)p}$	$y = (1-g)x_j + gx_{j+1}$ where x_{n+1} is taken to be x_n
5	Empirical distribution function with averaging	$y = \frac{1}{2}(x_j + x_{j+1})$ if $g = 0$ $y = x_{j+1}$ if $g > 0$

Weighted Percentiles

When you use a WEIGHT statement, the percentiles are computed differently. The $100p$th weighted percentile y is computed from the empirical distribution function with averaging

$$y = \begin{cases} \frac{1}{2}(x_i + x_{i+1}) & \text{if } \sum_{j=1}^{i} w_j = pW \\ x_{i+1} & \text{if } \sum_{j=1}^{i} w_j < pW < \sum_{j=1}^{i+1} w_j \end{cases}$$

where w_i is the weight associated with x_i, and where $W = \sum_{i=1}^{n} w_i$ is the sum of the weights.

Note that the PCTLDEF= option is not applicable when a WEIGHT statement is used. However, in this case, if all the weights are identical, the weighted percentiles are the same as the percentiles that would be computed without a WEIGHT statement and with PCTLDEF=5.

Confidence Limits for Percentiles

You can use the CIPCTLNORMAL option to request confidence limits for percentiles, assuming the data are normally distributed. These limits are described in Section 4.4.1 of Hahn and Meeker (1991). When $0 < p < \frac{1}{2}$, the two-sided $100(1-\alpha)\%$ confidence limits for the $100p$th percentile are

$$\begin{aligned} \text{lower limit} &= \bar{X} - g'(\tfrac{\alpha}{2}; 1-p, n)s \\ \text{upper limit} &= \bar{X} - g'(1-\tfrac{\alpha}{2}; p, n)s \end{aligned}$$

where n is the sample size. When $\frac{1}{2} \le p < 1$, the two-sided $100(1-\alpha)\%$ confidence limits for the $100p$th percentile are

$$\begin{aligned} \text{lower limit} &= \bar{X} + g'(\tfrac{\alpha}{2}; 1-p, n)s \\ \text{upper limit} &= \bar{X} + g'(1-\tfrac{\alpha}{2}; p, n)s \end{aligned}$$

One-sided $100(1-\alpha)\%$ confidence bounds are computed by replacing $\frac{\alpha}{2}$ by α in the appropriate preceding equation. The factor $g'(\gamma, p, n)$ is related to the noncentral t distribution and is described in Owen and Hua (1977) and Odeh and Owen (1980). See Example 3.10.

You can use the CIPCTLDF option to request distribution-free confidence limits for percentiles. In particular, it is not necessary to assume that the data are normally distributed. These limits are described in Section 5.2 of Hahn and Meeker (1991). The two-sided $100(1-\alpha)\%$ confidence limits for the $100p$th percentile are

$$\begin{aligned} \text{lower limit} &= X_{(l)} \\ \text{upper limit} &= X_{(u)} \end{aligned}$$

where $X_{(j)}$ is the jth order statistic when the data values are arranged in increasing order:

$$X_{(1)} \le X_{(2)} \le \ldots \le X_{(n)}$$

The lower rank l and upper rank u are integers that are symmetric (or nearly symmetric) around $[np] + 1$ where $[np]$ is the integer part of np, and where n is the sample size. Furthermore, l and u are chosen so that $X_{(l)}$ and $X_{(u)}$ are as close to $X_{[n+1]p}$ as possible while satisfying the coverage probability requirement

$$Q(u-1; n, p) - Q(l-1; n, p) \ge 1 - \alpha$$

where $Q(k; n, p)$ is the cumulative binomial probability

$$Q(k; n, p) = \sum_{i=0}^{k} \binom{n}{i} p^i (1-p)^{n-i}$$

In some cases, the coverage requirement cannot be met, particularly when n is small and p is near 0 or 1. To relax the requirement of symmetry, you can specify CIPCTLDF(TYPE = ASYMMETRIC). This option requests symmetric limits when the coverage requirement can be met, and asymmetric limits otherwise.

If you specify CIPCTLDF(TYPE = LOWER), a one-sided $100(1-\alpha)\%$ lower confidence bound is computed as $X_{(l)}$, where l is the largest integer that satisfies the inequality

$$1 - Q(l-1; n, p) \geq 1 - \alpha$$

with $0 < l \leq n$. Likewise, if you specify CIPCTLDF(TYPE = UPPER), a one-sided $100(1-\alpha)\%$ lower confidence bound is computed as $X_{(u)}$, where u is the largest integer that satisfies the inequality

$$Q(u-1; n, p) \geq 1 - \alpha \quad \text{where } 0 < u \leq n$$

Note that confidence limits for percentiles are not computed when a WEIGHT statement is specified. See Example 3.10.

Tests for Location

PROC UNIVARIATE provides three tests for location: Student's t test, the sign test, and the Wilcoxon signed rank test. All three tests produce a test statistic for the null hypothesis that the mean or median is equal to a given value μ_0 against the two-sided alternative that the mean or median is not equal to μ_0. By default, PROC UNIVARIATE sets the value of μ_0 to zero. You can use the MU0= option in the PROC UNIVARIATE statement to specify the value of μ_0. Student's t test is appropriate when the data are from an approximately normal population; otherwise, use nonparametric tests such as the sign test or the signed rank test. For large sample situations, the t test is asymptotically equivalent to a z test. If you use the WEIGHT statement, PROC UNIVARIATE computes only one weighted test for location, the t test. You must use the default value for the VARDEF= option in the PROC statement (VARDEF=DF). See Example 3.12.

You can also use these tests to compare means or medians of *paired data*. Data are said to be paired when subjects or units are matched in pairs according to one or more variables, such as pairs of subjects with the same age and gender. Paired data also occur when each subject or unit is measured at two times or under two conditions. To compare the means or medians of the two times, create an analysis variable that is the difference between the two measures. The test that the mean or the median difference of the variables equals zero is equivalent to the test that the means or medians of the two original variables are equal. Note that you can also carry out these tests using the PAIRED statement in the TTEST procedure; refer to Chapter 77, "The TTEST Procedure," in *SAS/STAT User's Guide*. Also see Example 3.13.

Student's t Test

PROC UNIVARIATE 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. The null hypothesis is that the population mean equals μ_0. When the data values are approximately normally distributed, the probability under the null hypothesis of a t statistic that is as extreme, 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 and the default value of VARDEF=, which is DF, the t statistic is calculated as

$$t_w = \frac{\bar{x}_w - \mu_0}{s_w/\sqrt{\sum_{i=1}^{n} w_i}}$$

where $\bar{x}_w$ 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, 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.

Sign Test

PROC UNIVARIATE calculates the sign test statistic as

$$M = (n^+ - n^-)/2$$

where n^+ is the number of values that are greater than μ_0, and n^- is the number of values that are less than μ_0. Values equal to μ_0 are discarded. Under the null hypothesis that the population median is equal to μ_0, the p-value for the observed statistic M_{obs} is

$$\Pr(|M_{obs}| \geq |M|) = 0.5^{(n_t - 1)} \sum_{j=0}^{min(n^+,n^-)} \binom{n_t}{j}$$

where $n_t = n^+ + n^-$ is the number of x_i values not equal to μ_0.

Note: If n^+ and n^- are equal, the p-value is equal to one.

Wilcoxon Signed Rank Test

The signed rank statistic S is computed as

$$S = \sum_{i:x_i>0} r_i^+ - \frac{n_t(n_t+1)}{4}$$

where r_i^+ is the rank of $|x_i - \mu_0|$ after discarding values of $x_i = \mu_0$, and n_t is the number of x_i values not equal to μ_0. Average ranks are used for tied values.

If $n \leq 20$, the significance of S is computed from the exact distribution of S, where the distribution is a convolution of scaled binomial distributions. When $n > 20$, the significance of S is computed by treating

$$S\sqrt{\frac{n-1}{nV - S^2}}$$

as a Student's t variate with $n-1$ degrees of freedom. V is computed as

$$V = \frac{1}{24}n(n+1)(2n+1) - \frac{1}{48}\sum t_i(t_i+1)(t_i-1)$$

where the sum is over groups tied in absolute value and where t_i is the number of values in the ith group (Iman 1974; Conover 1999). The null hypothesis tested is that the mean (or median) is zero, assuming that the distribution is symmetric. Refer to Lehmann (1998).

Confidence Limits for Parameters of the Normal Distribution

The two-sided $100(1-\alpha)\%$ confidence interval for the mean has upper and lower limits

$$\bar{x} \pm t_{1-\frac{\alpha}{2};n-1}\frac{s}{\sqrt{n}}$$

where $s^2 = \frac{1}{n-1}\sum(x_i - \bar{x})^2$ and $t_{1-\frac{\alpha}{2};n-1}$ is the $(1-\frac{\alpha}{2})$ percentile of the t distribution with $n-1$ degrees of freedom. The one-sided upper $100(1-\alpha)\%$ confidence limit is computed as $\bar{x} + \frac{s}{\sqrt{n}}t_{1-\alpha;n-1}$ and the one-sided lower $100(1-\alpha)\%$ confidence limit is computed as $\bar{x} - \frac{s}{\sqrt{n}}t_{1-\alpha;n-1}$. See Example 3.9.

The two-sided $100(1-\alpha)\%$ confidence interval for the standard deviation has lower and upper limits

$$s\sqrt{\frac{n-1}{\chi^2_{1-\frac{\alpha}{2};n-1}}} \quad \text{and} \quad s\sqrt{\frac{n-1}{\chi^2_{\frac{\alpha}{2};n-1}}}$$

respectively, where $\chi^2_{1-\frac{\alpha}{2};n-1}$ and $\chi^2_{\frac{\alpha}{2};n-1}$ are the $(1-\frac{\alpha}{2})$ and $\frac{\alpha}{2}$ percentiles of the chi-square distribution with $n-1$ degrees of freedom. A one-sided $100(1-\alpha)\%$ confidence limit has lower and upper limits

$$s\sqrt{\frac{n-1}{\chi^2_{1-\alpha;n-1}}} \quad \text{and} \quad s\sqrt{\frac{n-1}{\chi^2_{\alpha;n-1}}}$$

respectively. The $100(1-\alpha)\%$ confidence interval for the variance has upper and lower limits equal to the squares of the corresponding upper and lower limits for the standard deviation. When you use the WEIGHT statement and specify VARDEF=DF in the PROC statement, the $100(1-\alpha)\%$ confidence interval for the weighted mean is

$$\bar{x}_w \pm t_{1-\frac{\alpha}{2}} \frac{s_w}{\sqrt{\sum_{i=1}^{n} w_i}}$$

where $\bar{x}_w$ is the weighted mean, s_w is the weighted standard deviation, w_i is the weight for ith observation, and $t_{1-\frac{\alpha}{2}}$ is the $(1-\frac{\alpha}{2})$ percentile for the t distribution with $n-1$ degrees of freedom.

Robust Estimators

A statistical method is robust if it is insensitive to moderate or even large departures from the assumptions that justify the method. PROC UNIVARIATE provides several methods for robust estimation of location and scale. See Example 3.11.

Winsorized Means

The Winsorized mean is a robust estimator of the location that is relatively insensitive to outliers. The k-times Winsorized mean is calculated as

$$\bar{x}_{wk} = \frac{1}{n}\left((k+1)x_{(k+1)} + \sum_{i=k+2}^{n-k-1} x_{(i)} + (k+1)x_{(n-k)} \right)$$

where n is the number of observations, and $x_{(i)}$ is the ith order statistic when the observations are arranged in increasing order:

$$x_{(1)} \le x_{(2)} \le \ldots \le x_{(n)}$$

The Winsorized mean is computed as the ordinary mean after the k smallest observations are replaced by the $(k+1)$st smallest observation, and the k largest observations are replaced by the $(k+1)$st largest observation.

For data from a symmetric distribution, the Winsorized mean is an unbiased estimate of the population mean. However, the Winsorized mean does not have a normal distribution even if the data are from a normal population.

The Winsorized sum of squared deviations is defined as

$$s_{wk}^2 = (k+1)(x_{(k+1)} - \bar{x}_{wk})^2 + \sum_{i=k+2}^{n-k-1} (x_{(i)} - \bar{x}_{wk})^2 + (k+1)(x_{(n-k)} - \bar{x}_{wk})^2$$

The Winsorized t statistic is given by

$$t_{wk} = \frac{\bar{x}_{wk} - \mu_0}{\mathrm{SE}(\bar{x}_{wk})}$$

where μ_0 denotes the location under the null hypothesis, and the standard error of the Winsorized mean is

$$\mathrm{SE}(\bar{x}_{wk}) = \frac{n-1}{n-2k-1} \times \frac{s_{wk}}{\sqrt{n(n-1)}}$$

When the data are from a symmetric distribution, the distribution of t_{wk} is approximated by a Student's t distribution with $n-2k-1$ degrees of freedom (Tukey and McLaughlin 1963; Dixon and Tukey 1968).

The "Winsorized" $100(1-\frac{\alpha}{2})\%$ confidence interval for the location parameter has upper and lower limits

$$\bar{x}_{wk} \pm t_{1-\frac{\alpha}{2};n-2k-1} \mathrm{SE}(\bar{x}_{wk})$$

where $t_{1-\frac{\alpha}{2};n-2k-1}$ is the $(1-\frac{\alpha}{2})100$th percentile of the Student's t distribution with $n-2k-1$ degrees of freedom.

Trimmed Means

Like the Winsorized mean, the trimmed mean is a robust estimator of the location that is relatively insensitive to outliers. The k-times trimmed mean is calculated as

$$\bar{x}_{tk} = \frac{1}{n-2k} \sum_{i=k+1}^{n-k} x_{(i)}$$

where n is the number of observations, and $x_{(i)}$ is the ith order statistic when the observations are arranged in increasing order:

$$x_{(1)} \le x_{(2)} \le \ldots \le x_{(n)}$$

The trimmed mean is computed after the k smallest and k largest observations are deleted from the sample. In other words, the observations are trimmed at each end.

For a symmetric distribution, the symmetrically trimmed mean is an unbiased estimate of the population mean. However, the trimmed mean does not have a normal distribution even if the data are from a normal population.

A robust estimate of the variance of the trimmed mean t_{tk} can be based on the Winsorized sum of squared deviations s^2_{wk}, which is defined in the section "Winsorized Means" on page 278; refer to Tukey and McLaughlin (1963). This can be used to compute a trimmed t test which is based on the test statistic

$$t_{tk} = \frac{(\bar{x}_{tk} - \mu_0)}{\mathrm{SE}(\bar{x}_{tk})}$$

where the standard error of the trimmed mean is

$$\mathrm{SE}(\bar{x}_{tk}) = \frac{s_{wk}}{\sqrt{(n-2k)(n-2k-1)}}$$

When the data are from a symmetric distribution, the distribution of t_{tk} is approximated by a Student's t distribution with $n - 2k - 1$ degrees of freedom (Tukey and McLaughlin 1963; Dixon and Tukey 1968).

The "trimmed" $100(1-\alpha)\%$ confidence interval for the location parameter has upper and lower limits

$$\bar{x}_{tk} \pm t_{1-\frac{\alpha}{2};n-2k-1}\mathrm{SE}(\bar{x}_{tk})$$

where $t_{1-\frac{\alpha}{2};n-2k-1}$ is the $(1-\frac{\alpha}{2})100$th percentile of the Student's t distribution with $n - 2k - 1$ degrees of freedom.

Robust Estimates of Scale

The sample standard deviation, which is the most commonly used estimator of scale, is sensitive to outliers. Robust scale estimators, on the other hand, remain bounded when a single data value is replaced by an arbitrarily large or small value. The UNIVARIATE procedure computes several robust measures of scale, including the interquartile range, Gini's mean difference G, the median absolute deviation about the median (MAD), Q_n, and S_n. In addition, the procedure computes estimates of the normal standard deviation σ derived from each of these measures.

The interquartile range (IQR) is simply the difference between the upper and lower quartiles. For a normal population, σ can be estimated as IQR/1.34898.

Gini's mean difference is computed as

$$G = \frac{1}{\binom{n}{2}} \sum_{i<j} |x_i - x_j|$$

For a normal population, the expected value of G is $2\sigma/\sqrt{\pi}$. Thus $G\sqrt{\pi}/2$ is a robust estimator of σ when the data are from a normal sample. For the normal distribution, this estimator has high efficiency relative to the usual sample standard deviation, and it is also less sensitive to the presence of outliers.

A very robust scale estimator is the MAD, the median absolute deviation from the median (Hampel 1974), which is computed as

$$\mathrm{MAD} = \mathrm{med}_i(|x_i - \mathrm{med}_j(x_j)|)$$

where the inner median, $\mathrm{med}_j(x_j)$, is the median of the n observations, and the outer median (taken over i) is the median of the n absolute values of the deviations about the inner median. For a normal population, $1.4826\mathrm{MAD}$ is an estimator of σ.

The MAD has low efficiency for normal distributions, and it may not always be appropriate for symmetric distributions. Rousseeuw and Croux (1993) proposed two statistics as alternatives to the MAD. The first is

$$S_n = 1.1926\mathrm{med}_i(\mathrm{med}_j(|x_i - x_j|))$$

where the outer median (taken over i) is the median of the n medians of $|x_i - x_j|$, $j = 1, 2, \ldots, n$. To reduce small-sample bias, $c_{sn}S_n$ is used to estimate σ, where c_{sn} is a correction factor; refer to Croux and Rousseeuw (1992).

The second statistic proposed by Rousseeuw and Croux (1993) is

$$Q_n = 2.219\{|x_i - x_j|; i < j\}_{(k)}$$

where

$$k = \binom{\left[\frac{n}{2}\right] + 1}{2}$$

In other words, Q_n is 2.219 times the kth order statistic of the $\binom{n}{2}$ distances between the data points. The bias-corrected statistic $c_{qn}Q_n$ is used to estimate σ, where c_{qn} is a correction factor; refer to Croux and Rousseeuw (1992).

Creating Line Printer Plots

The PLOTS option in the PROC UNIVARIATE statement provides up to four diagnostic line printer plots to examine the data distribution. These plots are the stem-and-leaf plot or horizontal bar chart, the box plot, the normal probability plot, and the side-by-side box plots. If you specify the WEIGHT statement, PROC UNIVARIATE provides a weighted histogram, a weighted box plot based on the weighted quantiles, and a weighted normal probability plot.

Note that these plots are a legacy feature of the UNIVARIATE procedure in earlier versions of SAS. They predate the addition of the HISTOGRAM, PROBPLOT, and QQPLOT statements, which provide high-resolution graphics displays. Also note that line printer plots requested with the PLOTS option are mainly intended for use with the ODS LISTING destination. See Example 3.5.

Stem-and-Leaf Plot

The first plot in the output is either a stem-and-leaf plot (Tukey 1977) or a horizontal bar chart. If any single interval contains more than 49 observations, the horizontal bar chart appears. Otherwise, the stem-and-leaf plot appears. The stem-and-leaf plot is like a horizontal bar chart in that both plots provide a method to visualize the overall distribution of the data. The stem-and-leaf plot provides more detail because each point in the plot represents an individual data value.

To change the number of stems that the plot displays, use PLOTSIZE= to increase or decrease the number of rows. Instructions that appear below the plot explain how to determine the values of the variable. If no instructions appear, you multiply *Stem.Leaf* by 1 to determine the values of the variable. For example, if the stem value is 10 and the leaf value is 1, then the variable value is approximately 10.1. For the stem-and-leaf plot, the procedure rounds a variable value to the nearest leaf. If the variable value is exactly halfway between two leaves, the value rounds to the nearest leaf with an even integer value. For example, a variable value of 3.15 has a stem value of 3 and a leaf value of 2.

Box Plot

The box plot, also known as a schematic box plot, appears beside the stem-and-leaf plot. Both plots use the same vertical scale. The box plot provides a visual summary of the data and identifies outliers. The bottom and top edges of the box correspond to the sample 25th (Q1) and 75th (Q3) percentiles. The box length is one *interquartile range* (Q3 - Q1). The center horizontal line with asterisk endpoints corresponds to the sample median. The central plus sign (+) corresponds to the sample mean. If the mean and median are equal, the plus sign falls on the line inside the box. The vertical lines that project out from the box, called *whiskers*, extend as far as the data extend, up to a distance of 1.5 interquartile ranges. Values farther away are potential outliers. The procedure identifies the extreme values with a zero or an asterisk (*). If zero appears, the value is between 1.5 and 3 interquartile ranges from the top or bottom edge of the box. If an asterisk appears, the value is more extreme.

Note: To produce box plots using high-resolution graphics, use the BOXPLOT procedure in SAS/STAT software; refer to Chapter 18, “The BOXPLOT Procedure,” in *SAS/STAT User’s Guide*.

Normal Probability Plot

The normal probability plot plots the empirical quantiles against the quantiles of a standard normal distribution. Asterisks (*) indicate the data values. The plus signs (+) provide a straight reference line that is drawn by using the sample mean and standard deviation. If the data are from a normal distribution, the asterisks tend to fall along the reference line. The vertical coordinate is the data value, and the horizontal coordinate is $\Phi^{-1}(v_i)$ where

$$
\begin{aligned}
v_i &= \tfrac{r_i - \frac{3}{8}}{n + \frac{1}{4}} \\
\Phi^{-1}(\cdot) &= \text{inverse of the standard normal distribution function} \\
r_i &= \text{rank of the } i\text{th data value when ordered from smallest to largest} \\
n &= \text{number of nonmissing observations}
\end{aligned}
$$

For a weighted normal probability plot, the ith ordered observation is plotted against $\Phi^{-1}(v_i)$ where

$$
\begin{aligned}
v_i &= \tfrac{(1-\frac{3}{8i})\sum_{j=1}^{i} w_{(j)}}{(1+\frac{1}{4n})\sum_{i=1}^{n} w_i} \\
w_{(j)} &= \text{weight associated with the } j\text{th ordered observation}
\end{aligned}
$$

When each observation has an identical weight, $w_j = w$, the formula for v_i reduces to the expression for v_i in the unweighted normal probability plot:

$$v_i = \frac{i - \frac{3}{8}}{n + \frac{1}{4}}$$

When the value of VARDEF= is WDF or WEIGHT, a reference line with intercept $\hat{\mu}$ and slope $\hat{\sigma}$ is added to the plot. When the value of VARDEF= is DF or N, the slope is $\frac{\hat{\sigma}}{\sqrt{\bar{w}}}$ where $\bar{w} = \frac{\sum_{i=1}^{n} w_i}{n}$ is the average weight.

When each observation has an identical weight and the value of VARDEF= is DF, N, or WEIGHT, the reference line reduces to the usual reference line with intercept $\hat{\mu}$ and slope $\hat{\sigma}$ in the unweighted normal probability plot.

If the data are normally distributed with mean μ, standard deviation σ, and each observation has an identical weight w, then the points on the plot should lie approximately on a straight line. The intercept for this line is μ. The slope is σ when VARDEF= is WDF or WEIGHT, and the slope is $\frac{\sigma}{\sqrt{w}}$ when VARDEF= is DF or N.

Note: To produce probability plots using high-resolution graphics, use the PROBPLOT statement in PROC UNIVARIATE; see the section "PROBPLOT Statement" on page 241.

Side-by-Side Box Plots

When you use a BY statement with the PLOT option, PROC UNIVARIATE produces side-by-side box plots, one for each BY group. The box plots (also known as schematic plots) use a common scale that enables you to compare the data distribution across BY groups. This plot appears after the univariate analyses of all BY groups. Use the NOBYPLOT option to suppress this plot.

Note: To produce side-by-side box plots using high-resolution graphics, use the BOXPLOT procedure in SAS/STAT software; refer to Chapter 18, "The BOXPLOT Procedure," in *SAS/STAT User's Guide*.

Creating High-Resolution Graphics

If your site licenses SAS/GRAPH software, you can use the HISTOGRAM, PROBPLOT, and QQPLOT statements to create high-resolution graphs. The HISTOGRAM statement creates histograms that enable you to examine the data distribution. You can optionally fit families of density curves and superimpose kernel density estimates on the histograms. For additional information about the fitted distributions and kernel density estimates, see the section “Formulas for Fitted Continuous Distributions” on page 288 and the section “Kernel Density Estimates” on page 297.

The PROBPLOT statement creates a probability plot, which compares ordered values of a variable with percentiles of a specified theoretical distribution. The QQPLOT statement creates a quantile-quantile plot, which compares ordered values of a variable with quantiles of a specified theoretical distribution. You can use these plots to determine how well a theoretical distribution models a data distribution.

Note: You can use the CLASS statement with the HISTOGRAM, PROBPLOT, or QQPLOT statements to create comparative histograms, probability plots, or Q-Q plots, respectively.

Using the CLASS Statement to Create Comparative Plots

When you use the CLASS statement with the HISTOGRAM, PROBPLOT, or QQPLOT statement, PROC UNIVARIATE creates comparative histograms, comparative probability plots, or comparative quantile-quantile plots. You can use these plot statements with the CLASS statement to create one-way and two-way comparative plots. When you use one class variable, PROC UNIVARIATE displays an array of component plots (stacked or side-by-side), one for each level of the classification variable. When you use two class variables, PROC UNIVARIATE displays a matrix of component plots, one for each combination of levels of the classification variables. The observations in a given level are referred to collectively as a *cell*.

When you create a one-way comparative plot, the observations in the input data set are sorted by the method specified in the ORDER= option. PROC UNIVARIATE creates a separate plot for the analysis variable values in each level, and arranges these component plots in an array to form the comparative plot with uniform horizontal and vertical axes. See Example 3.15.

When you create a two-way comparative plot, the observations in the input data set are cross-classified according to the values (levels) of these variables. PROC UNIVARIATE creates a separate plot for the analysis variable values in each cell of the cross-classification and arranges these component plots in a matrix to form the comparative plot with uniform horizontal and vertical axes. The levels of the first class variable are the labels for the rows of the matrix, and the levels of the second class variable are the labels for the columns of the matrix. See Example 3.16.

PROC UNIVARIATE determines the layout of a two-way comparative plot by using the order for the first class variable to obtain the order of the rows from top to bottom. Then it applies the order for the second class variable to the observations that correspond to the first row to obtain the order of the columns from left to right. If any columns remain unordered (that is, the categories are unbalanced), PROC UNIVARIATE applies the order for the second class variable to the observations in the second row, and so on, until all the columns have been ordered.

If you associate a label with a variable, PROC UNIVARIATE displays the variable label in the comparative plot and this label is parallel to the column (or row) labels.

Use the MISSING option to treat missing values as valid levels.

To reduce the number of classification levels, use a FORMAT statement to combine variable values.

Positioning the Inset

Positioning the Inset Using Compass Point Values

To position the inset by using a compass point position, specify the value N, NE, E, SE, S, SW, W, or NW with the POSITION= option. The default position of the inset is NW. The following statements produce a histogram to show the position of the inset for the eight compass points:

```
data Score;
   input Student $ PreTest PostTest @@;
   label ScoreChange = 'Change in Test Scores';
   ScoreChange = PostTest - PreTest;
datalines;
Capalleti  94 91  Dubose     51 65
Engles     95 97  Grant      63 75
Krupski    80 75  Lundsford  92 55
Mcbane     75 78  Mullen     89 82
Nguyen     79 76  Patel      71 77
Si         75 70  Tanaka     87 73
;
run;

title 'Test Scores for a College Course';
proc univariate data=Score noprint;
   histogram PreTest / midpoints = 45 to 95 by 10;
   inset n     / cfill=blank
                 header='Position = NW' pos=nw;
   inset mean  / cfill=blank
                 header='Position = N ' pos=n ;
   inset sum   / cfill=blank
                 header='Position = NE' pos=ne;
   inset max   / cfill=blank
                 header='Position = E ' pos=e ;
   inset min   / cfill=blank
                 header='Position = SE' pos=se;
   inset nobs  / cfill=blank
                 header='Position = S ' pos=s ;
   inset range / cfill=blank
                 header='Position = SW' pos=sw;
   inset mode  / cfill=blank
                 header='Position = W ' pos=w ;
   label PreTest = 'Pretest Score';
run;
```

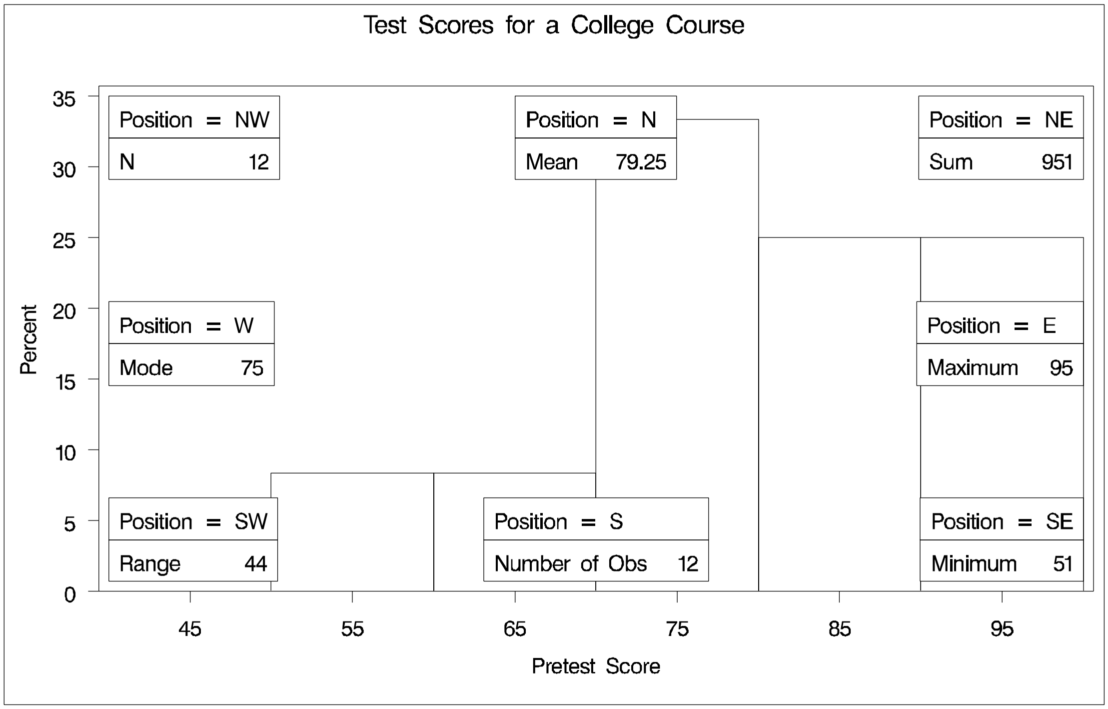


Figure 3.7. Compass Positions for Inset

Positioning the Inset in the Margins

To position the inset in one of the four margins that surround the plot area, specify the value LM, RM, TM, or BM with the POSITION= option.

Locating the Inset in the Margins

Margin positions are recommended if you list a large number of statistics in the INSET statement. If you attempt to display a lengthy inset in the interior of the plot, it is most likely that the inset will collide with the data display.

Positioning the Inset Using Coordinates

To position the inset with coordinates, use POSITION=(x,y). You specify the coordinates in axis data units or in axis percentage units (the default).

If you specify the DATA option immediately following the coordinates, PROC UNIVARIATE positions the inset by using axis data units. For example, the following statements place the bottom left corner of the inset at 45 on the horizontal axis and 10 on the vertical axis:

```
title 'Test Scores for a College Course';
proc univariate data=Score noprint;
   histogram PreTest / midpoints = 45 to 95 by 10;
   inset n / header   = 'Position=(45,10)'
             position = (45,10) data;
run;
```

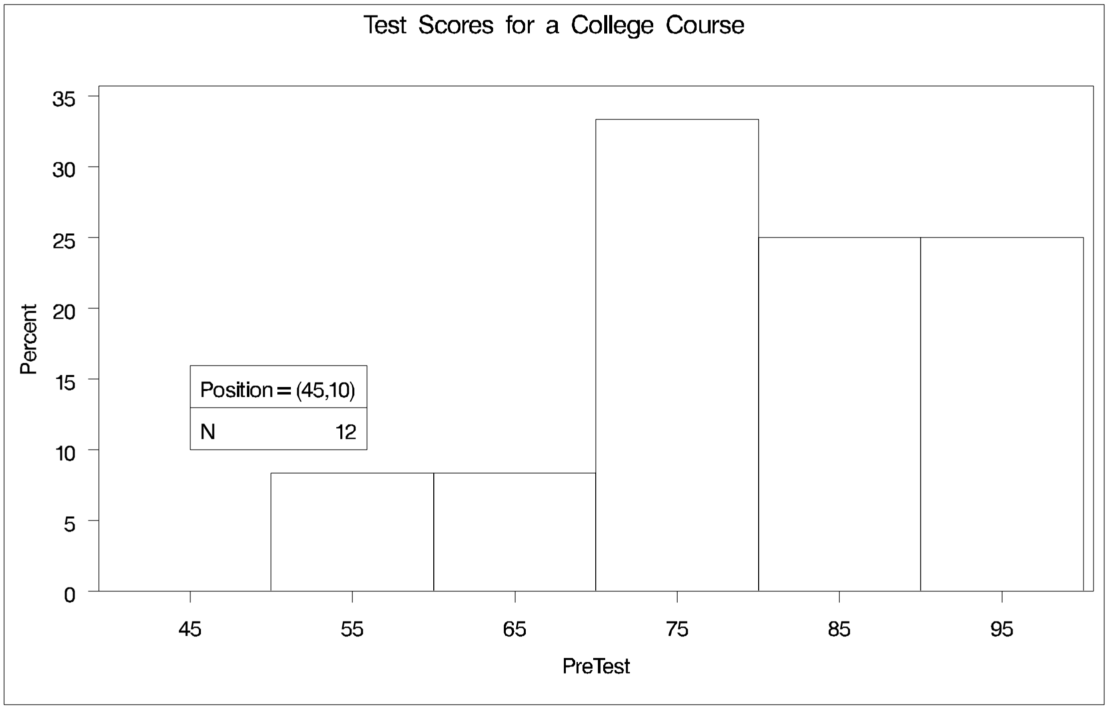


Figure 3.8. Coordinate Position for Inset

By default, the specified coordinates determine the position of the bottom left corner of the inset. To change this reference point, use the REFPOINT= option (see the next example).

If you omit the DATA option, PROC UNIVARIATE positions the inset by using axis percentage units. The coordinates in axis percentage units must be between 0 and 100. The coordinates of the bottom left corner of the display are (0,0), while the upper right corner is (100, 100). For example, the following statements create a histogram and use coordinates in axis percentage units to position the two insets:

```
title 'Test Scores for a College Course';
proc univariate data=Score noprint;
   histogram PreTest / midpoints = 45 to 95 by 10;
   inset min / position = (5,25)
               header   = 'Position=(5,25)'
               refpoint = tl;
   inset max / position = (95,95)
               header   = 'Position=(95,95)'
               refpoint = tr;
   run;
```

The REFPOINT= option determines which corner of the inset to place at the coordinates that are specified with the POSITION= option. The first inset uses REFPOINT=TL, so that the top left corner of the inset is positioned 5% of the way across the horizontal axis and 25% of the way up the vertical axis. The second inset uses REFPOINT=TR, so that the top right corner of the inset is positioned 95% of the way across the horizontal axis and 95% of the way up the vertical axis.

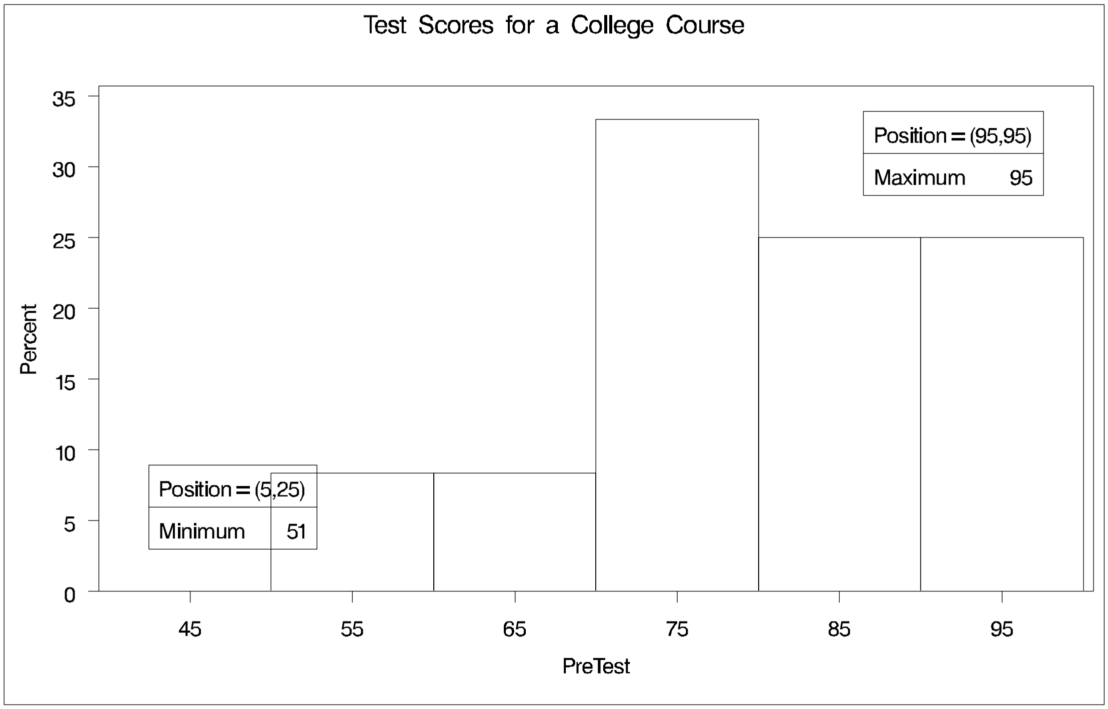


Figure 3.9. Reference Point for Inset

A sample program, univar3.sas, for these examples is available in the SAS Sample Library for Base SAS software.

Formulas for Fitted Continuous Distributions

The following sections provide information on the families of parametric distributions that you can fit with the HISTOGRAM statement. Properties of these distributions are discussed by Johnson, Kotz, and Balakrishnan (1994, 1995).

Beta Distribution

The fitted density function is

$$p(x) = \begin{cases} 100h\% \frac{(x-\theta)^{\alpha-1}(\sigma+\theta-x)^{\beta-1}}{B(\alpha,\beta)\sigma^{(\alpha+\beta-1)}} & \text{for } \theta < x < \theta+\sigma \\ 0 & \text{for } x \leq \theta \text{ or } x \geq \theta+\sigma \end{cases}$$

where $B(\alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$ and

$\theta =$ lower threshold parameter (lower endpoint parameter)
$\sigma =$ scale parameter $(\sigma > 0)$
$\alpha =$ shape parameter $(\alpha > 0)$
$\beta =$ shape parameter $(\beta > 0)$
$h =$ width of histogram interval

Note: This notation is consistent with that of other distributions that you can fit with the HISTOGRAM statement. However, many texts, including Johnson, Kotz, and Balakrishnan (1995), write the beta density function as

$$p(x) = \begin{cases} \frac{(x-a)^{p-1}(b-x)^{q-1}}{B(p,q)(b-a)^{p+q-1}} & \text{for } a < x < b \\ 0 & \text{for } x \leq a \text{ or } x \geq b \end{cases}$$

The two parameterizations are related as follows:

$$\sigma = b - a$$
$$\theta = a$$
$$\alpha = p$$
$$\beta = q$$

The range of the beta distribution is bounded below by a threshold parameter $\theta = a$ and above by $\theta + \sigma = b$. If you specify a fitted beta curve using the BETA option, θ must be less than the minimum data value, and $\theta + \sigma$ must be greater than the maximum data value. You can specify θ and σ with the THETA= and SIGMA= *beta-options* in parentheses after the keyword BETA. By default, $\sigma = 1$ and $\theta = 0$. If you specify THETA=EST and SIGMA=EST, maximum likelihood estimates are computed for θ and σ. However, three- and four-parameter maximum likelihood estimation may not always converge.

In addition, you can specify α and β with the ALPHA= and BETA= *beta-options*, respectively. By default, the procedure calculates maximum likelihood estimates for α and β. For example, to fit a beta density curve to a set of data bounded below by 32 and above by 212 with maximum likelihood estimates for α and β, use the following statement:

```
histogram Length / beta(theta=32 sigma=180);
```

The beta distributions are also referred to as Pearson Type I or II distributions. These include the power-function distribution ($\beta = 1$), the arc-sine distribution ($\alpha = \beta = \frac{1}{2}$), and the generalized arc-sine distributions ($\alpha + \beta = 1$, $\beta \neq \frac{1}{2}$).

You can use the DATA step function BETAINV to compute beta quantiles and the DATA step function PROBBETA to compute beta probabilities.

Exponential Distribution

The fitted density function is

$$p(x) = \begin{cases} \frac{100h\%}{\sigma} \exp(-(\frac{x-\theta}{\sigma})) & \text{for } x \geq \theta \\ 0 & \text{for } x < \theta \end{cases}$$

where

$\theta =$ threshold parameter
$\sigma =$ scale parameter $(\sigma > 0)$
$h =$ width of histogram interval

The threshold parameter θ must be less than or equal to the minimum data value. You can specify θ with the THRESHOLD= *exponential-option*. By default, $\theta = 0$. If you specify THETA=EST, a maximum likelihood estimate is computed for θ. In addition, you can specify σ with the SCALE= *exponential-option*. By default, the procedure calculates a maximum likelihood estimate for σ. Note that some authors define the scale parameter as $\frac{1}{\sigma}$.

The exponential distribution is a special case of both the gamma distribution (with $\alpha = 1$) and the Weibull distribution (with $c = 1$). A related distribution is the extreme value distribution. If $Y = \exp(-X)$ has an exponential distribution, then X has an extreme value distribution.

Gamma Distribution

The fitted density function is

$$p(x) = \begin{cases} \frac{100h\%}{\Gamma(\alpha)\sigma}(\frac{x-\theta}{\sigma})^{\alpha-1}\exp(-(\frac{x-\theta}{\sigma})) & \text{for } x > \theta \\ 0 & \text{for } x \le \theta \end{cases}$$

where

$\theta =$ threshold parameter
$\sigma =$ scale parameter $(\sigma > 0)$
$\alpha =$ shape parameter $(\alpha > 0)$
$h =$ width of histogram interval

The threshold parameter θ must be less than the minimum data value. You can specify θ with the THRESHOLD= *gamma-option*. By default, $\theta = 0$. If you specify THETA=EST, a maximum likelihood estimate is computed for θ. In addition, you can specify σ and α with the SCALE= and ALPHA= *gamma-options*. By default, the procedure calculates maximum likelihood estimates for σ and α.

The gamma distributions are also referred to as Pearson Type III distributions, and they include the chi-square, exponential, and Erlang distributions. The probability density function for the chi-square distribution is

$$p(x) = \begin{cases} \frac{1}{2\Gamma(\frac{\nu}{2})}\left(\frac{x}{2}\right)^{\frac{\nu}{2}-1}\exp(-\frac{x}{2}) & \text{for } x > 0 \\ 0 & \text{for } x \le 0 \end{cases}$$

Notice that this is a gamma distribution with $\alpha = \frac{\nu}{2}$, $\sigma = 2$, and $\theta = 0$. The exponential distribution is a gamma distribution with $\alpha = 1$, and the Erlang distribution is a gamma distribution with α being a positive integer. A related distribution is the Rayleigh distribution. If $R = \frac{\max(X_1,\ldots,X_n)}{\min(X_1,\ldots,X_n)}$ where the X_i's are independent χ^2_ν variables, then $\log R$ is distributed with a χ_ν distribution having a probability density function of

$$p(x) = \begin{cases} \left[2^{\frac{\nu}{2}-1}\Gamma(\frac{\nu}{2})\right]^{-1} x^{\nu-1}\exp(-\frac{x^2}{2}) & \text{for } x > 0 \\ 0 & \text{for } x \le 0 \end{cases}$$

If $\nu = 2$, the preceding distribution is referred to as the Rayleigh distribution.

You can use the DATA step function GAMINV to compute gamma quantiles and the DATA step function PROBGAM to compute gamma probabilities.

Lognormal Distribution

The fitted density function is

$$p(x) = \begin{cases} \frac{100h\%}{\sigma\sqrt{2\pi}(x-\theta)} \exp\left(-\frac{(\log(x-\theta)-\zeta)^2}{2\sigma^2}\right) & \text{for } x > \theta \\ 0 & \text{for } x \leq \theta \end{cases}$$

where

$\theta =$ threshold parameter
$\zeta =$ scale parameter $(-\infty < \zeta < \infty)$
$\sigma =$ shape parameter $(\sigma > 0)$
$h =$ width of histogram interval

The threshold parameter θ must be less than the minimum data value. You can specify θ with the THRESHOLD= *lognormal-option*. By default, $\theta = 0$. If you specify THETA=EST, a maximum likelihood estimate is computed for θ. You can specify ζ and σ with the SCALE= and SHAPE= *lognormal-options*, respectively. By default, the procedure calculates maximum likelihood estimates for these parameters.

Note: The lognormal distribution is also referred to as the S_L distribution in the Johnson system of distributions.

Note: This book uses σ to denote the shape parameter of the lognormal distribution, whereas σ is used to denote the scale parameter of the beta, exponential, gamma, normal, and Weibull distributions. The use of σ to denote the lognormal shape parameter is based on the fact that $\frac{1}{\sigma}(\log(X-\theta)-\zeta)$ has a standard normal distribution if X is lognormally distributed. Based on this relationship, you can use the DATA step function PROBIT to compute lognormal quantiles and the DATA step function PROBNORM to compute probabilities.

Normal Distribution

The fitted density function is

$$p(x) = \frac{100h\%}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}(\frac{x-\mu}{\sigma})^2\right) \quad \text{for } -\infty < x < \infty$$

where

$\mu =$ mean
$\sigma =$ standard deviation $(\sigma > 0)$
$h =$ width of histogram interval

You can specify μ and σ with the MU= and SIGMA= *normal-options*, respectively. By default, the procedure estimates μ with the sample mean and σ with the sample standard deviation.

You can use the DATA step function PROBIT to compute normal quantiles and the DATA step function PROBNORM to compute probabilities.

Note: The normal distribution is also referred to as the S_N distribution in the Johnson system of distributions.

Weibull Distribution

The fitted density function is

$$p(x) = \begin{cases} 100h\% \frac{c}{\sigma}(\frac{x-\theta}{\sigma})^{c-1} \exp(-(\frac{x-\theta}{\sigma})^c) & \text{for } x > \theta \\ 0 & \text{for } x \le \theta \end{cases}$$

where

$\theta =$ threshold parameter
$\sigma =$ scale parameter $(\sigma > 0)$
$c =$ shape parameter $(c > 0)$
$h =$ width of histogram interval

The threshold parameter θ must be less than the minimum data value. You can specify θ with the THRESHOLD= *Weibull-option*. By default, $\theta = 0$. If you specify THETA=EST, a maximum likelihood estimate is computed for θ. You can specify σ and c with the SCALE= and SHAPE= *Weibull-options*, respectively. By default, the procedure calculates maximum likelihood estimates for σ and c.

The exponential distribution is a special case of the Weibull distribution where $c = 1$.

Goodness-of-Fit Tests

When you specify the NORMAL option in the PROC UNIVARIATE statement or you request a fitted parametric distribution in the HISTOGRAM statement, the procedure computes goodness-of-fit tests for the null hypothesis that the values of the analysis variable are a random sample from the specified theoretical distribution. See Example 3.22.

When you specify the NORMAL option, these tests, which are summarized in the output table labeled “Tests for Normality,” include the following:

- Shapiro-Wilk test
- Kolmogorov-Smirnov test
- Anderson-Darling test
- Cramér-von Mises test

The Kolmogorov-Smirnov D statistic, the Anderson-Darling statistic, and the Cramér-von Mises statistic are based on the empirical distribution function (EDF). However, some EDF tests are not supported when certain combinations of the parameters of a specified distribution are estimated. See Table 3.62 on page 296 for a list of the EDF tests available. You determine whether to reject the null hypothesis by examining the p-value that is associated with a goodness-of-fit statistic. When the p-value is less than the predetermined critical value (α), you reject the null hypothesis and conclude that the data did not come from the specified distribution.

If you want to test the normality assumptions for analysis of variance methods, beware of using a statistical test for normality alone. A test's ability to reject the null hypothesis (known as the *power* of the test) increases with the sample size. As the sample size becomes larger, increasingly smaller departures from normality can be detected. Since small deviations from normality do not severely affect the validity of analysis of variance tests, it is important to examine other statistics and plots to make a final assessment of normality. The skewness and kurtosis measures and the plots that are provided by the PLOTS option, the HISTOGRAM statement, the PROBPLOT statement, and the QQPLOT statement can be very helpful. For small sample sizes, power is low for detecting larger departures from normality that may be important. To increase the test's ability to detect such deviations, you may want to declare significance at higher levels, such as 0.15 or 0.20, rather than the often-used 0.05 level. Again, consulting plots and additional statistics will help you assess the severity of the deviations from normality.

Shapiro-Wilk Statistic

If the sample size is less than or equal to 2000 and you specify the NORMAL option, PROC UNIVARIATE computes the Shapiro-Wilk statistic, W (also denoted as W_n to emphasize its dependence on the sample size n). The W statistic is the ratio of the best estimator of the variance (based on the square of a linear combination of the order statistics) to the usual corrected sum of squares estimator of the variance (Shapiro and Wilk 1965). When n is greater than three, the coefficients to compute the linear combination of the order statistics are approximated by the method of Royston (1992). The statistic W is always greater than zero and less than or equal to one $(0 < W \leq 1)$.

Small values of W lead to the rejection of the null hypothesis of normality. The distribution of W is highly skewed. Seemingly large values of W (such as 0.90) may be considered small and lead you to reject the null hypothesis. The method for computing the p-value (the probability of obtaining a W statistic less than or equal to the observed value) depends on n. For $n = 3$, the probability distribution of W is known and is used to determine the p-value. For $n > 4$, a normalizing transformation is computed:

$$Z_n = \begin{cases} (-\log(\gamma - \log(1 - W_n)) - \mu)/\sigma & \text{if } 4 \leq n \leq 11 \\ (\log(1 - W_n) - \mu)/\sigma & \text{if } 12 \leq n \leq 2000 \end{cases}$$

The values of σ, γ, and μ are functions of n obtained from simulation results. Large values of Z_n indicate departure from normality, and since the statistic Z_n has an approximately standard normal distribution, this distribution is used to determine the p-values for $n > 4$.

EDF Goodness-of-Fit Tests

When you fit a parametric distribution, PROC UNIVARIATE provides a series of goodness-of-fit tests based on the empirical distribution function (EDF). The EDF tests offer advantages over traditional chi-square goodness-of-fit test, including improved power and invariance with respect to the histogram midpoints. For a thorough discussion, refer to D'Agostino and Stephens (1986).

The empirical distribution function is defined for a set of n independent observations $X_1, \ldots, X_n$ with a common distribution function $F(x)$. Denote the observations ordered from smallest to largest as $X_{(1)}, \ldots, X_{(n)}$. The empirical distribution function, $F_n(x)$, is defined as

$$\begin{array}{lll} F_n(x) = 0, & x < X_{(1)} & \\ F_n(x) = \frac{i}{n}, & X_{(i)} \le x < X_{(i+1)} & i = 1, \ldots, n-1 \\ F_n(x) = 1, & X_{(n)} \le x & \end{array}$$

Note that $F_n(x)$ is a step function that takes a step of height $\frac{1}{n}$ at each observation. This function estimates the distribution function $F(x)$. At any value x, $F_n(x)$ is the proportion of observations less than or equal to x, while $F(x)$ is the probability of an observation less than or equal to x. EDF statistics measure the discrepancy between $F_n(x)$ and $F(x)$.

The computational formulas for the EDF statistics make use of the probability integral transformation $U = F(X)$. If $F(X)$ is the distribution function of X, the random variable U is uniformly distributed between 0 and 1.

Given n observations $X_{(1)}, \ldots, X_{(n)}$, the values $U_{(i)} = F(X_{(i)})$ are computed by applying the transformation, as discussed in the next three sections.

PROC UNIVARIATE provides three EDF tests:

- Kolmogorov-Smirnov
- Anderson-Darling
- Cramér-von Mises

The following sections provide formal definitions of these EDF statistics.

Kolmogorov D Statistic

The Kolmogorov-Smirnov statistic (D) is defined as

$$D = \text{sup}_x |F_n(x) - F(x)|$$

The Kolmogorov-Smirnov statistic belongs to the supremum class of EDF statistics. This class of statistics is based on the largest vertical difference between $F(x)$ and $F_n(x)$.

The Kolmogorov-Smirnov statistic is computed as the maximum of D^+ and D^-, where D^+ is the largest vertical distance between the EDF and the distribution function when the EDF is greater than the distribution function, and D^- is the largest vertical distance when the EDF is less than the distribution function.

$$\begin{array}{lcl} D^+ & = & \max_i \left(\frac{i}{n} - U_{(i)}\right) \\ D^- & = & \max_i \left(U_{(i)} - \frac{i-1}{n}\right) \\ D & = & \max\left(D^+, D^-\right) \end{array}$$

PROC UNIVARIATE uses a modified Kolmogorov D statistic to test the data against a normal distribution with mean and variance equal to the sample mean and variance.

Anderson-Darling Statistic

The Anderson-Darling statistic and the Cramér-von Mises statistic belong to the quadratic class of EDF statistics. This class of statistics is based on the squared difference $(F_n(x) - F(x))^2$. Quadratic statistics have the following general form:

$$Q = n \int_{-\infty}^{+\infty} (F_n(x) - F(x))^2 \psi(x) dF(x)$$

The function $\psi(x)$ weights the squared difference $(F_n(x) - F(x))^2$.

The Anderson-Darling statistic (A^2) is defined as

$$A^2 = n \int_{-\infty}^{+\infty} (F_n(x) - F(x))^2 \left[F(x)\left(1 - F(x)\right)\right]^{-1} dF(x)$$

Here the weight function is $\psi(x) = \left[F(x)\left(1 - F(x)\right)\right]^{-1}$.

The Anderson-Darling statistic is computed as

$$A^2 = -n - \frac{1}{n} \sum_{i=1}^{n} \left[(2i-1)\log U_{(i)} + (2n+1-2i)\log(1 - U_{(i)})\right]$$

Cramér-von Mises Statistic

The Cramér-von Mises statistic (W^2) is defined as

$$W^2 = n \int_{-\infty}^{+\infty} (F_n(x) - F(x))^2 dF(x)$$

Here the weight function is $\psi(x) = 1$.

The Cramér-von Mises statistic is computed as

$$W^2 = \sum_{i=1}^{n} \left(U_{(i)} - \frac{2i-1}{2n}\right)^2 + \frac{1}{12n}$$

Probability Values of EDF Tests

Once the EDF test statistics are computed, PROC UNIVARIATE computes the associated probability values (p-values). The UNIVARIATE procedure uses internal tables of probability levels similar to those given by D'Agostino and Stephens (1986). If the value is between two probability levels, then linear interpolation is used to estimate the probability value.

The probability value depends upon the parameters that are known and the parameters that are estimated for the distribution. Table 3.62 summarizes different combinations fitted for which EDF tests are available.

Table 3.62. Availability of EDF Tests

Distribution	Parameters			Tests Available
	Threshold	Scale	Shape	
Beta	θ known	σ known	α, β known	all
	θ known	σ known	$\alpha, \beta < 5$ unknown	all
Exponential	θ known,	σ known		all
	θ known	σ unknown		all
	θ unknown	σ known		all
	θ unknown	σ unknown		all
Gamma	θ known	σ known	α known	all
	θ known	σ unknown	α known	all
	θ known	σ known	α unknown	all
	θ known	σ unknown	α unknown	all
	θ unknown	σ known	$\alpha > 1$ known	all
	θ unknown	σ unknown	$\alpha > 1$ known	all
	θ unknown	σ known	$\alpha > 1$ unknown	all
	θ unknown	σ unknown	$\alpha > 1$ unknown	all
Lognormal	θ known	ζ known	σ known	all
	θ known	ζ known	σ unknown	A^2 and W^2
	θ known	ζ unknown	σ known	A^2 and W^2
	θ known	ζ unknown	σ unknown	all
	θ unknown	ζ known	$\sigma < 3$ known	all
	θ unknown	ζ known	$\sigma < 3$ unknown	all
	θ unknown	ζ unknown	$\sigma < 3$ known	all
	θ unknown	ζ unknown	$\sigma < 3$ unknown	all
Normal	θ known	σ known		all
	θ known	σ unknown		A^2 and W^2
	θ unknown	σ known		A^2 and W^2
	θ unknown	σ unknown		all
Weibull	θ known	σ known	c known	all
	θ known	σ unknown	c known	A^2 and W^2
	θ known	σ known	c unknown	A^2 and W^2
	θ known	σ unknown	c unknown	A^2 and W^2
	θ unknown	σ known	$c > 2$ known	all
	θ unknown	σ unknown	$c > 2$ known	all
	θ unknown	σ known	$c > 2$ unknown	all
	θ unknown	σ unknown	$c > 2$ unknown	all

Kernel Density Estimates

You can use the KERNEL option to superimpose kernel density estimates on histograms. Smoothing the data distribution with a kernel density estimate can be more effective than using a histogram to identify features that might be obscured by the choice of histogram bins or sampling variation. A kernel density estimate can also be more effective than a parametric curve fit when the process distribution is multi-modal. See Example 3.23.

The general form of the kernel density estimator is

$$\hat{f}_\lambda(x) = \frac{100h\%}{n\lambda} \sum_{i=1}^{n} K_0\left(\frac{x - x_i}{\lambda}\right)$$

where $K_0(\cdot)$ is the kernel function, λ is the bandwidth, n is the sample size and x_i is the ith observation.

The KERNEL option provides three kernel functions (K_0): normal, quadratic, and triangular. You can specify the function with the K= *kernel-option* in parentheses after the KERNEL option. Values for the K= option are NORMAL, QUADRATIC, and TRIANGULAR (with aliases of N, Q, and T, respectively). By default, a normal kernel is used. The formulas for the kernel functions are

Normal	$K_0(t) = \frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2}t^2)$	for $-\infty < t < \infty$
Quadratic	$K_0(t) = \frac{3}{4}(1 - t^2)$	for $\|t\| \leq 1$
Triangular	$K_0(t) = 1 - \|t\|$	for $\|t\| \leq 1$

The value of λ, referred to as the bandwidth parameter, determines the degree of smoothness in the estimated density function. You specify λ indirectly by specifying a standardized bandwidth c with the C= *kernel-option*. If Q is the interquartile range, and n is the sample size, then c is related to λ by the formula

$$\lambda = cQn^{-\frac{1}{5}}$$

For a specific kernel function, the discrepancy between the density estimator $\hat{f}_\lambda(x)$ and the true density $f(x)$ is measured by the mean integrated square error (MISE):

$$\mathrm{MISE}(\lambda) = \int_x \{E(\hat{f}_\lambda(x)) - f(x)\}^2 dx + \int_x var(\hat{f}_\lambda(x))dx$$

The MISE is the sum of the integrated squared bias and the variance. An approximate mean integrated square error (AMISE) is

$$\mathrm{AMISE}(\lambda) = \frac{1}{4}\lambda^4 \left(\int_t t^2 K(t)dt\right)^2 \int_x \left(f''(x)\right)^2 dx + \frac{1}{n\lambda} \int_t K(t)^2 dt$$

A bandwidth that minimizes AMISE can be derived by treating $f(x)$ as the normal density having parameters μ and σ estimated by the sample mean and standard deviation. If you do not specify a bandwidth parameter or if you specify C=MISE, the bandwidth that minimizes AMISE is used. The value of AMISE can be used to compare different density estimates. For each estimate, the bandwidth parameter c, the kernel function type, and the value of AMISE are reported in the SAS log.

The general kernel density estimates assume that the domain of the density to estimate can take on all values on a real line. However, sometimes the domain of a density is an interval bounded on one or both sides. For example, if a variable Y is a measurement of only positive values, then the kernel density curve should be bounded so that is zero for negative Y values. You can use the LOWER= and UPPER= *kernel-options* to specify the bounds.

The UNIVARIATE procedure uses a reflection technique to create the bounded kernel density curve, as described in Silverman (1986, pp. 30-31). It adds the reflections of the kernel density that are outside the boundary to the bounded kernel estimates. The general form of the bounded kernel density estimator is computed by replacing $K_0\left(\frac{x-x_i}{\lambda}\right)$ in the original equation with

$$\left\{K_0\left(\frac{x-x_i}{\lambda}\right)+K_0\left(\frac{(x-x_l)+(x_i-x_l)}{\lambda}\right)+K_0\left(\frac{(x_u-x)+(x_u-x_i)}{\lambda}\right)\right\}$$

where x_l is the lower bound and x_u is the upper bound.

Without a lower bound, $x_l = -\infty$ and $K_0\left(\frac{(x-x_l)+(x_i-x_l)}{\lambda}\right)$ equals zero. Similarly, without an upper bound, $x_u = \infty$ and $K_0\left(\frac{(x_u-x)+(x_u-x_i)}{\lambda}\right)$ equals zero.

When C=MISE is used with a bounded kernel density, the UNIVARIATE procedure uses a bandwidth that minimizes the AMISE for its corresponding unbounded kernel.

Construction of Quantile-Quantile and Probability Plots

Figure 3.10 illustrates how a Q-Q plot is constructed for a specified theoretical distribution. First, the n nonmissing values of the variable are ordered from smallest to largest:

$$x_{(1)} \le x_{(2)} \le \cdots \le x_{(n)}$$

Then the ith ordered value $x_{(i)}$ is plotted as a point whose y-coordinate is $x_{(i)}$ and whose x-coordinate is $F^{-1}\left(\frac{i-0.375}{n+0.25}\right)$, where $F(\cdot)$ is the specified distribution with zero location parameter and unit scale parameter.

You can modify the adjustment constants -0.375 and 0.25 with the RANKADJ= and NADJ= options. This default combination is recommended by Blom (1958). For additional information, refer to Chambers et al. (1983). Since $x_{(i)}$ is a quantile of the empirical cumulative distribution function (ecdf), a Q-Q plot compares quantiles of the ecdf with quantiles of a theoretical distribution. Probability plots (see the section "PROBPLOT Statement" on page 241) are constructed the same way, except that the x-axis is scaled nonlinearly in percentiles.

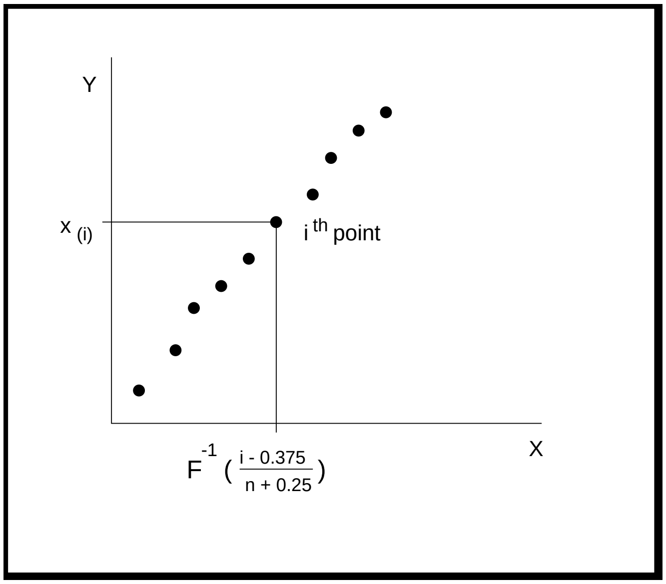


Figure 3.10. Construction of a Q-Q Plot

Interpretation of Quantile-Quantile and Probability Plots

The following properties of Q-Q plots and probability plots make them useful diagnostics of how well a specified theoretical distribution fits a set of measurements:

- If the quantiles of the theoretical and data distributions agree, the plotted points fall on or near the line $y = x$.
- If the theoretical and data distributions differ only in their location or scale, the points on the plot fall on or near the line $y = ax + b$. The slope a and intercept b are visual estimates of the scale and location parameters of the theoretical distribution.

Q-Q plots are more convenient than probability plots for graphical estimation of the location and scale parameters since the x-axis of a Q-Q plot is scaled linearly. On the other hand, probability plots are more convenient for estimating percentiles or probabilities.

There are many reasons why the point pattern in a Q-Q plot may not be linear. Chambers et al. (1983) and Fowlkes (1987) discuss the interpretations of commonly encountered departures from linearity, and these are summarized in Table 3.63.

In some applications, a nonlinear pattern may be more revealing than a linear pattern. However, Chambers et al. (1983) note that departures from linearity can also be due to chance variation.

Table 3.63. Quantile-Quantile Plot Diagnostics

Description of Point Pattern	Possible Interpretation
All but a few points fall on a line	Outliers in the data
Left end of pattern is below the line; right end of pattern is above the line	Long tails at both ends of the data distribution
Left end of pattern is above the line; right end of pattern is below the line	Short tails at both ends of the data distribution
Curved pattern with slope increasing from left to right	Data distribution is skewed to the right
Curved pattern with slope decreasing from left to right	Data distribution is skewed to the left
Staircase pattern (plateaus and gaps)	Data have been rounded or are discrete

When the pattern is linear, you can use Q-Q plots to estimate shape, location, and scale parameters and to estimate percentiles. See Example 3.26 through Example 3.34.

Distributions for Probability and Q-Q Plots

You can use the PROBPLOT and QQPLOT statements to request probability and Q-Q plots that are based on the theoretical distributions summarized in Table 3.64.

Table 3.64. Distributions and Parameters

			Parameters		
Distribution	Density Function $p(x)$	Range	Location	Scale	Shape
Beta	$\frac{(x-\theta)^{\alpha-1}(\theta+\sigma-x)^{\beta-1}}{B(\alpha,\beta)\sigma^{(\alpha+\beta-1)}}$	$\theta < x < \theta + \sigma$	θ	σ	α, β
Exponential	$\frac{1}{\sigma}\exp\left(-\frac{x-\theta}{\sigma}\right)$	$x \geq \theta$	θ	σ	
Gamma	$\frac{1}{\sigma\Gamma(\alpha)}\left(\frac{x-\theta}{\sigma}\right)^{\alpha-1}\exp\left(-\frac{x-\theta}{\sigma}\right)$	$x > \theta$	θ	σ	α
Lognormal (3-parameter)	$\frac{1}{\sigma\sqrt{2\pi}(x-\theta)}\exp\left(-\frac{(\log(x-\theta)-\zeta)^2}{2\sigma^2}\right)$	$x > \theta$	θ	ζ	σ
Normal	$\frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$	all x	μ	σ	
Weibull (3-parameter)	$\frac{c}{\sigma}\left(\frac{x-\theta}{\sigma}\right)^{c-1}\exp\left(-\left(\frac{x-\theta}{\sigma}\right)^{c}\right)$	$x > \theta$	θ	σ	c
Weibull (2-parameter)	$\frac{c}{\sigma}\left(\frac{x-\theta_0}{\sigma}\right)^{c-1}\exp\left(-\left(\frac{x-\theta_0}{\sigma}\right)^{c}\right)$	$x > \theta_0$	θ_0 (known)	σ	c

You can request these distributions with the BETA, EXPONENTIAL, GAMMA, LOGNORMAL, NORMAL, WEIBULL, and WEIBULL2 options, respectively. If you do not specify a distribution option, a normal probability plot or a normal Q-Q plot is created.

The following sections provide details for constructing Q-Q plots that are based on these distributions. Probability plots are constructed similarly except that the horizontal axis is scaled in percentile units.

Beta Distribution

To create the plot, the observations are ordered from smallest to largest, and the ith ordered observation is plotted against the quantile $B_{\alpha\beta}^{-1}\left(\frac{i-0.375}{n+0.25}\right)$, where $B_{\alpha\beta}^{-1}(\cdot)$ is the inverse normalized incomplete beta function, n is the number of nonmissing observations, and α and β are the shape parameters of the beta distribution. In a probability plot, the horizontal axis is scaled in percentile units.

The pattern on the plot for ALPHA=α and BETA=β tends to be linear with intercept θ and slope σ if the data are beta distributed with the specific density function

$$p(x) = \begin{cases} \frac{(x-\theta)^{\alpha-1}(\theta+\sigma-x)^{\beta-1}}{B(\alpha,\beta)\sigma^{(\alpha+\beta-1)}} & \text{for } \theta < x < \theta+\sigma \\ 0 & \text{for } x \leq \theta \text{ or } x \geq \theta+\sigma \end{cases}$$

where $B(\alpha,\beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$ and

$\theta =$ lower threshold parameter
$\sigma =$ scale parameter $(\sigma > 0)$
$\alpha =$ first shape parameter $(\alpha > 0)$
$\beta =$ second shape parameter $(\beta > 0)$

Exponential Distribution

To create the plot, the observations are ordered from smallest to largest, and the ith ordered observation is plotted against the quantile $-\log\left(1-\frac{i-0.375}{n+0.25}\right)$, where n is the number of nonmissing observations. In a probability plot, the horizontal axis is scaled in percentile units.

The pattern on the plot tends to be linear with intercept θ and slope σ if the data are exponentially distributed with the specific density function

$$p(x) = \begin{cases} \frac{1}{\sigma}\exp\left(-\frac{x-\theta}{\sigma}\right) & \text{for } x \geq \theta \\ 0 & \text{for } x < \theta \end{cases}$$

where θ is a threshold parameter, and σ is a positive scale parameter.

Gamma Distribution

To create the plot, the observations are ordered from smallest to largest, and the ith ordered observation is plotted against the quantile $G_{\alpha}^{-1}\left(\frac{i-0.375}{n+0.25}\right)$, where $G_{\alpha}^{-1}(\cdot)$ is the inverse normalized incomplete gamma function, n is the number of nonmissing observations, and α is the shape parameter of the gamma distribution. In a probability plot, the horizontal axis is scaled in percentile units.

The pattern on the plot for ALPHA=α tends to be linear with intercept θ and slope σ if the data are gamma distributed with the specific density function

$$p(x) = \begin{cases} \frac{1}{\sigma\Gamma(\alpha)} \left(\frac{x-\theta}{\sigma}\right)^{\alpha-1} \exp\left(-\frac{x-\theta}{\sigma}\right) & \text{for } x > \theta \\ 0 & \text{for } x \le \theta \end{cases}$$

where

$\theta =$ threshold parameter
$\sigma =$ scale parameter $(\sigma > 0)$
$\alpha =$ shape parameter $(\alpha > 0)$

Lognormal Distribution

To create the plot, the observations are ordered from smallest to largest, and the ith ordered observation is plotted against the quantile $\exp\left(\sigma\Phi^{-1}\left(\frac{i-0.375}{n+0.25}\right)\right)$, where $\Phi^{-1}(\cdot)$ is the inverse cumulative standard normal distribution, n is the number of nonmissing observations, and σ is the shape parameter of the lognormal distribution. In a probability plot, the horizontal axis is scaled in percentile units.

The pattern on the plot for SIGMA=σ tends to be linear with intercept θ and slope $\exp(\zeta)$ if the data are lognormally distributed with the specific density function

$$p(x) = \begin{cases} \frac{1}{\sigma\sqrt{2\pi}(x-\theta)} \exp\left(-\frac{(\log(x-\theta)-\zeta)^2}{2\sigma^2}\right) & \text{for } x > \theta \\ 0 & \text{for } x \le \theta \end{cases}$$

where

$\theta =$ threshold parameter
$\zeta =$ scale parameter
$\sigma =$ shape parameter $(\sigma > 0)$

See Example 3.26 and Example 3.33.

Normal Distribution

To create the plot, the observations are ordered from smallest to largest, and the ith ordered observation is plotted against the quantile $\Phi^{-1}\left(\frac{i-0.375}{n+0.25}\right)$, where $\Phi^{-1}(\cdot)$ is the inverse cumulative standard normal distribution, and n is the number of nonmissing observations. In a probability plot, the horizontal axis is scaled in percentile units.

The point pattern on the plot tends to be linear with intercept μ and slope σ if the data are normally distributed with the specific density function

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \quad \text{for all } x$$

where μ is the mean, and σ is the standard deviation ($\sigma > 0$).

Three-Parameter Weibull Distribution

To create the plot, the observations are ordered from smallest to largest, and the ith ordered observation is plotted against the quantile $\left(-\log\left(1 - \frac{i-0.375}{n+0.25}\right)\right)^{\frac{1}{c}}$, where n is the number of nonmissing observations, and c is the Weibull distribution shape parameter. In a probability plot, the horizontal axis is scaled in percentile units.

The pattern on the plot for C=c tends to be linear with intercept θ and slope σ if the data are Weibull distributed with the specific density function

$$p(x) = \begin{cases} \frac{c}{\sigma}\left(\frac{x-\theta}{\sigma}\right)^{c-1} \exp\left(-\left(\frac{x-\theta}{\sigma}\right)^{c}\right) & \text{for } x > \theta \\ 0 & \text{for } x \le \theta \end{cases}$$

where

$\theta =$ threshold parameter
$\sigma =$ scale parameter $(\sigma > 0)$
$c =$ shape parameter $(c > 0)$

See Example 3.34.

Two-Parameter Weibull Distribution

To create the plot, the observations are ordered from smallest to largest, and the log of the shifted ith ordered observation $x_{(i)}$, denoted by $\log(x_{(i)} - \theta_0)$, is plotted against the quantile $\log\left(-\log\left(1 - \frac{i-0.375}{n+0.25}\right)\right)$, where n is the number of nonmissing observations. In a probability plot, the horizontal axis is scaled in percentile units.

Unlike the three-parameter Weibull quantile, the preceding expression is free of distribution parameters. Consequently, the C= shape parameter is not mandatory with the WEIBULL2 distribution option.

The pattern on the plot for THETA=θ_0 tends to be linear with intercept $\log(\sigma)$ and slope $\frac{1}{c}$ if the data are Weibull distributed with the specific density function

$$p(x) = \begin{cases} \frac{c}{\sigma}\left(\frac{x-\theta_0}{\sigma}\right)^{c-1} \exp\left(-\left(\frac{x-\theta_0}{\sigma}\right)^{c}\right) & \text{for } x > \theta_0 \\ 0 & \text{for } x \le \theta_0 \end{cases}$$

where

$\theta_0 =$ known lower threshold
$\sigma =$ scale parameter $(\sigma > 0)$
$c =$ shape parameter $(c > 0)$

See Example 3.34.

Estimating Shape Parameters Using Q-Q Plots

Some of the distribution options in the PROBPLOT or QQPLOT statements require you to specify one or two shape parameters in parentheses after the distribution keyword. These are summarized in Table 3.65.

You can visually estimate the value of a shape parameter by specifying a list of values for the shape parameter option. A separate plot is produced for each value, and you can then select the value of the shape parameter that produces the most nearly linear point pattern. Alternatively, you can request that the plot be created using an estimated shape parameter. See the entries for the distribution options in the section "Dictionary of Options" on page 245 (for the PROBPLOT statement) and in the section "Dictionary of Options" on page 258 (for the QQPLOT statement).

Note: For Q-Q plots created with the WEIBULL2 option, you can estimate the shape parameter c from a linear pattern using the fact that the slope of the pattern is $\frac{1}{c}$.

Table 3.65. Shape Parameter Options

Distribution Keyword	Mandatory Shape Parameter Option	Range
BETA	ALPHA=α, BETA=β	$\alpha > 0, \beta > 0$
EXPONENTIAL	None	
GAMMA	ALPHA=α	$\alpha > 0$
LOGNORMAL	SIGMA=σ	$\sigma > 0$
NORMAL	None	
WEIBULL	C=c	$c > 0$
WEIBULL2	None	

Estimating Location and Scale Parameters Using Q-Q Plots

If you specify location and scale parameters for a distribution in a PROBPLOT or QQPLOT statement (or if you request estimates for these parameters), a diagonal distribution reference line is displayed on the plot. (An exception is the two-parameter Weibull distribution, for which a line is displayed when you specify or estimate the scale and shape parameters.) Agreement between this line and the point pattern indicates that the distribution with these parameters is a good fit.

When the point pattern on a Q-Q plot is linear, its intercept and slope provide estimates of the location and scale parameters. (An exception to this rule is the two-parameter Weibull distribution, for which the intercept and slope are related to the scale and shape parameters.)

Table 3.66 shows how the specified parameters determine the intercept and slope of the line. The intercept and slope are based on the quantile scale for the horizontal axis, which is used in Q-Q plots.

Table 3.66. Intercept and Slope of Distribution Reference Line

	Parameters			Linear Pattern	
Distribution	Location	Scale	Shape	Intercept	Slope
Beta	θ	σ	α , β	θ	σ
Exponential	θ	σ		θ	σ
Gamma	θ	σ	α	θ	σ
Lognormal	θ	ζ	σ	θ	$\exp(\zeta)$
Normal	μ	σ		μ	σ
Weibull (3-parameter)	θ	σ	c	θ	σ
Weibull (2-parameter)	θ_0 (known)	σ	c	$\log(\sigma)$	$\frac{1}{c}$

For instance, specifying MU=3 and SIGMA=2 with the NORMAL option requests a line with intercept 3 and slope 2. Specifying SIGMA=1 and C=2 with the WEIBULL2 option requests a line with intercept $\log(1) = 0$ and slope $\frac{1}{2}$. On a probability plot with the LOGNORMAL and WEIBULL2 options, you can specify the slope directly with the SLOPE= option. That is, for the LOGNORMAL option, specifying THETA= θ_0 and SLOPE=$\exp(\zeta_0)$ displays the same line as specifying THETA= θ_0 and ZETA= ζ_0. For the WEIBULL2 option, specifying SIGMA= σ_0 and SLOPE= $\frac{1}{c_0}$ displays the same line as specifying SIGMA= σ_0 and C= c_0.

Estimating Percentiles Using Q-Q Plots

There are two ways to estimate percentiles from a Q-Q plot:

- Specify the PCTLAXIS option, which adds a percentile axis opposite the theoretical quantile axis. The scale for the percentile axis ranges between 0 and 100 with tick marks at percentile values such as 1, 5, 10, 25, 50, 75, 90, 95, and 99.
- Specify the PCTLSCALE option, which relabels the horizontal axis tick marks with their percentile equivalents but does not alter their spacing. For example, on a normal Q-Q plot, the tick mark labeled "0" is relabeled as "50" since the 50th percentile corresponds to the zero quantile.

You can also estimate percentiles using probability plots created with the PROBPLOT statement. See Example 3.32.

Input Data Sets

DATA= Data Set

The DATA= data set provides the set of variables that are analyzed. The UNIVARIATE procedure must have a DATA= data set. If you do not specify one with the DATA= option in the PROC UNIVARIATE statement, the procedure uses the last data set created.

ANNOTATE= Data Sets

You can add features to plots by specifying ANNOTATE= data sets either in the PROC UNIVARIATE statement or in individual plot statements.

Information contained in an ANNOTATE= data set specified in the PROC UNIVARIATE statement is used for all plots produced in a given PROC step; this is a "global" ANNOTATE= data set. By using this global data set, you can keep information common to all high-resolution plots in one data set.

Information contained in the ANNOTATE= data set specified in a plot statement is used only for plots produced by that statement; this is a "local" ANNOTATE= data set. By using this data set, you can add statement-specific features to plots. For example, you can add different features to plots produced by the HISTOGRAM and QQPLOT statements by specifying an ANNOTATE= data set in each plot statement.

You can specify an ANNOTATE= data set in the PROC UNIVARIATE statement and in plot statements. This enables you to add some features to all plots and also add statement-specific features to plots. See Example 3.25.

OUT= Output Data Set in the OUTPUT Statement

PROC UNIVARIATE creates an OUT= data set for each OUTPUT statement. This data set contains an observation for each combination of levels of the variables in the BY statement, or a single observation if you do not specify a BY statement. Thus the number of observations in the new data set corresponds to the number of groups for which statistics are calculated. Without a BY statement, the procedure computes statistics and percentiles by using all the observations in the input data set. With a BY statement, the procedure computes statistics and percentiles by using the observations within each BY group.

The variables in the OUT= data set are as follows:

- BY statement variables. The values of these variables match the values in the corresponding BY group in the DATA= data set and indicate which BY group each observation summarizes.
- variables created by selecting statistics in the OUTPUT statement. The statistics are computed using all the nonmissing data, or they are computed for each BY group if you use a BY statement.
- variables created by requesting new percentiles with the PCTLPTS= option. The names of these new variables depend on the values of the PCTLPRE= and PCTLNAME= options.

If the output data set contains a percentile variable or a quartile variable, the percentile definition assigned with the PCTLDEF= option in the PROC UNIVARIATE statement is recorded in the output data set label. See Example 3.8.

The following table lists variables available in the OUT= data set.

Table 3.67. Variables Available in the OUT= Data Set

Variable Name	Description
Descriptive Statistics	
CSS	Sum of squares corrected for the mean
CV	Percent coefficient of variation
KURTOSIS	Measurement of the heaviness of tails
MAX	Largest (maximum) value
MEAN	Arithmetic mean
MIN	Smallest (minimum) value
MODE	Most frequent value (if not unique, the smallest mode)
N	Number of observations on which calculations are based
NMISS	Number of missing observations
NOBS	Total number of observations
RANGE	Difference between the maximum and minimum values
SKEWNESS	Measurement of the tendency of the deviations to be larger in one direction than in the other
STD	Standard deviation
STDMEAN	Standard error of the mean
SUM	Sum
SUMWGT	Sum of the weights

Table 3.67. (continued)

Variable Name	Description
USS	Uncorrected sum of squares
VAR	Variance
Quantile Statistics	
MEDIAN|P50	Middle value (50th percentile)
P1	1st percentile
P5	5th percentile
P10	10th percentile
P90	90th percentile
P95	95th percentile
P99	99th percentile
Q1|P25	Lower quartile (25th percentile)
Q3|P75	Upper quartile (75th percentile)
QRANGE	Difference between the upper and lower quartiles (also known as the inner quartile range)
Robust Statistics	
GINI	Gini's mean difference
MAD	Median absolute difference
QN	2nd variation of median absolute difference
SN	1st variation of median absolute difference
STD_GINI	Standard deviation for Gini's mean difference
STD_MAD	Standard deviation for median absolute difference
STD_QN	Standard deviation for the second variation of the median absolute difference
STD_QRANGE	Estimate of the standard deviation, based on interquartile range
STD_SN	Standard deviation for the first variation of the median absolute difference
Hypothesis Test Statistics	
MSIGN	Sign statistic
NORMAL	Test statistic for normality. If the sample size is less than or equal to 2000, this is the Shapiro-Wilk W statistic. Otherwise, it is the Kolmogorov D statistic.
PROBM	Probability of a greater absolute value for the sign statistic
PROBN	Probability that the data came from a normal distribution
PROBS	Probability of a greater absolute value for the signed rank statistic
PROBT	Two-tailed p-value for Student's t statistic with $n - 1$ degrees of freedom
SIGNRANK	Signed rank statistic
T	Student's t statistic to test the null hypothesis that the population mean is equal to μ_0

OUTHISTOGRAM= Output Data Set

You can create an OUTHISTOGRAM= data set with the HISTOGRAM statement. This data set contains information about histogram intervals. Since you can specify multiple HISTOGRAM statements with the UNIVARIATE procedure, you can create multiple OUTHISTOGRAM= data sets.

An OUTHISTOGRAM= data set contains a group of observations for each variable in the HISTOGRAM statement. The group contains an observation for each interval of the histogram, beginning with the leftmost interval that contains a value of the variable and ending with the rightmost interval that contains a value of the variable. These intervals will not necessarily coincide with the intervals displayed in the histogram since the histogram may be padded with empty intervals at either end. If you superimpose one or more fitted curves on the histogram, the OUTHISTOGRAM= data set contains multiple groups of observations for each variable (one group for each curve). If you use a BY statement, the OUTHISTOGRAM= data set contains groups of observations for each BY group. ID variables are not saved in an OUTHISTOGRAM= data set.

By default, an OUTHISTOGRAM= data set contains the _MIDPT_ variable, whose values identify histogram intervals by their midpoints. When the ENDPOINTS= or NENDPOINTS option is specified, intervals are identified by endpoint values instead. If the RTINCLUDE option is specified, the _MAXPT_ variable contains upper endpoint values. Otherwise, the _MINPT_ variable contains lower endpoint values. See Example 3.18.

Table 3.68. Variables in the OUTHISTOGRAM= Data Set

Variable	Description
CURVE	Name of fitted distribution (if requested in HISTOGRAM statement)
EXPPCT	Estimated percent of population in histogram interval determined from optional fitted distribution
MAXPT	Upper endpoint of histogram interval
MIDPT	Midpoint of histogram interval
MINPT	Lower endpoint of histogram interval
OBSPCT	Percent of variable values in histogram interval
VAR	Variable name

Tables for Summary Statistics

By default, PROC UNIVARIATE produces ODS tables of moments, basic statistical measures, tests for location, quantiles, and extreme observations. You must specify options in the PROC UNIVARIATE statement to request other statistics and tables. The CIBASIC option produces a table that displays confidence limits for the mean, standard deviation, and variance. The CIPCTLDF and CIPCTLNORMAL options request tables of confidence limits for the quantiles. The LOCCOUNT option requests a table that shows the number of values greater than, not equal to, and less than the value of MU0=. The FREQ option requests a table of frequencies counts.

The NEXTRVAL= option requests a table of extreme values. The NORMAL option requests a table with tests for normality.

The TRIMMED=, WINSORIZED=, and ROBUSTCALE options request tables with robust estimators. The table of trimmed or Winsorized means includes the percentage and the number of observations that are trimmed or Winsorized at each end, the mean and standard error, confidence limits, and the Student's t test. The table with robust measures of scale includes interquartile range, Gini's mean difference *G*, *MAD*, Q_n, and S_n, with their corresponding estimates of σ.

See the section "ODS Table Names" on page 309 for the names of ODS tables created by PROC UNIVARIATE.

ODS Table Names

PROC UNIVARIATE 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.

Table 3.69. ODS Tables Produced with the PROC UNIVARIATE Statement

ODS Table Name	Description	Option
BasicIntervals	Confidence intervals for mean, standard deviation, variance	CIBASIC
BasicMeasures	Measures of location and variability	Default
ExtremeObs	Extreme observations	Default
ExtremeValues	Extreme values	NEXTRVAL=
Frequencies	Frequencies	FREQ
LocationCounts	Counts used for sign test and signed rank test	LOCCOUNT
MissingValues	Missing values	Default, if missing values exist
Modes	Modes	MODES
Moments	Sample moments	Default
Plots	Line printer plots	PLOTS
Quantiles	Quantiles	Default
RobustScale	Robust measures of scale	ROBUSTSCALE
SSPlots	Line printer side-by-side box plots	PLOTS (with BY statement)
TestsForLocation	Tests for location	Default
TestsForNormality	Tests for normality	NORMALTEST
TrimmedMeans	Trimmed means	TRIMMED=
WinsorizedMeans	Winsorized means	WINSORIZED=

Table 3.70. ODS Tables Produced with the HISTOGRAM Statement

ODS Table Name	Description	Option
Bins	Histogram bins	MIDPERCENTS secondary option
FitQuantiles	Quantiles of fitted distribution	Any distribution option
GoodnessOfFit	Goodness-of-fit tests for fitted distribution	Any distribution option
HistogramBins	Histogram bins	MIDPERCENTS option
ParameterEstimates	Parameter estimates for fitted distribution	Any distribution option

ODS Tables for Fitted Distributions

If you request a fitted parametric distribution with a HISTOGRAM statement, PROC UNIVARIATE creates a summary that is organized into the ODS tables described in this section.

Parameters

The ParameterEstimates table lists the estimated (or specified) parameters for the fitted curve as well as the estimated mean and estimated standard deviation. See "Formulas for Fitted Continuous Distributions" on page 288.

EDF Goodness-of-Fit Tests

When you fit a parametric distribution, the HISTOGRAM statement provides a series of goodness-of-fit tests based on the empirical distribution function (EDF). See "EDF Goodness-of-Fit Tests" on page 294. These are displayed in the GoodnessOfFit table.

Histogram Intervals

The Bins table is included in the summary only if you specify the MIDPERCENTS option in parentheses after the distribution option. This table lists the midpoints for the histogram bins along with the observed and estimated percentages of the observations that lie in each bin. The estimated percentages are based on the fitted distribution.

If you specify the MIDPERCENTS option without requesting a fitted distribution, the HistogramBins table is included in the summary. This table lists the interval midpoints with the observed percent of observations that lie in the interval. See the entry for the MIDPERCENTS option on page 225.

Quantiles

The FitQuantiles table lists observed and estimated quantiles. You can use the PERCENTS= option to specify the list of quantiles in this table. See the entry for the PERCENTS= option on page 227. By default, the table lists observed and estimated quantiles for the 1, 5, 10, 25, 50, 75, 90, 95, and 99 percent of a fitted parametric distribution.

Computational Resources

Because the UNIVARIATE procedure computes quantile statistics, it requires additional memory to store a copy of the data in memory. By default, the MEANS, SUMMARY, and TABULATE procedures require less memory because they do not automatically compute quantiles. These procedures also provide an option to use a new fixed-memory quantiles estimation method that is usually less memory intensive.

In the UNIVARIATE procedure, the only factor that limits the number of variables that you can analyze is the computer resources that are available. The amount of temporary storage and CPU time required depends on the statements and the options that you specify. To calculate the computer resources the procedure needs, let

N	be the number of observations in the data set
V	be the number of variables in the VAR statement
U_i	be the number of unique values for the ith variable

Then the minimum memory requirement in bytes to process all variables is $M = 24\sum_i U_i$. If M bytes are not available, PROC UNIVARIATE must process the data multiple times to compute all the statistics. This reduces the minimum memory requirement to $M = 24\max(U_i)$.

Using the ROUND= option reduces the number of unique values (U_i), thereby reducing memory requirements. The ROBUSTSCALE option requires $40U_i$ bytes of temporary storage.

Several factors affect the CPU time:

- The time to create V tree structures to internally store the observations is proportional to $NV\log(N)$.
- The time to compute moments and quantiles for the ith variable is proportional to U_i.
- The time to compute the NORMAL option test statistics is proportional to N.
- The time to compute the ROBUSTSCALE option test statistics is proportional to $U_i\log(U_i)$.
- The time to compute the exact significance level of the sign rank statistic may increase when the number of nonzero values is less than or equal to 20.

Each of these factors has a different constant of proportionality. For additional information on optimizing CPU performance and memory usage, see the SAS documentation for your operating environment.

Examples

Example 3.1. Computing Descriptive Statistics for Multiple Variables

This example computes univariate statistics for two variables. The following statements create the data set BPressure, which contains the systolic (Systolic) and diastolic (Diastolic) blood pressure readings for 22 patients:

```
data BPressure;
   length PatientID $2;
   input PatientID $ Systolic Diastolic @@;
   datalines;
CK 120 50  SS 96  60 FR 100 70
CP 120 75  BL 140 90 ES 120 70
CP 165 110 JI 110 40 MC 119 66
FC 125 76  RW 133 60 KD 108 54
DS 110 50  JW 130 80 BH 120 65
JW 134 80  SB 118 76 NS 122 78
GS 122 70  AB 122 78 EC 112 62
HH 122 82
;
run;
```

The following statements produce descriptive statistics and quantiles for the variables Systolic and Diastolic:

```
title 'Systolic and Diastolic Blood Pressure';
ods select BasicMeasures Quantiles;
proc univariate data=BPressure;
   var Systolic Diastolic;
run;
```

The ODS SELECT statement restricts the output, which is shown in Output 3.1.1, to the “BasicMeasures” and “Quantiles” tables; see the section “ODS Table Names” on page 309. You use the PROC UNIVARIATE statement to request univariate statistics for the variables listed in the VAR statement, which specifies the analysis variables and their order in the output. Formulas for computing the statistics in the “BasicMeasures” table are provided in the section “Descriptive Statistics” on page 269. The quantiles are calculated using *Definition 5*, which is the default definition; see the section “Calculating Percentiles” on page 273.

A sample program, uniex01.sas, for this example is available in the SAS Sample Library for Base SAS software.

Output 3.1.1. Display Basic Measures and Quantiles

```
                Systolic and Diastolic Blood Pressure

                       The UNIVARIATE Procedure
                         Variable:  Systolic

                      Basic Statistical Measures

            Location                    Variability

        Mean     121.2727     Std Deviation           14.28346
        Median   120.0000     Variance               204.01732
        Mode     120.0000     Range                   69.00000
                              Interquartile Range     13.00000

    NOTE: The mode displayed is the smallest of 2 modes with a count of 4.


                       Quantiles (Definition 5)

                        Quantile      Estimate

                        100% Max           165
                        99%                165
                        95%                140
                        90%                134
                        75% Q3             125
                        50% Median         120
                        25% Q1             112
                        10%                108
                        5%                 100
                        1%                  96
                        0% Min              96



                Systolic and Diastolic Blood Pressure

                       The UNIVARIATE Procedure
                         Variable:  Diastolic

                      Basic Statistical Measures

            Location                    Variability

        Mean     70.09091     Std Deviation           15.16547
        Median   70.00000     Variance               229.99134
        Mode     70.00000     Range                   70.00000
                              Interquartile Range     18.00000


                       Quantiles (Definition 5)

                        Quantile      Estimate

                        100% Max           110
                        99%                110
                        95%                 90
                        90%                 82
                        75% Q3              78
                        50% Median          70
                        25% Q1              60
                        10%                 50
                        5%                  50
                        1%                  40
                        0% Min              40
```

Example 3.2. Calculating Modes

An instructor is interested in calculating all the modes of the scores on a recent exam. The following statements create a data set named Exam, which contains the exam scores in the variable Score:

```
data Exam;
   label Score = 'Exam Score';
   input Score @@;
   datalines;
81 97 78 99 77 81 84 86 86 97
85 86 94 76 75 42 91 90 88 86
97 97 89 69 72 82 83 81 80 81
;
run;
```

The following statements use the MODES option to request a table of all possible modes:

```
title 'Table of Modes for Exam Scores';
ods select Modes;
proc univariate data=Exam modes;
   var Score;
run;
```

The ODS SELECT statement restricts the output to the "Modes" table; see the section "ODS Table Names" on page 309.

Output 3.2.1. Table of Modes Display

```
                  Table of Modes for Exam Scores

                     The UNIVARIATE Procedure
                  Variable:  Score  (Exam Score)

                              Modes

                          Mode     Count

                            81         4
                            86         4
                            97         4
```

By default, when the MODES option is used, and there is more than one mode, the lowest mode is displayed in the "BasicMeasures" table. The following statements illustrate the default behavior:

```
title 'Default Output';
ods select BasicMeasures;
proc univariate data=Exam;
   var Score;
run;
```

Output 3.2.2. Default Output (Without MODES Option)

```
                          Default Output

                     The UNIVARIATE Procedure
                  Variable:  Score  (Exam Score)

                    Basic Statistical Measures

        Location                    Variability

    Mean     83.66667     Std Deviation           11.08069
    Median   84.50000     Variance               122.78161
    Mode     81.00000     Range                   57.00000
                          Interquartile Range     10.00000

NOTE: The mode displayed is the smallest of 3 modes with a count of 4.
```

The default output displays a mode of 81 and includes a note regarding the number of modes; the modes 86 and 97 are not displayed. The ODS SELECT statement restricts the output to the “BasicMeasures” table; see the section “ODS Table Names” on page 309.

A sample program, uniex02.sas, for this example is available in the SAS Sample Library for Base SAS software.

Example 3.3. Identifying Extreme Observations and Extreme Values

This example, which uses the data set BPressure introduced in Example 3.1, illustrates how to produce a table of the extreme observations and a table of the extreme values in a data set. The following statements generate the “Extreme Observations” tables for Systolic and Diastolic, which enable you to identify the extreme observations for each variable:

```
title 'Extreme Blood Pressure Observations';
ods select ExtremeObs;
proc univariate data=BPressure;
   var Systolic Diastolic;
   id PatientID;
run;
```

The ODS SELECT statement restricts the output to the “ExtremeObs” table; see the section “ODS Table Names” on page 309. The ID statement requests that the extreme observations are to be identified using the value of PatientID as well as the observation number. By default, the five lowest and five highest observations are displayed. You can use the NEXTROBS= option to request a different number of extreme observations.

Output 3.3.1 shows that the patient identified as ‘CP’ (Observation 7) has the highest values for both Systolic and Diastolic. To visualize extreme observations, you can create histograms; see Example 3.14.

Output 3.3.1. Blood Pressure Extreme Observations

```
                      Extreme Blood Pressure Observations

                            The UNIVARIATE Procedure
                              Variable:  Systolic

                              Extreme Observations

             ---------Lowest---------        ---------Highest--------

                     Patient                          Patient
             Value   ID            Obs        Value   ID            Obs

                96   SS              2          130   JW             14
               100   FR              3          133   RW             11
               108   KD             12          134   JW             16
               110   DS             13          140   BL              5
               110   JI              8          165   CP              7


                      Extreme Blood Pressure Observations

                            The UNIVARIATE Procedure
                              Variable:  Diastolic

                              Extreme Observations

             ---------Lowest---------        ---------Highest--------

                     Patient                          Patient
             Value   ID            Obs        Value   ID            Obs

                40   JI              8           80   JW             14
                50   DS             13           80   JW             16
                50   CK              1           82   HH             22
                54   KD             12           90   BL              5
                60   RW             11          110   CP              7
```

The following statements generate the “Extreme Values” tables for Systolic and Diastolic, which tabulate the tails of the distributions:

```
title 'Extreme Blood Pressure Values';
ods select ExtremeValues;
proc univariate data=BPressure nextrval=5;
   var Systolic Diastolic;
run;
```

The ODS SELECT statement restricts the output to the “ExtremeValues” table; see the section “ODS Table Names” on page 309. The NEXTRVAL= option specifies the number of extreme values at each end of the distribution to be shown in the tables in Output 3.3.2.

Output 3.3.2 shows that the values `78` and `80` occurred twice for Diastolic and the maximum of Diastolic is `110`. Note that Output 3.3.1 displays the value of `80` twice for Diastolic because there are two observations with that value. In Output 3.3.2, the value `80` is only displayed once.

Output 3.3.2. Blood Pressure Extreme Values

```
                     Extreme Blood Pressure Values

                        The UNIVARIATE Procedure
                          Variable:  Systolic

                             Extreme Values

        ---------Lowest--------   --------Highest--------

         Order    Value     Freq    Order    Value     Freq

             1       96        1       11      130        1
             2      100        1       12      133        1
             3      108        1       13      134        1
             4      110        2       14      140        1
             5      112        1       15      165        1


                     Extreme Blood Pressure Values

                        The UNIVARIATE Procedure
                          Variable:  Diastolic

                             Extreme Values

        ---------Lowest--------   --------Highest--------

         Order    Value     Freq    Order    Value     Freq

             1       40        1       11       78        2
             2       50        2       12       80        2
             3       54        1       13       82        1
             4       60        2       14       90        1
             5       62        1       15      110        1
```

A sample program, uniex01.sas, for this example is available in the SAS Sample Library for Base SAS software.

Example 3.4. Creating a Frequency Table

An instructor is interested in creating a frequency table of score changes between a pair of tests given in one of his college courses. The data set Score contains test scores for his students who took a pretest and a posttest on the same material. The variable ScoreChange contains the difference between the two test scores. The following statements create the data set:

```
data Score;
   input Student $ PreTest PostTest @@;
   label ScoreChange = 'Change in Test Scores';
   ScoreChange = PostTest - PreTest;
   datalines;
Capalleti  94 91  Dubose     51 65
Engles     95 97  Grant      63 75
Krupski    80 75  Lundsford  92 55
Mcbane     75 78  Mullen     89 82
Nguyen     79 76  Patel      71 77
Si         75 70  Tanaka     87 73
;
run;
```

The following statements produce a frequency table for the variable ScoreChange:

```
title 'Analysis of Score Changes';
ods select Frequencies;
proc univariate data=Score freq;
   var ScoreChange;
run;
```

The ODS SELECT statement restricts the output to the "Frequencies" table; see the section "ODS Table Names" on page 309. The FREQ option on the PROC UNIVARIATE statement requests the table of frequencies shown in Output 3.4.1.

Output 3.4.1. Table of Frequencies

```
                          Analysis of Score Changes

                           The UNIVARIATE Procedure
               Variable:  ScoreChange  (Change in Test Scores)

                               Frequency Counts

             Percents                    Percents                    Percents
Value Count  Cell   Cum     Value Count  Cell   Cum     Value Count  Cell   Cum

  -37     1   8.3   8.3       -3     2  16.7  58.3         6     1   8.3  83.3
  -14     1   8.3  16.7        2     1   8.3  66.7        12     1   8.3  91.7
   -7     1   8.3  25.0        3     1   8.3  75.0        14     1   8.3 100.0
   -5     2  16.7  41.7
```

From Output 3.4.1, the instructor sees that only score changes of -3 and -5 occurred more than once.

A sample program, uniex03.sas, for this example is available in the SAS Sample Library for Base SAS software.

Example 3.5. Creating Plots for Line Printer Output

The PLOT option in the PROC UNIVARIATE statement requests several basic plots for display in line printer output. For more information on plots created by the PLOT option, see the section "Creating Line Printer Plots" on page 281. This example illustrates the use of the PLOT option as well as BY processing in PROC UNIVARIATE.

A researcher is analyzing a data set consisting of air pollution data from three different measurement sites. The data set AirPoll, created by the following statements, contains the variables Site and Ozone, which are the site number and ozone level, respectively.

```
data AirPoll (keep = Site Ozone);
   label Site  = 'Site Number'
         Ozone = 'Ozone level (in ppb)';
   do i = 1 to 3;
      input Site @@;
      do j = 1 to 15;
         input Ozone @@;
         output;
      end;
   end;
   datalines;
102 4 6 3 4 7 8 2 3 4 1 3 8 9 5 6
134 5 3 6 2 1 2 4 3 2 4 6 4 6 3 1
137 8 9 7 8 6 7 6 7 9 8 9 8 7 8 5
;
run;
```

The following statements produce stem-and-leaf plots, box plots, and normal probability plots for each site in the AirPoll data set:

```
ods select Plots SSPlots;
proc univariate data=AirPoll plot;
   by Site;
   var Ozone;
run;
```

The PLOT option produces a stem-and-leaf plot, a box plot, and a normal probability plot for the Ozone variable at each site. Since the BY statement is used, a side-by-side box plot is also created to compare the ozone levels across sites. Note that AirPoll is sorted by Site; in general, the data set should be sorted by the BY variable using the SORT procedure. The ODS SELECT statement restricts the output to the "Plots" and "SSPlots" tables; see the section "ODS Table Names" on page 309. Optionally, you can specify the PLOTSIZE=n option to control the approximate number of rows (between 8 and the page size) that the plots occupy.

Output 3.5.1 through Output 3.5.3 show the plots produced for each BY group. Output 3.5.4 shows the side-by-side box plot for comparing Ozone values across sites.

Output 3.5.1. Ozone Plots for BY Group Site = 102

```
------------------------------ Site Number=102 ------------------------------

                       The UNIVARIATE Procedure
               Variable:  Ozone  (Ozone level (in ppb))

                Stem Leaf                     #             Boxplot
                   9 0                        1                |
                   8 00                       2                |
                   7 0                        1             +-----+
                   6 00                       2             |     |
                   5 0                        1             |     |
                   4 000                      3             *--+--*
                   3 000                      3             +-----+
                   2 0                        1                |
                   1 0                        1                |
                     ----+----+----+----+


                            Normal Probability Plot
               9.5+                                             *++++
                  |                                     *  * ++++
                  |                                   * +++++
                  |                               * *+++
               5.5+                             +*++
                  |                         **+*
                  |                   * *+*+
                  |                *++++
               1.5+           *++++
                   +----+----+----+----+----+----+----+----+----+----+
                       -2         -1         0         +1         +2
```

Output 3.5.2. Ozone Plots for BY Group Site = 134

```
------------------------------ Site Number=134 ------------------------------

                       The UNIVARIATE Procedure
               Variable:  Ozone  (Ozone level (in ppb))

                Stem Leaf                     #             Boxplot
                   6 000                      3                |
                   5 0                        1             +-----+
                   4 000                      3             |     |
                   3 000                      3             *--+--*
                   2 000                      3             +-----+
                   1 00                       2                |
                     ----+----+----+----+


                            Normal Probability Plot
               6.5+                                     *  *  ++*+++
                  |                                   * ++++++
                  |                              **+*+++
                  |                         **+*+++
                  |                   *+*+*++
               1.5+           *  ++*+++
                   +----+----+----+----+----+----+----+----+----+----+
                       -2         -1         0         +1         +2
```

Output 3.5.3. Ozone Plots for BY Group Site = 137

```
------------------------------- Site Number=137 -------------------------------

                        The UNIVARIATE Procedure
               Variable:  Ozone  (Ozone level (in ppb))

                Stem Leaf                     #             Boxplot
                   9 000                      3                |
                   8 00000                    5             +-----+
                   7 0000                     4             +--+--+
                   6 00                       2                |
                   5 0                        1                0
                     ----+----+----+----+


                              Normal Probability Plot
               9.5+                               *   *++++*++++
                  |                          * ** *+*+++++
               7.5+                   * * **++++++
                  |              *++*+++++
               5.5+     +++*++++
                   +----+----+----+----+----+----+----+----+----+----+
                       -2        -1         0        +1        +2
```

Output 3.5.4. Ozone Side-by-Side Boxplot for All BY Groups

```
                        The UNIVARIATE Procedure
               Variable:  Ozone  (Ozone level (in ppb))

                           Schematic Plots

                   |
                10 +
                   |
                   |            |                       |
                   |            |                       |
                 8 +            |                    *-----*
                   |            |                    |  +  |
                   |         +-----+                 +-----+
                   |         |     |                    |
                 6 +         |     |          |         |
                   |         |     |          |
                   |         |  +  |       +-----+      0
                   |         |     |       |     |
                 4 +         *-----*       |     |
                   |         |     |       |  +  |
                   |         +-----+       *-----*
                   |            |          |     |
                 2 +            |          +-----+
                   |            |             |
                   |            |             |
                   |
                 0 +
                     ------------+-----------+-----------+-----------
               Site             102         134         137
```

Note that you can use the PROBPLOT statement with the NORMAL option to produce high-resolution normal probability plots; see the section "Modeling a Data Distribution" on page 200.

Note that you can use the BOXPLOT procedure to produce box plots using high-resolution graphics; refer to Chapter 18, "The BOXPLOT Procedure," in *SAS/STAT User's Guide*.

A sample program, uniex04.sas, for this example is available in the SAS Sample Library for Base SAS software.

Example 3.6. Analyzing a Data Set With a FREQ Variable

This example illustrates how to use PROC UNIVARIATE to analyze a data set with a variable that contains the frequency of each observation. The data set Speeding contains data on the number of cars pulled over for speeding on a stretch of highway with a 65 mile per hour speed limit. Speed is the speed at which the cars were traveling, and Number is the number of cars at each speed. The following statements create the data set:

```
data Speeding;
   label Speed = 'Speed (in miles per hour)';
   do Speed = 66 to 85;
      input Number @@;
      output;
   end;
   datalines;
 2  3  2  1  3  6  8  9 10 13
12 14  6  2  0  0  1  1  0  1
;
run;
```

The following statements create a table of moments for the variable Speed:

```
title 'Analysis of Speeding Data';
ods select Moments;
proc univariate data=Speeding;
   freq Number;
   var Speed;
run;
```

The ODS SELECT statement restricts the output, which is shown in Output 3.6.1, to the "Moments" table; see the section "ODS Table Names" on page 309. The FREQ statement specifies that the value of the variable Number represents the frequency of each observation.

For the formulas used to compute these moments, see the section "Descriptive Statistics" on page 269. A sample program, uniex05.sas, for this example is available in the SAS Sample Library for Base SAS software.

Output 3.6.1. Table of Moments

```
                          Analysis of Speeding Data

                           The UNIVARIATE Procedure
                Variable:  Speed  (Speed (in miles per hour))

                                Freq:  Number

                                   Moments

        N                          94    Sum Weights                 94
        Mean               74.3404255    Sum Observations          6988
        Std Deviation      3.44403237    Variance             11.861359
        Skewness           -0.1275543    Kurtosis            0.92002287
        Uncorrected SS         520594    Corrected SS        1103.10638
        Coeff Variation    4.63278538    Std Error Mean      0.35522482
```

Example 3.7. Saving Summary Statistics in an OUT= Output Data Set

This example illustrates how to save summary statistics in an output data set. The following statements create a data set named Belts, which contains the breaking strengths (Strength) and widths (Width) of a sample of 50 automotive seat belts:

```
data Belts;
   label Strength = 'Breaking Strength (lb/in)'
         Width    = 'Width in Inches';
   input Strength Width @@;
   datalines;
1243.51  3.036  1221.95  2.995  1131.67  2.983  1129.70  3.019
1198.08  3.106  1273.31  2.947  1250.24  3.018  1225.47  2.980
1126.78  2.965  1174.62  3.033  1250.79  2.941  1216.75  3.037
1285.30  2.893  1214.14  3.035  1270.24  2.957  1249.55  2.958
1166.02  3.067  1278.85  3.037  1280.74  2.984  1201.96  3.002
1101.73  2.961  1165.79  3.075  1186.19  3.058  1124.46  2.929
1213.62  2.984  1213.93  3.029  1289.59  2.956  1208.27  3.029
1247.48  3.027  1284.34  3.073  1209.09  3.004  1146.78  3.061
1224.03  2.915  1200.43  2.974  1183.42  3.033  1195.66  2.995
1258.31  2.958  1136.05  3.022  1177.44  3.090  1246.13  3.022
1183.67  3.045  1206.50  3.024  1195.69  3.005  1223.49  2.971
1147.47  2.944  1171.76  3.005  1207.28  3.065  1131.33  2.984
1215.92  3.003  1202.17  3.058
;
run;
```

The following statements produce two output data sets containing summary statistics:

```
proc univariate data=Belts noprint;
   var Strength Width;
   output out=Means         mean=StrengthMean WidthMean;
   output out=StrengthStats mean=StrengthMean std=StrengthSD
                            min=StrengthMin   max=StrengthMax;
run;
```

When you specify an OUTPUT statement, you must also specify a VAR statement. You can use multiple OUTPUT statements with a single procedure statement. Each OUTPUT statement creates a new data set with the name specified by the OUT= option. In this example, two data sets, Means and StrengthStats, are created. See Output 3.7.1 for a listing of Means and Output 3.7.2 for a listing of StrengthStats.

Output 3.7.1. Listing of Output Data Set Means

Obs	Strength Mean	Width Mean
1	1205.75	3.00584

Output 3.7.2. Listing of Output Data Set StrengthStats

Obs	Strength Mean	Strength SD	Strength Max	Strength Min
1	1205.75	48.3290	1289.59	1101.73

Summary statistics are saved in an output data set by specifying *keyword=names* after the OUT= option. In the preceding statements, the first OUTPUT statement specifies the *keyword* MEAN followed by the *names* StrengthMean and WidthMean. The second OUTPUT statement specifies the *keywords* MEAN, STD, MAX, and MIN, for which the *names* StrengthMean, StrengthSD, StrengthMax, and StrengthMin are given.

The *keyword* specifies the statistic to be saved in the output data set, and the *names* determine the names for the new variables. The first *name* listed after a keyword contains that statistic for the first variable listed in the VAR statement; the second *name* contains that statistic for the second variable in the VAR statement, and so on.

The data set Means contains the mean of Strength in a variable named StrengthMean and the mean of Width in a variable named WidthMean. The data set StrengthStats contains the mean, standard deviation, maximum value, and minimum value of Strength in the variables StrengthMean, StrengthSD, StrengthMax, and StrengthMin, respectively.

See the section "OUT= Output Data Set in the OUTPUT Statement" on page 306 for more information on OUT= output data sets.

A sample program, uniex06.sas, for this example is available in the SAS Sample Library for Base SAS software.

Example 3.8. Saving Percentiles in an Output Data Set

This example, which uses the Belts data set from the previous example, illustrates how to save percentiles in an output data set. The UNIVARIATE procedure automatically computes the 1st, 5th, 10th, 25th, 75th, 90th, 95th, and 99th percentiles for each variable. You can save these percentiles in an output data set by specifying the appropriate keywords. For example, the following statements create an output data set named PctlStrength, which contains the 5th and 95th percentiles of the variable Strength:

```
proc univariate data=Belts noprint;
   var Strength Width;
   output out=PctlStrength p5=p5str p95=p95str;
run;
```

The output data set PctlStrength is listed in Output 3.8.1.

Output 3.8.1. Listing of Output Data Set PctlStrength

```
Obs     p95str     p5str

 1     1284.34    1126.78
```

You can use the PCTLPTS=, PCTLPRE=, and PCTLNAME= options to save percentiles not automatically computed by the UNIVARIATE procedure. For example, the following statements create an output data set named Pctls, which contains the 20th and 40th percentiles of the variables Strength and Width:

```
proc univariate data=Belts noprint;
   var Strength Width;
   output out=Pctls pctlpts  = 20 40
                    pctlpre  = Strength Width
                    pctlname = pct20 pct40;
run;
```

The PCTLPTS= option specifies the percentiles to compute (in this case, the 20th and 40th percentiles). The PCTLPRE= and PCTLNAME= options build the names for the variables containing the percentiles. The PCTLPRE= option gives prefixes for the new variables, and the PCTLNAME= option gives a suffix to add to the prefix. When you use the PCTLPTS= specification, you must also use the PCTLPRE= specification.

The OUTPUT statement saves the 20th and 40th percentiles of Strength and Width in the variables Strengthpct20, Widthpct20, Strengthpct40, and Weightpct40. The output data set Pctls is listed in Output 3.8.2.

Output 3.8.2. Listing of Output Data Set Pctls

Obs	Strengthpct20	Strengthpct40	Widthpct20	Widthpct40
1	1165.91	1199.26	2.9595	2.995

A sample program, uniex06.sas, for this example is available in the SAS Sample Library for Base SAS software.

Example 3.9. Computing Confidence Limits for the Mean, Standard Deviation, and Variance

This example illustrates how to compute confidence limits for the mean, standard deviation, and variance of a population. A researcher is studying the heights of a certain population of adult females. She has collected a random sample of heights of 75 females, which are saved in the data set Heights:

```
data Heights;
   label Height = 'Height (in)';
   input Height @@;
   datalines;
64.1 60.9 64.1 64.7 66.7 65.0 63.7 67.4 64.9 63.7
64.0 67.5 62.8 63.9 65.9 62.3 64.1 60.6 68.6 68.6
63.7 63.0 64.7 68.2 66.7 62.8 64.0 64.1 62.1 62.9
62.7 60.9 61.6 64.6 65.7 66.6 66.7 66.0 68.5 64.4
60.5 63.0 60.0 61.6 64.3 60.2 63.5 64.7 66.0 65.1
63.6 62.0 63.6 65.8 66.0 65.4 63.5 66.3 66.2 67.5
65.8 63.1 65.8 64.4 64.0 64.9 65.7 61.0 64.1 65.5
68.6 66.6 65.7 65.1 70.0
;
run;
```

The following statements produce confidence limits for the mean, standard deviation, and variance of the population of heights:

```
title 'Analysis of Female Heights';
ods select BasicIntervals;
proc univariate data=Heights cibasic;
   var Height;
run;
```

The CIBASIC option requests confidence limits for the mean, standard deviation, and variance. For example, Output 3.9.1 shows that the 95% confidence interval for the population mean is $(64.06, 65.07)$. The ODS SELECT statement restricts the output to the "BasicIntervals" table; see the section "ODS Table Names" on page 309.

The confidence limits in Output 3.9.1 assume that the heights are normally distributed, so you should check this assumption before using these confidence limits.

See the section "Shapiro-Wilk Statistic" on page 293 for information on the Shapiro-Wilk test for normality in PROC UNIVARIATE. See Example 3.19 for an example using the test for normality.

Output 3.9.1. Default 95% Confidence Limits

```
                    Analysis of Female Heights

                      The UNIVARIATE Procedure
                  Variable:  Height  (Height (in))

              Basic Confidence Limits Assuming Normality

     Parameter          Estimate     95% Confidence Limits

     Mean               64.56667      64.06302    65.07031
     Std Deviation       2.18900       1.88608     2.60874
     Variance            4.79171       3.55731     6.80552
```

By default, the confidence limits produced by the CIBASIC option produce 95% confidence intervals. You can request different level confidence limits by using the ALPHA= option in parentheses after the CIBASIC option. The following statements produce 90% confidence limits:

```
title 'Analysis of Female Heights';
ods select BasicIntervals;
proc univariate data=Heights cibasic(alpha=.1);
   var Height;
run;
```

The 90% confidence limits are displayed in Output 3.9.2.

Output 3.9.2. 90% Confidence Limits

```
                    Analysis of Female Heights

                      The UNIVARIATE Procedure
                  Variable:  Height  (Height (in))

              Basic Confidence Limits Assuming Normality

     Parameter          Estimate     90% Confidence Limits

     Mean               64.56667      64.14564    64.98770
     Std Deviation       2.18900       1.93114     2.53474
     Variance            4.79171       3.72929     6.42492
```

For the formulas used to compute these limits, see the section "Confidence Limits for Parameters of the Normal Distribution" on page 277.

A sample program, uniex07.sas, for this example is available in the SAS Sample Library for Base SAS software.

Example 3.10. Computing Confidence Limits for Quantiles and Percentiles

This example, which is a continuation of Example 3.9, illustrates how to compute confidence limits for quantiles and percentiles. A second researcher is more interested in summarizing the heights with quantiles than the mean and standard deviation. He is also interested in computing 90% confidence intervals for the quantiles. The following statements produce estimated quantiles and confidence limits for the population quantiles:

```
title 'Analysis of Female Heights';
ods select Quantiles;
proc univariate data=Heights ciquantnormal(alpha=.1);
   var Height;
run;
```

The ODS SELECT statement restricts the output to the "Quantiles" table; see the section "ODS Table Names" on page 309. The CIQUANTNORMAL option produces confidence limits for the quantiles. As noted in Output 3.10.1, these limits assume that the data are normally distributed. You should check this assumption before using these confidence limits. See the section "Shapiro-Wilk Statistic" on page 293 for information on the Shapiro-Wilk test for normality in PROC UNIVARIATE; see Example 3.19 for an example using the test for normality.

Output 3.10.1. Normal-Based Quantile Confidence Limits

```
                    Analysis of Female Heights

                     The UNIVARIATE Procedure
                  Variable:  Height  (Height (in))

                    Quantiles (Definition 5)

                                      90% Confidence Limits
          Quantile       Estimate       Assuming Normality

          100% Max           70.0
          99%                70.0     68.94553     70.58228
          95%                68.6     67.59184     68.89311
          90%                67.5     66.85981     68.00273
          75% Q3             66.0     65.60757     66.54262
          50% Median         64.4     64.14564     64.98770
          25% Q1             63.1     62.59071     63.52576
          10%                61.6     61.13060     62.27352
          5%                 60.6     60.24022     61.54149
          1%                 60.0     58.55106     60.18781
          0% Min             60.0
```

It is also possible to use PROC UNIVARIATE to compute confidence limits for quantiles without assuming normality. The following statements use the CIQUANTDF option to request distribution-free confidence limits for the quantiles of the population of heights:

```
title 'Analysis of Female Heights';
ods select Quantiles;
proc univariate data=Heights ciquantdf(alpha=.1);
   var Height;
run;
```

The distribution-free confidence limits are shown in Output 3.10.2.

Output 3.10.2. Distribution-Free Quantile Confidence Limits

Analysis of Female Heights

The UNIVARIATE Procedure
Variable: Height (Height (in))

Quantiles (Definition 5)

Quantile	Estimate	90% Confidence Limits Distribution Free		-------Order Statistics------- LCL Rank	UCL Rank	Coverage
100% Max	70.0					
99%	70.0	68.6	70.0	73	75	48.97
95%	68.6	67.5	70.0	68	75	94.50
90%	67.5	66.6	68.6	63	72	91.53
75% Q3	66.0	65.7	66.6	50	63	91.77
50% Median	64.4	64.1	65.1	31	46	91.54
25% Q1	63.1	62.7	63.7	13	26	91.77
10%	61.6	60.6	62.7	4	13	91.53
5%	60.6	60.0	61.6	1	8	94.50
1%	60.0	60.0	60.5	1	3	48.97
0% Min	60.0					

The table in Output 3.10.2 includes the ranks from which the confidence limits are computed. For more information on how these confidence limits are calculated, see the section "Confidence Limits for Percentiles" on page 274. Note that confidence limits for quantiles are not produced when the WEIGHT statement is used.

A sample program, uniex07.sas, for this example is available in the SAS Sample Library for Base SAS software.

Example 3.11. Computing Robust Estimates

This example illustrates how you can use the UNIVARIATE procedure to compute robust estimates of location and scale. The following statements compute these estimates for the variable Systolic in the data set BPressure, which was introduced in Example 3.1:

```
title 'Robust Estimates for Blood Pressure Data';
ods select TrimmedMeans WinsorizedMeans RobustScale;
proc univariate data=BPressure trimmed=1 .1
                winsorized=.1  robustscale;
   var Systolic;
run;
```

The ODS SELECT statement restricts the output to the "TrimmedMeans," "WinsorizedMeans," and "RobustScale" tables; see the section "ODS Table Names" on page 309. The TRIMMED= option computes two trimmed means, the first after removing one observation and the second after removing 10% of the observations. If the value of TRIMMED= is greater than or equal to one, it is interpreted as the number of observations to be trimmed. The WINSORIZED= option computes a Winsorized mean that replaces three observations from the tails with the next closest observations. (Three observations are replaced because $np = (22)(.1) = 2.2$, and three is the smallest integer greater than 2.2.) The trimmed and Winsorized means for Systolic are displayed in Output 3.11.1.

Output 3.11.1. Computation of Trimmed and Winsorized Means

```
                          Robust Estimates for Blood Pressure Data

                                 The UNIVARIATE Procedure
                                   Variable:  Systolic

                                      Trimmed Means

 Percent    Number                Std Error
 Trimmed   Trimmed     Trimmed     Trimmed     95% Confidence               t for H0:
 in Tail   in Tail        Mean        Mean         Limits           DF     Mu0=0.00    Pr > |t|

    4.55         1    120.3500    2.573536   114.9635  125.7365      19    46.76446      <.0001
   13.64         3    120.3125    2.395387   115.2069  125.4181      15    50.22675      <.0001


                                     Winsorized Means

   Percent       Number                  Std Error
Winsorized   Winsorized  Winsorized  Winsorized     95% Confidence              t for H0:
   in Tail      in Tail        Mean        Mean         Limits           DF    Mu0=0.00  Pr > |t|

     13.64            3    120.6364    2.417065   115.4845  125.7882      15    49.91027    <.0001
```

Output 3.11.1 shows the trimmed mean for Systolic is 120.35 after one observation has been trimmed, and 120.31 after 3 observations are trimmed. The Winsorized mean for Systolic is 120.64. For details on trimmed and Winsorized means, see the section "Robust Estimators" on page 278. The trimmed means can be compared with the means shown in Output 3.1.1 (from Example 3.1), which displays the mean for Systolic as 121.273.

The ROBUSTSCALE option requests a table, displayed in Output 3.11.2, which includes the interquartile range, Gini's mean difference, the median absolute deviation about the median, Q_n, and S_n.

Output 3.11.2 shows the robust estimates of scale for Systolic. For instance, the interquartile range is 13. The estimates of σ range from 9.54 to 13.32. See the section "Robust Estimators" on page 278.

A sample program, uniex01.sas, for this example is available in the SAS Sample Library for Base SAS software.

Output 3.11.2. Computation of Robust Estimates of Scale

```
                 Robust Estimates for Blood Pressure Data

                           Variable:  Systolic

                         Robust Measures of Scale

                                                    Estimate
              Measure                      Value    of Sigma

              Interquartile Range       13.00000     9.63691
              Gini's Mean Difference    15.03030    13.32026
              MAD                        6.50000     9.63690
              Sn                         9.54080     9.54080
              Qn                        13.33140    11.36786
```

Example 3.12. Testing for Location

This example, which is a continuation of Example 3.9, illustrates how to carry out three tests for location: the Student's t test, the sign test, and the Wilcoxon signed rank test. These tests are discussed in the section "Tests for Location" on page 275.

The following statements demonstrate the tests for location using the Heights data set introduced in Example 3.9. Since the data consists of adult female heights, the researchers are not interested in testing if the mean of the population is equal to zero inches, which is the default μ_0 value. Instead, they are interested in testing if the mean is equal to 66 inches. The following statements test the null hypothesis H_0: $\mu_0 = 66$:

```
title 'Analysis of Female Height Data';
ods select TestsForLocation LocationCounts;
proc univariate data=Heights mu0=66 loccount;
   var Height;
run;
```

The ODS SELECT statement restricts the output to the "TestsForLocation" and "LocationCounts" tables; see the section "ODS Table Names" on page 309. The MU0= option specifies the null hypothesis value of μ_0 for the tests for location; by default, $\mu_0 = 0$. The LOCCOUNT option produces the table of the number of observations greater than, not equal to, and less than 66 inches.

Output 3.12.1 contains the results of the tests for location. All three tests are highly significant, causing the researchers to reject the hypothesis that the mean is 66 inches.

A sample program, uniex07.sas, for this example is available in the SAS Sample Library for Base SAS software.

Output 3.12.1. Tests for Location with MU0=66 and LOCCOUNT

```
                    Analysis of Female Height Data

                       The UNIVARIATE Procedure
                  Variable:  Height  (Height (in))

                      Tests for Location: Mu0=66

          Test           -Statistic-    -----p Value------

          Student's t    t  -5.67065    Pr > |t|    <.0001
          Sign           M       -20    Pr >= |M|   <.0001
          Signed Rank    S      -849    Pr >= |S|   <.0001


                      Location Counts: Mu0=66.00

                      Count                  Value

                      Num Obs > Mu0             16
                      Num Obs ^= Mu0            72
                      Num Obs < Mu0             56
```

Example 3.13. Performing a Sign Test Using Paired Data

This example demonstrates a sign test for paired data, which is a specific application of the Tests for Location discussed in Example 3.12.

The instructor from Example 3.4 is now interested in performing a sign test for the pairs of test scores in his college course. The following statements request basic statistical measures and tests for location:

```
title 'Test Scores for a College Course';
ods select BasicMeasures TestsForLocation;
proc univariate data=Score;
   var ScoreChange;
run;
```

The ODS SELECT statement restricts the output to the "BasicMeasures" and "TestsForLocation" tables; see the section "ODS Table Names" on page 309. The instructor is not willing to assume the ScoreChange variable is normal or even symmetric, so he decides to examine the sign test. The large p-value (0.7744) of the sign test provides insufficient evidence of a difference in test score medians.

Output 3.13.1. Sign Test for ScoreChange

```
                        Test Scores for a College Course

                            The UNIVARIATE Procedure
                 Variable:  ScoreChange  (Change in Test Scores)

                           Basic Statistical Measures

                  Location                    Variability

              Mean      -3.08333     Std Deviation            13.33797
              Median    -3.00000     Variance                177.90152
              Mode      -5.00000     Range                    51.00000
                                     Interquartile Range      10.50000

      NOTE: The mode displayed is the smallest of 2 modes with a count of 2.


                           Tests for Location: Mu0=0

                 Test           -Statistic-    -----p Value------

                 Student's t    t  -0.80079    Pr > |t|    0.4402
                 Sign           M        -1    Pr >= |M|   0.7744
                 Signed Rank    S      -8.5    Pr >= |S|   0.5278
```

A sample program, uniex03.sas, for this example is available in the SAS Sample Library for Base SAS software.

Example 3.14. Creating a Histogram

This example illustrates how to create a histogram. A semiconductor manufacturer produces printed circuit boards that are sampled to determine the thickness of their copper plating. The following statements create a data set named Trans, which contains the plating thicknesses (Thick) of 100 boards:

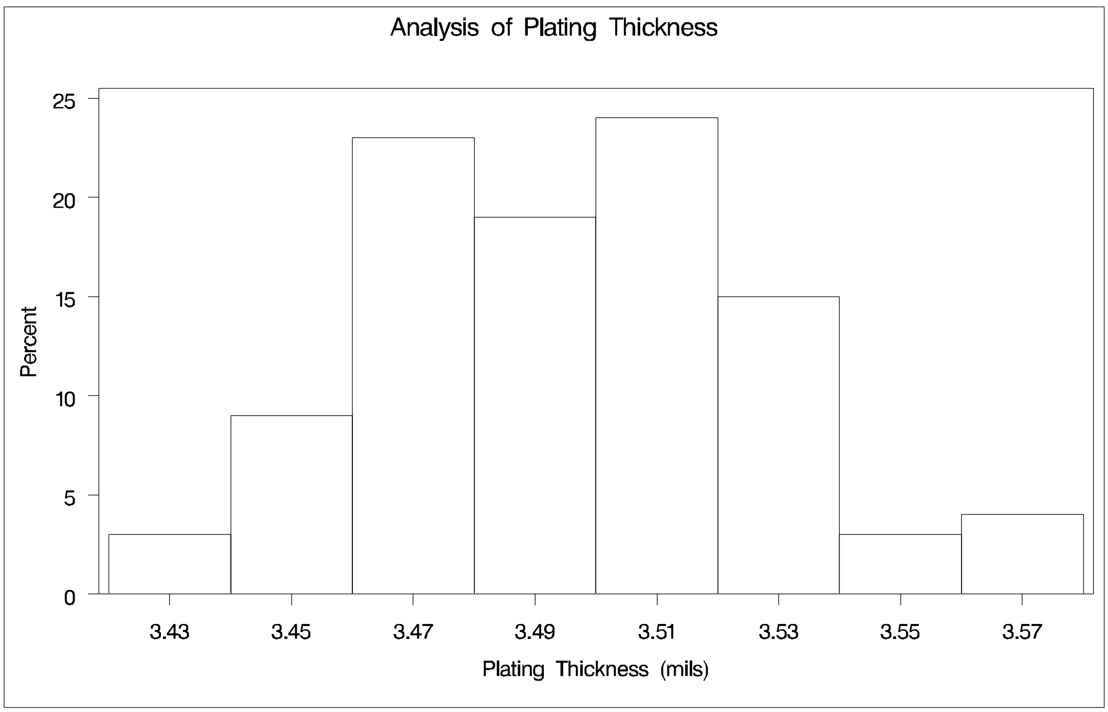
```
data Trans;
   input Thick @@;
   label Thick = 'Plating Thickness (mils)';
   datalines;
3.468 3.428 3.509 3.516 3.461 3.492 3.478 3.556 3.482 3.512
3.490 3.467 3.498 3.519 3.504 3.469 3.497 3.495 3.518 3.523
3.458 3.478 3.443 3.500 3.449 3.525 3.461 3.489 3.514 3.470
3.561 3.506 3.444 3.479 3.524 3.531 3.501 3.495 3.443 3.458
3.481 3.497 3.461 3.513 3.528 3.496 3.533 3.450 3.516 3.476
3.512 3.550 3.441 3.541 3.569 3.531 3.468 3.564 3.522 3.520
3.505 3.523 3.475 3.470 3.457 3.536 3.528 3.477 3.536 3.491
3.510 3.461 3.431 3.502 3.491 3.506 3.439 3.513 3.496 3.539
3.469 3.481 3.515 3.535 3.460 3.575 3.488 3.515 3.484 3.482
3.517 3.483 3.467 3.467 3.502 3.471 3.516 3.474 3.500 3.466
;
run;
```

The following statements create the histogram shown in Output 3.14.1.

```
title 'Analysis of Plating Thickness';
proc univariate data=Trans noprint;
   histogram Thick;
run;
```

The NOPRINT option in the PROC UNIVARIATE statement suppresses tables of summary statistics for the variable Thick that would be displayed by default. A histogram is created for each variable listed in the HISTOGRAM statement.

Output 3.14.1. Histogram for Plating Thickness



A sample program, uniex08.sas, for this example is available in the SAS Sample Library for Base SAS software.

Example 3.15. Creating a One-Way Comparative Histogram

This example illustrates how to create a comparative histogram. The effective channel length (in microns) is measured for 1225 field effect transistors. The channel lengths (Length) are stored in a data set named Channel, which is partially listed in Output 3.15.1:

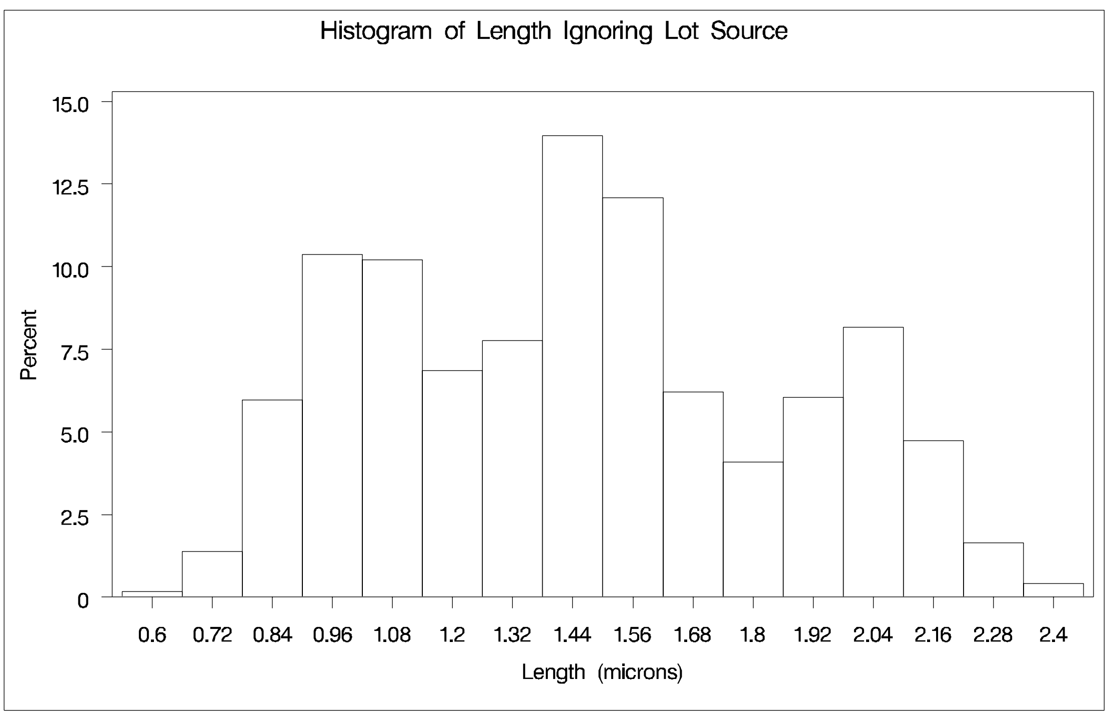
Output 3.15.1. Partial Listing of Data Set Channel

```
                    The Data Set Channel

                      Lot      Length

                     Lot 1      0.91
                       .         .
                     Lot 1      1.17
                     Lot 2      1.47
                       .         .
                     Lot 2      1.39
                     Lot 3      2.04
                       .         .
                     Lot 3      1.91
```

The following statements request a histogram, which is shown in Output 3.15.2 of Length ignoring the lot source:

```
title 'Histogram of Length Ignoring Lot Source';
proc univariate data=Channel noprint;
   histogram Length;
run;
```

Output 3.15.2. Histogram for Length Ignoring Lot Source

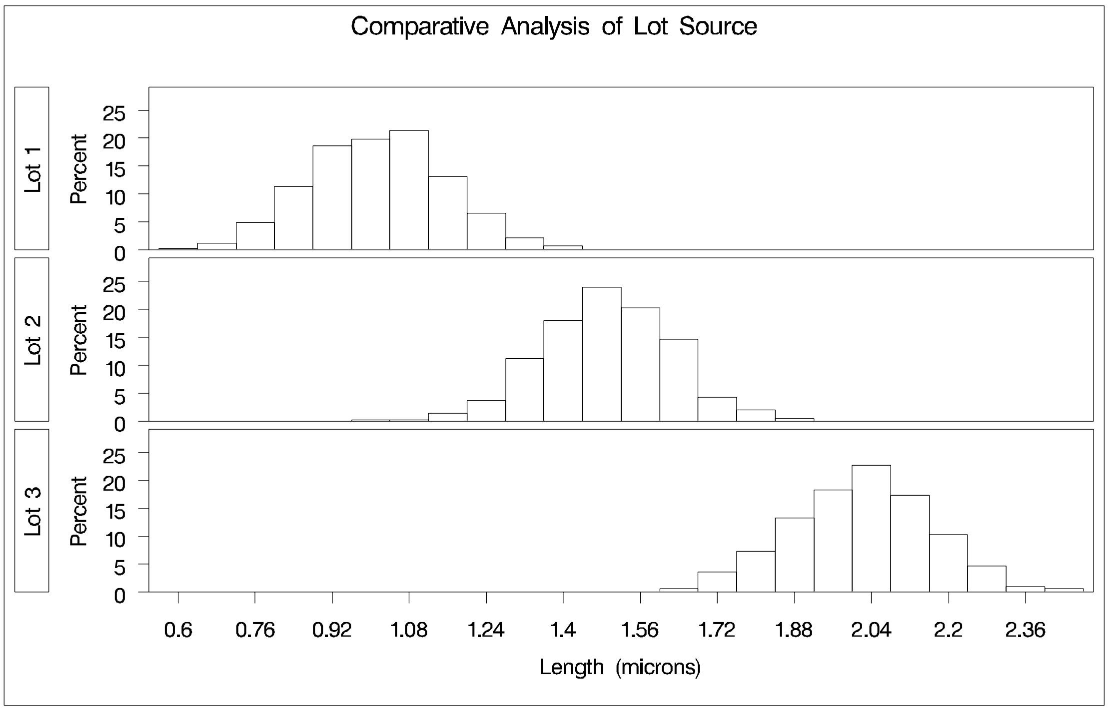


To investigate whether the peaks (modes) in Output 3.15.2 are related to the lot source, you can create a comparative histogram using Lot as a classification variable. The following statements create the histogram shown in Output 3.15.3:

```
title 'Comparative Analysis of Lot Source';
proc univariate data=Channel noprint;
   class Lot;
   histogram Length / nrows = 3;
run;
```

The CLASS statement requests comparisons for each level (distinct value) of the classification variable Lot. The HISTOGRAM statement requests a comparative histogram for the variable Length. The NROWS= option specifies the number of rows in the comparative histogram. By default, comparative histograms are displayed in two rows per panel.

Output 3.15.3. Comparison by Lot Source



Output 3.15.3 reveals that the distributions of Length are similarly distributed except for shifts in mean.

A sample program, uniex09.sas, for this example is available in the SAS Sample Library for Base SAS software.

Example 3.16. Creating a Two-Way Comparative Histogram

This example illustrates how to create a two-way comparative histogram. Two suppliers (A and B) provide disk drives for a computer manufacturer. The manufacturer measures the disk drive opening width to determine whether there has been a change in variability from 2002 to 2003 for each supplier.

The following statements save the measurements in a data set named Disk. There are two classification variables, Supplier and Year, and a user-defined format is associated with Year.

```
proc format ;
   value mytime  1 = '2002' 2 = '2003';

data Disk;
   input @1 Supplier $10. Year Width;
   label Width = 'Opening Width (inches)';
   format Year mytime.;
datalines;
Supplier A   1   1.8932
    .        .    .
Supplier B   1   1.8986
Supplier A   2   1.8978
    .        .    .
Supplier B   2   1.8997
;
```

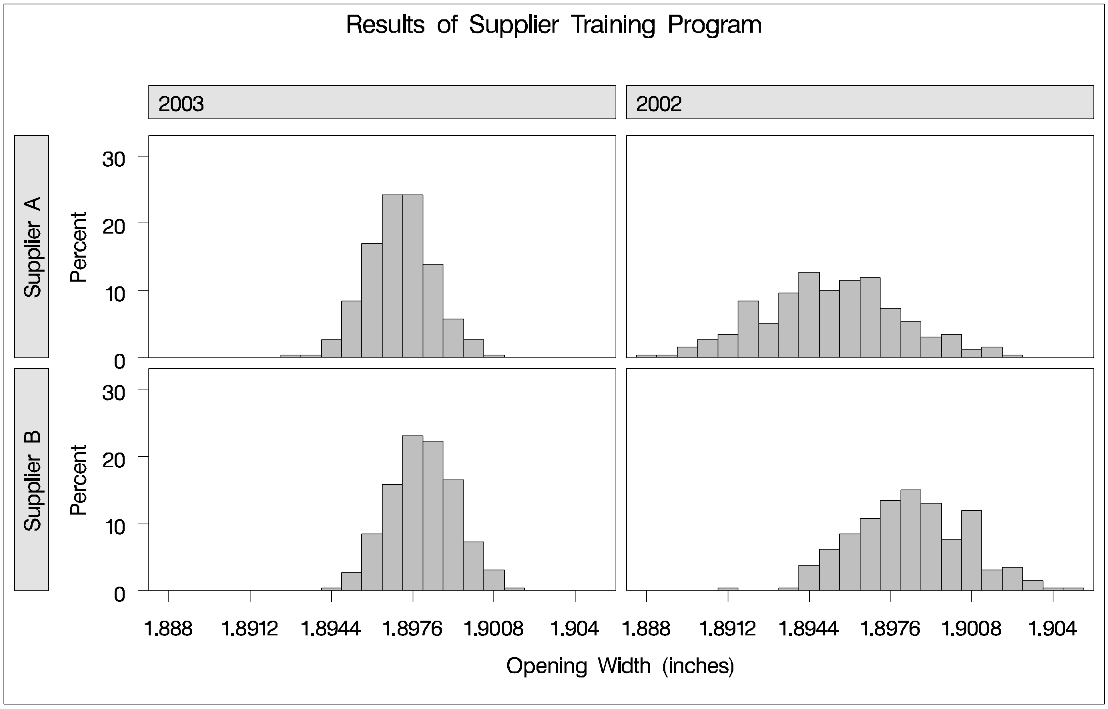
The following statements create the comparative histogram in Output 3.16.1:

```
title 'Results of Supplier Training Program';
proc univariate data=Disk noprint;
   class Supplier Year / keylevel = ('Supplier A' '2003');
   histogram Width / intertile  = 1.0
                     vaxis      = 0 10 20 30
                     ncols      = 2
                     nrows      = 2
                     cfill      = ligr
                     cframetop  = yellow
                     cframeside = yellow;
run;
```

The KEYLEVEL= option specifies the key cell as the cell for which Supplier is equal to 'SUPPLIER A' and Year is equal to '2003.' This cell determines the binning for the other cells, and the columns are arranged so that this cell is displayed in the upper left corner. Without the KEYLEVEL= option, the default key cell would be the cell for which Supplier is equal to 'SUPPLIER A' and Year is equal to '2002'; the column labeled '2002' would be displayed to the left of the column labeled '2003.'

The VAXIS= option specifies the tick mark labels for the vertical axis. The NROWS=2 and NCOLS=2 options specify a 2×2 arrangement for the tiles. The CFRAMESIDE= and CFRAMETOP= options specify fill colors for the row and column labels, and the CFILL= option specifies a fill color for the bars. Output 3.16.1 provides evidence that both suppliers have reduced variability from 2002 to 2003.

Output 3.16.1. Two-Way Comparative Histogram



A sample program, uniex10.sas, for this example is available in the SAS Sample Library for Base SAS software.

Example 3.17. Adding Insets with Descriptive Statistics

This example illustrates how to add insets with descriptive statistics to a comparative histogram; see Output 3.17.1. Three similar machines are used to attach a part to an assembly. One hundred assemblies are sampled from the output of each machine, and a part position is measured in millimeters. The following statements create the data set Machines, which contains the measurements in a variable named Position:

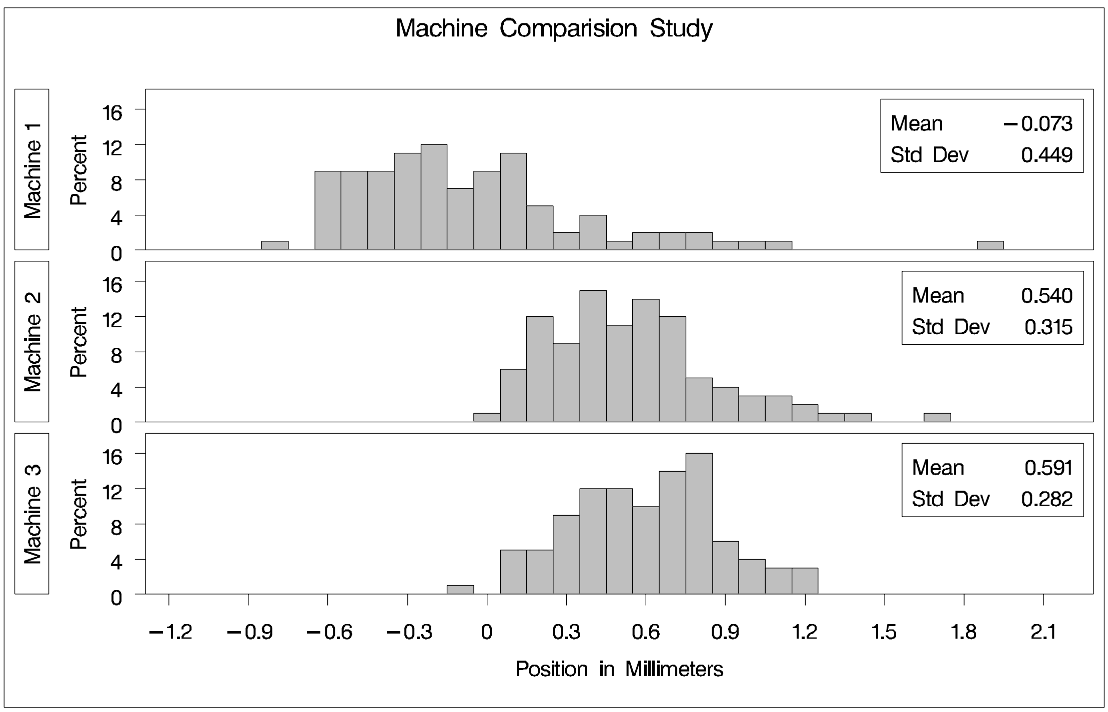
```
data Machines;
   input Position @@;
   label Position = 'Position in Millimeters';
   if      (_n_ <= 100) then Machine = 'Machine 1';
   else if (_n_ <= 200) then Machine = 'Machine 2';
   else                      Machine = 'Machine 3';
   datalines;
-0.17  -0.19  -0.24  -0.24  -0.12   0.07  -0.61   0.22   1.91  -0.08
-0.59   0.05  -0.38   0.82  -0.14   0.32   0.12  -0.02   0.26   0.19

  ...

 0.48   0.41   0.78   0.58   0.43   0.07   0.27   0.49   0.79   0.92
 0.79   0.66   0.22   0.71   0.53   0.57   0.90   0.48   1.17   1.03
;
run;
```

The following statements create the comparative histogram in Output 3.17.1:

```
title 'Machine Comparision Study';
proc univariate data=Machines noprint;
   class Machine;
   histogram Position / nrows     = 3
                        intertile = 1
                        midpoints = -1.2 to 2.2 by 0.1
                        vaxis     =   0  to 16  by  4
                        cfill     = ligr;
   inset mean std="Std Dev" / pos = ne format = 6.3;
run;
```

The INSET statement requests insets containing the sample mean and standard deviation for each machine in the corresponding tile. The MIDPOINTS= option specifies the midpoints of the histogram bins.

Output 3.17.1. Comparative Histograms



Output 3.17.1 shows that the average position for Machines 2 and 3 are similar and that the spread for Machine 1 is much larger than for Machines 2 and 3.

A sample program, uniex11.sas, for this example is available in the SAS Sample Library for Base SAS software.

Example 3.18. Binning a Histogram

This example, which is a continuation of Example 3.14, demonstrates various methods for binning a histogram. This example also illustrates how to save bin percentages in an OUTHISTOGRAM= data set.

The manufacturer from Example 3.14 now wants to enhance the histogram by changing the endpoints of the bins using the ENDPOINTS= option. The following statements create a histogram with bins that have end points 3.425 and 3.6 and width 0.025:

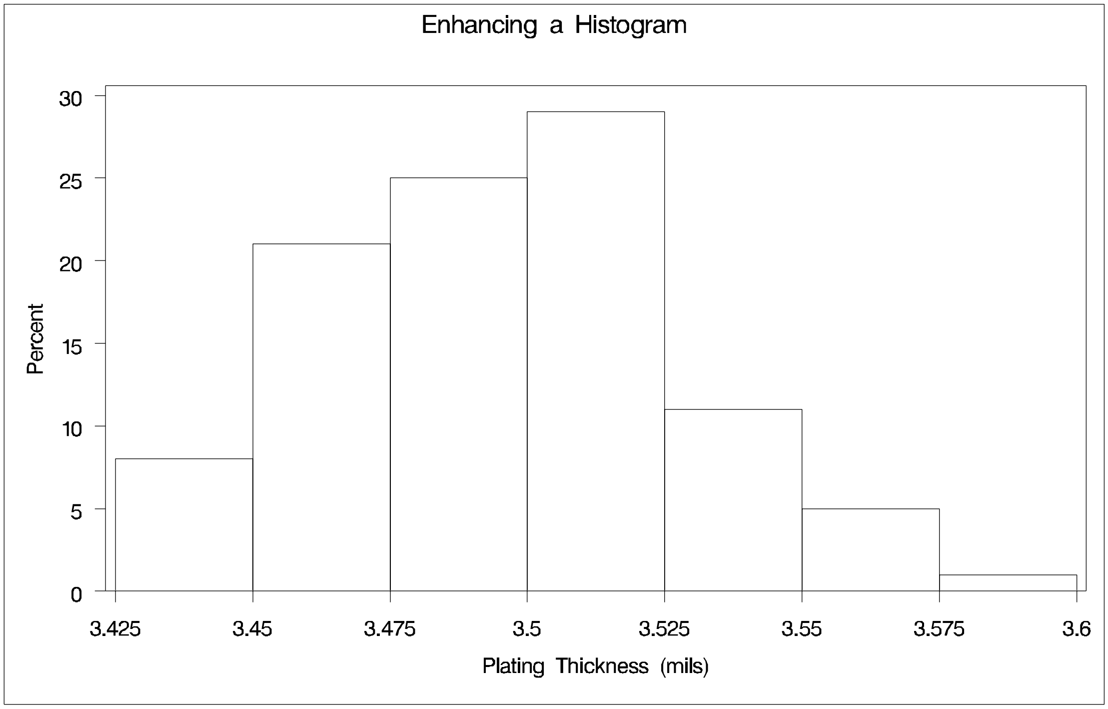
```
title 'Enhancing a Histogram';
ods select HistogramBins MyHist;
proc univariate data=Trans;
   histogram Thick / midpercents name='MyHist'
                     endpoints = 3.425 to 3.6 by .025;
run;
```

The ODS SELECT statement restricts the output to the “HistogramBins” table and the “MyHist” histogram; see the section “ODS Table Names” on page 309. The ENDPOINTS= option specifies the endpoints for the histogram bins. By default, if the ENDPOINTS= option is not specified, the automatic binning algorithm computes values for the midpoints of the bins. The MIDPERCENTS option requests a table of the midpoints of each histogram bin and the percent of the observations that fall in each bin. This table is displayed in Output 3.18.1; the histogram is displayed in Output 3.18.2. The NAME= option specifies a name for the histogram that can be used in the ODS SELECT statement.

Output 3.18.1. Table of Bin Percentages Requested with MIDPERCENTS Option

```
                    Enhancing a Histogram

                  The UNIVARIATE Procedure
                      Variable:  Thick

                  Histogram Bins for Thick

                         Bin
                      Minimum   Observed
                        Point    Percent

                        3.425      8.000
                        3.450     21.000
                        3.475     25.000
                        3.500     29.000
                        3.525     11.000
                        3.550      5.000
                        3.575      1.000
```

Output 3.18.2. Histogram with ENDPOINTS= Option



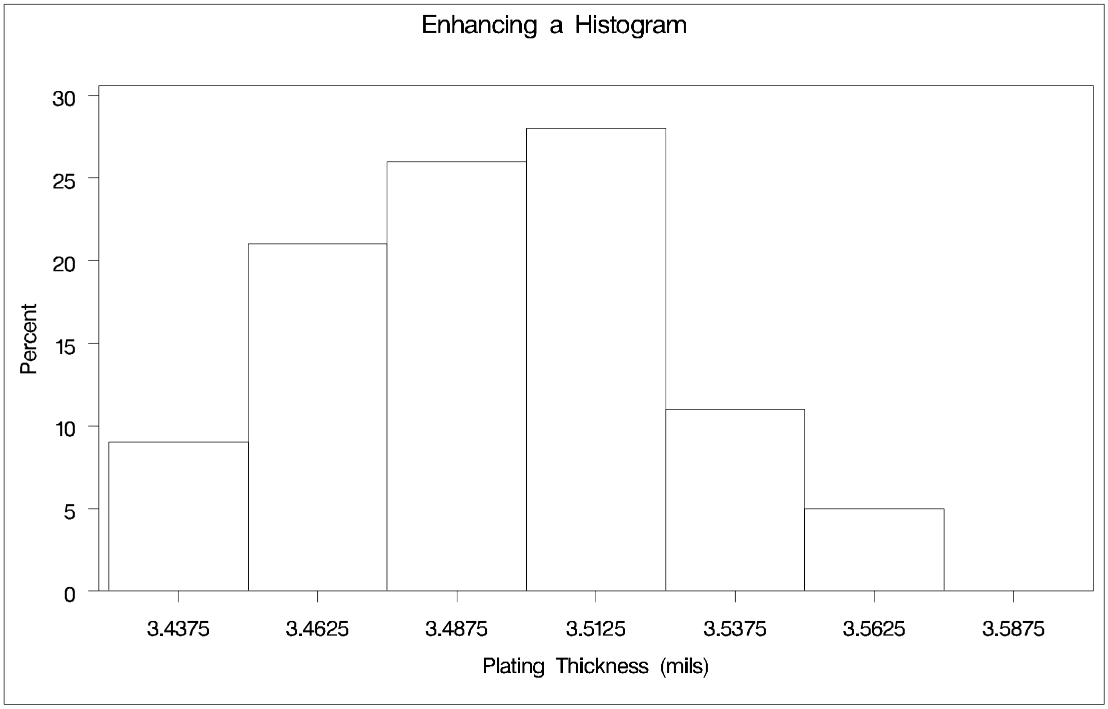
The MIDPOINTS= option is an alternative to the ENDPOINTS= option for specifying histogram bins. The following statements create a similar histogram, which is shown in Output 3.18.3, to the one in Output 3.18.2:

```
title 'Enhancing a Histogram';
proc univariate data=Trans noprint;
   histogram Thick / midpoints    = 3.4375 to 3.5875 by .025
                     rtinclude
                     outhistogram = OutMdpts;
run;
```

Output 3.18.3 differs from Output 3.18.2 in two ways:

- The MIDPOINTS= option specifies the bins for the histogram by specifying the midpoints of the bins instead of specifying the endpoints. Note that the histogram displays midpoints instead of endpoints.
- The RTINCLUDE option request that the right endpoint of each bin be included in the histogram interval instead of the default, which is to include the left endpoint in the interval. This changes the histogram slightly from Output 3.18.2. Six observations have a thickness equal to an endpoint of an interval. For instance, there is one observation with a thickness of 3.45 mils. In Output 3.18.3, this observation is included in the bin from 3.425 to 3.45.

Output 3.18.3. Histogram with MIDPOINTS= and RTINCLUDE Options



The OUTHISTOGRAM= option produces an output data set named OutMdpts, displayed in Output 3.18.4. This data set provides information on the bins of the histogram. For more information, see the section "OUTHISTOGRAM= Output Data Set" on page 308.

Output 3.18.4. The OUTHISTOGRAM= Data Set OutMdpts

OUTHISTOGRAM= Data Set

Obs	_VAR_	_MIDPT_	_OBSPCT_
1	Thick	3.4375	9
2	Thick	3.4625	21
3	Thick	3.4875	26
4	Thick	3.5125	28
5	Thick	3.5375	11
6	Thick	3.5625	5
7	Thick	3.5875	0

A sample program, uniex08.sas, for this example is available in the SAS Sample Library for Base SAS software.

Example 3.19. Adding a Normal Curve to a Histogram

This example is a continuation of Example 3.14. The following statements fit a normal distribution to the thickness measurements in the Trans data set and superimpose the fitted density curve on the histogram:

```
title 'Analysis of Plating Thickness';
ods select ParameterEstimates GoodnessOfFit FitQuantiles Bins MyPlot;
proc univariate data=Trans;
   histogram Thick / normal(percents=20 40 60 80 midpercents)
                     name='MyPlot';
   inset n normal(ksdpval) / pos = ne format = 6.3;
run;
```

The ODS SELECT statement restricts the output to the “ParameterEstimates,” “GoodnessOfFit,” “FitQuantiles,” and “Bins” tables; see the section “ODS Table Names” on page 309. The NORMAL option requests specifies that the normal curve is to be displayed on the histogram shown in Output 3.19.3. It also requests a summary of the fitted distribution, which is shown in Output 3.19.1 and Output 3.19.2. This summary includes goodness-of-fit tests, parameter estimates, and quantiles of the fitted distribution. (If you specify the NORMALTEST option in the PROC UNIVARIATE statement, the Shapiro-Wilk test for normality will be included in the tables of statistical output.)

Two secondary options are specified in parentheses after the NORMAL primary option. The PERCENTS= option specifies quantiles, which are to be displayed in the “FitQuantiles” table. The MIDPERCENTS option requests a table that lists the midpoints, the observed percentage of observations, and the estimated percentage of the population in each interval (estimated from the fitted normal distribution). See Table 3.3 on page 214 and Table 3.8 on page 215 for the secondary options that can be specified with after the NORMAL primary option.

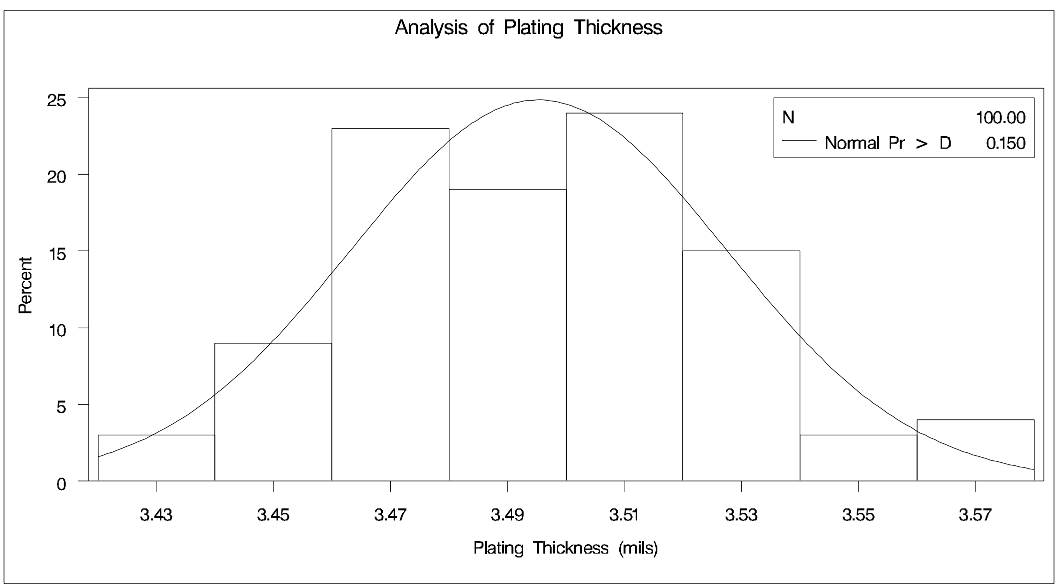
Output 3.19.1. Summary of Fitted Normal Distribution

```
                     Analysis of Plating Thickness

                        The UNIVARIATE Procedure
                     Fitted Distribution for Thick

                   Parameters for Normal Distribution

                     Parameter    Symbol    Estimate

                     Mean         Mu         3.49533
                     Std Dev      Sigma     0.032117


               Goodness-of-Fit Tests for Normal Distribution

       Test                  ---Statistic----   -----p Value-----

       Kolmogorov-Smirnov    D      0.05563823   Pr > D      >0.150
       Cramer-von Mises      W-Sq   0.04307548   Pr > W-Sq   >0.250
       Anderson-Darling      A-Sq   0.27840748   Pr > A-Sq   >0.250
```

Output 3.19.2. Summary of Fitted Normal Distribution (cont.)

```
                    Analysis of Plating Thickness

                    Fitted Distribution for Thick

            Histogram Bin Percents for Normal Distribution

                   Bin    -------Percent------
              Midpoint    Observed   Estimated

                  3.43       3.000       3.296
                  3.45       9.000       9.319
                  3.47      23.000      18.091
                  3.49      19.000      24.124
                  3.51      24.000      22.099
                  3.53      15.000      13.907
                  3.55       3.000       6.011
                  3.57       4.000       1.784


              Quantiles for Normal Distribution

                          ------Quantile------
               Percent    Observed   Estimated

                  20.0     3.46700     3.46830
                  40.0     3.48350     3.48719
                  60.0     3.50450     3.50347
                  80.0     3.52250     3.52236
```

Output 3.19.3. Histogram Superimposed with Normal Curve



The histogram of the variable Thick with a superimposed normal curve is shown in Output 3.19.3.

The estimated parameters for the normal curve ($\hat{\mu} = 3.50$ and $\hat{\sigma} = 0.03$) are shown in Output 3.19.1. By default, the parameters are estimated unless you specify values with the MU= and SIGMA= secondary options after the NORMAL primary option. The results of three goodness-of-fit tests based on the empirical distribution function (EDF) are displayed in Output 3.19.1. Since the p-values are all greater than 0.15, the hypothesis of normality is not rejected.

A sample program, uniex08.sas, for this example is available in the SAS Sample Library for Base SAS software.

Example 3.20. Adding Fitted Normal Curves to a Comparative Histogram

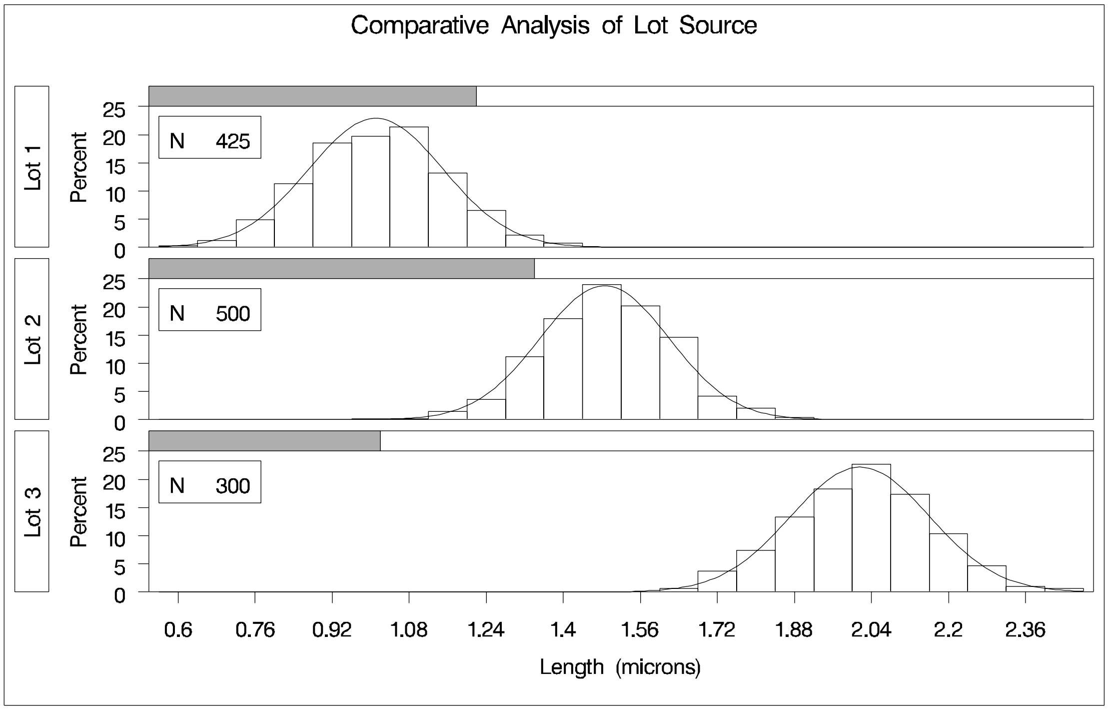
This example is a continuation of Example 3.15, which introduced the data set Channel on page 334. In Output 3.15.3, it appears that the channel lengths in each lot are normally distributed. The following statements use the NORMAL option to fit a normal distribution for each lot:

```
title 'Comparative Analysis of Lot Source';
proc univariate data=Channel noprint;
   class Lot;
   histogram Length / nrows        = 3
                      intertile    = 1
                      cprop        = orange
                      normal(color = black noprint);
   inset n = "N" / pos = nw;
run;
```

The NOPRINT option in the PROC UNIVARIATE statement suppresses the tables of statistical output produced by default; the NOPRINT option in parentheses after the NORMAL option suppresses the tables of statistical output related to the fit of the normal distribution. The normal parameters are estimated from the data for each lot, and the curves are superimposed on each component histogram. The INTERTILE= option specifies the space between the framed areas, which are referred to as tiles. The CPROP= option requests the shaded bars above each tile, which represent the relative frequencies of observations in each lot. The comparative histogram is displayed in Output 3.20.1.

A sample program, uniex09.sas, for this example is available in the SAS Sample Library for Base SAS software.

Output 3.20.1. Fitting Normal Curves to a Comparative Histogram



Example 3.21. Fitting a Beta Curve

You can use a beta distribution to model the distribution of a variable that is known to vary between lower and upper bounds. In this example, a manufacturing company uses a robotic arm to attach hinges on metal sheets. The attachment point should be offset 10.1 mm from the left edge of the sheet. The actual offset varies between 10.0 and 10.5 mm due to variation in the arm. The following statements save the offsets for 50 attachment points as the values of the variable Length in the data set Robots:

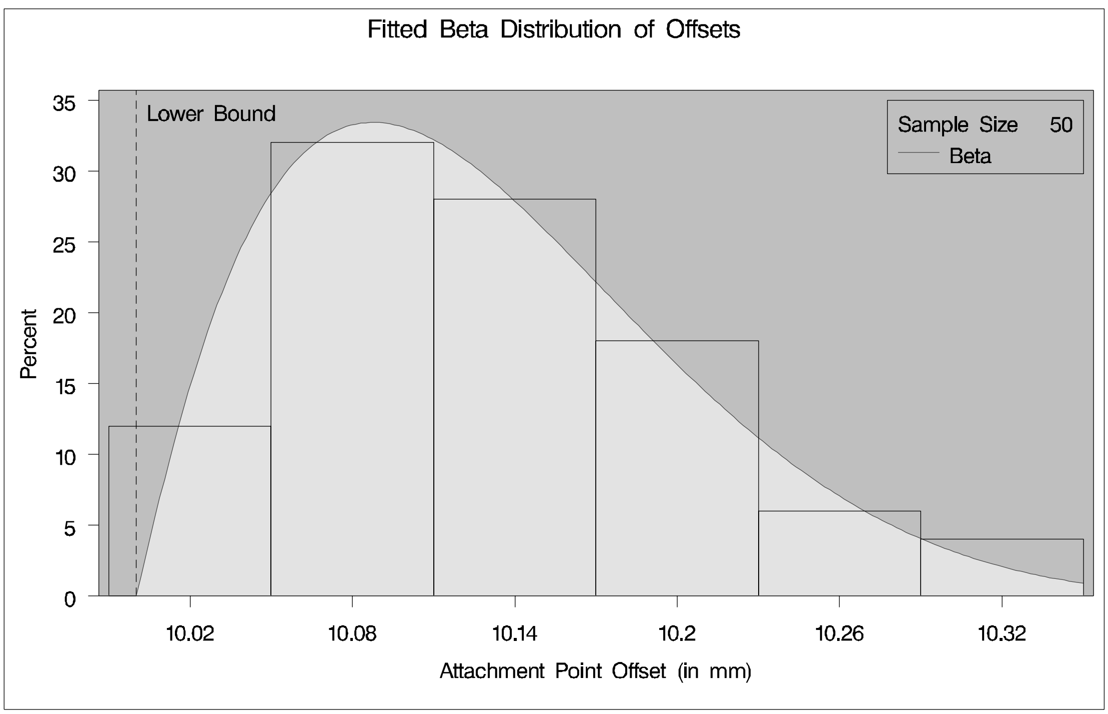
```
data Robots;
   input Length @@;
   label Length = 'Attachment Point Offset (in mm)';
   datalines;
10.147 10.070 10.032 10.042 10.102
10.034 10.143 10.278 10.114 10.127
10.122 10.018 10.271 10.293 10.136
10.240 10.205 10.186 10.186 10.080
10.158 10.114 10.018 10.201 10.065
10.061 10.133 10.153 10.201 10.109
10.122 10.139 10.090 10.136 10.066
10.074 10.175 10.052 10.059 10.077
10.211 10.122 10.031 10.322 10.187
10.094 10.067 10.094 10.051 10.174
;
run;
```

The following statements create a histogram with a fitted beta density curve, shown in Output 3.21.1:

```
title 'Fitted Beta Distribution of Offsets';
ods select ParameterEstimates FitQuantiles MyHist;
proc univariate data=Robots;
   histogram Length /
      beta(theta=10 scale=0.5 color=red fill)
      cfill     = yellow
      cframe    = ligr
      href      = 10
      hreflabel = 'Lower Bound'
      lhref     = 2
      vaxis     = axis1
      name      = 'MyHist';
   axis1 label=(a=90 r=0);
   inset n = 'Sample Size'
         beta / pos=ne  cfill=blank;
run;
```

The ODS SELECT statement restricts the output to the "ParameterEstimates" and "FitQuantiles" tables; see the section "ODS Table Names" on page 309. The BETA primary option requests a fitted beta distribution. The THETA= secondary option specifies the lower threshold. The SCALE= secondary option specifies the range between the lower threshold and the upper threshold. Note that the default THETA= and SCALE= values are zero and one, respectively.

Output 3.21.1. Superimposing a Histogram with a Fitted Beta Curve



The FILL secondary option specifies that the area under the curve is to be filled with the CFILL= color. (If FILL were omitted, the CFILL= color would be used to fill the histogram bars instead.)

The HREF= option draws a reference line at the lower bound, and the HREFLABEL= option adds the label *Lower Bound*. The LHREF= option specifies a dashed line type for the reference line. The INSET statement adds an inset with the sample size positioned in the northeast corner of the plot.

In addition to displaying the beta curve, the BETA option requests a summary of the curve fit. This summary, which includes parameters for the curve and the observed and estimated quantiles, is shown in Output 3.21.2. A sample program, uniex12.sas, for this example is available in the SAS Sample Library for Base SAS software.

Output 3.21.2. Summary of Fitted Beta Distribution

```
                    Fitted Beta Distribution of Offsets

                       Fitted Distribution for Length

                      Parameters for Beta Distribution

                       Parameter   Symbol   Estimate

                       Threshold   Theta          10
                       Scale       Sigma         0.5
                       Shape       Alpha     2.06832
                       Shape       Beta     6.022479
                       Mean                 10.12782
                       Std Dev              0.072339


                       Quantiles for Beta Distribution

                                 ------Quantile------
                       Percent   Observed   Estimated

                           1.0    10.0180     10.0124
                           5.0    10.0310     10.0285
                          10.0    10.0380     10.0416
                          25.0    10.0670     10.0718
                          50.0    10.1220     10.1174
                          75.0    10.1750     10.1735
                          90.0    10.2255     10.2292
                          95.0    10.2780     10.2630
                          99.0    10.3220     10.3237
```

Example 3.22. Fitting Lognormal, Weibull, and Gamma Curves

To determine an appropriate model for a data distribution, you should consider curves from several distribution families. As shown in this example, you can use the HISTOGRAM statement to fit more than one distribution and display the density curves on a histogram.

The gap between two plates is measured (in cm) for each of 50 welded assemblies selected at random from the output of a welding process. The following statements save the measurements (Gap) in a data set named Plates:

```
data Plates;
   label Gap = 'Plate Gap in cm';
   input Gap @@;
   datalines;
0.746  0.357  0.376  0.327  0.485 1.741  0.241  0.777  0.768  0.409
0.252  0.512  0.534  1.656  0.742 0.378  0.714  1.121  0.597  0.231
0.541  0.805  0.682  0.418  0.506 0.501  0.247  0.922  0.880  0.344
0.519  1.302  0.275  0.601  0.388 0.450  0.845  0.319  0.486  0.529
1.547  0.690  0.676  0.314  0.736 0.643  0.483  0.352  0.636  1.080
;
run;
```

The following statements fit three distributions (lognormal, Weibull, and gamma) and display their density curves on a single histogram:

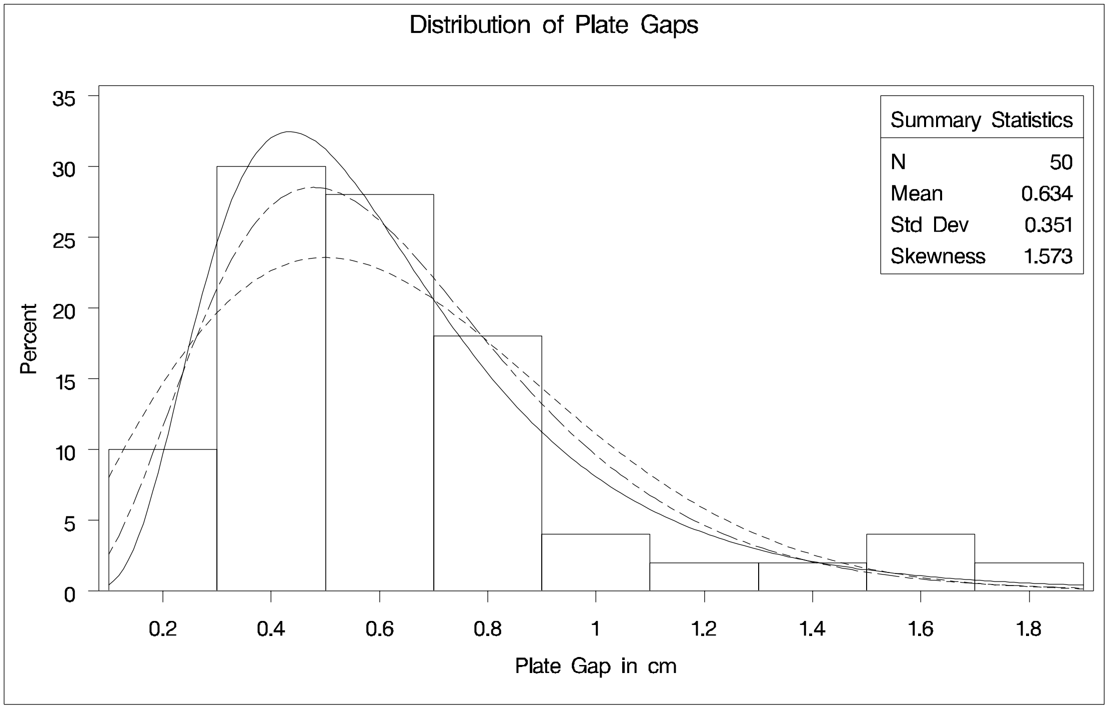
```
title 'Distribution of Plate Gaps';
ods select ParameterEstimates GoodnessOfFit FitQuantiles MyHist;
proc univariate data=Plates;
   var Gap;
   histogram / midpoints=0.2 to 1.8 by 0.2
               lognormal (l=1)
               weibull   (l=2)
               gamma     (l=8)
               vaxis   = axis1
               name    = 'MyHist';
   inset n mean(5.3) std='Std Dev'(5.3) skewness(5.3)
          / pos = ne  header = 'Summary Statistics';
   axis1 label=(a=90 r=0);
run;
```

The ODS SELECT statement restricts the output to the "ParameterEstimates," "GoodnessOfFit," and "FitQuantiles" tables; see the section "ODS Table Names" on page 309. The LOGNORMAL, WEIBULL, and GAMMA primary options request superimposed fitted curves on the histogram in Output 3.22.1. The L= secondary options specify distinct line types for the curves. Note that a threshold parameter $\theta = 0$ is assumed for each curve. In applications where the threshold is not zero, you can specify θ with the THETA= secondary option.

The LOGNORMAL, WEIBULL, and GAMMA options also produce the summaries for the fitted distributions shown in Output 3.22.2 through Output 3.22.5.

Output 3.22.2 provides three EDF goodness-of-fit tests for the lognormal distribution: the Anderson-Darling, the Cramér-von Mises, and the Kolmogorov-Smirnov tests. At the $\alpha = 0.10$ significance level, all tests support the conclusion that the two-parameter lognormal distribution with scale parameter $\hat{\zeta} = -0.58$ and shape parameter $\hat{\sigma} = 0.50$ provides a good model for the distribution of plate gaps.

Output 3.22.1. Superimposing a Histogram with Fitted Curves



Output 3.22.2. Summary of Fitted Lognormal Distribution

```
                       Distribution of Plate Gaps

                      Fitted Distributions for Gap

                  Parameters for Lognormal Distribution

                      Parameter   Symbol   Estimate

                      Threshold   Theta           0
                      Scale       Zeta     -0.58375
                      Shape       Sigma    0.499546
                      Mean                 0.631932
                      Std Dev              0.336436


             Goodness-of-Fit Tests for Lognormal Distribution

        Test                  ---Statistic----   -----p Value-----

        Kolmogorov-Smirnov    D     0.06441431   Pr > D      >0.150
        Cramer-von Mises      W-Sq  0.02823022   Pr > W-Sq   >0.500
        Anderson-Darling      A-Sq  0.24308402   Pr > A-Sq   >0.500
```

Output 3.22.3. Summary of Fitted Lognormal Distribution (cont.)

Distribution of Plate Gaps

Fitted Distributions for Gap

Quantiles for Lognormal Distribution

Percent	Quantile Observed	Quantile Estimated
1.0	0.23100	0.17449
5.0	0.24700	0.24526
10.0	0.29450	0.29407
25.0	0.37800	0.39825
50.0	0.53150	0.55780
75.0	0.74600	0.78129
90.0	1.10050	1.05807
95.0	1.54700	1.26862
99.0	1.74100	1.78313

Output 3.22.4. Summary of Fitted Weibull Distribution

Distribution of Plate Gaps

Fitted Distributions for Gap

Parameters for Weibull Distribution

Parameter	Symbol	Estimate
Threshold	Theta	0
Scale	Sigma	0.719208
Shape	C	1.961159
Mean		0.637641
Std Dev		0.339248

Goodness-of-Fit Tests for Weibull Distribution

Test	---Statistic----		-----p Value-----	
Cramer-von Mises	W-Sq	0.15937281	Pr > W-Sq	0.016
Anderson-Darling	A-Sq	1.15693542	Pr > A-Sq	<0.010

Quantiles for Weibull Distribution

Percent	Quantile Observed	Quantile Estimated
1.0	0.23100	0.06889
5.0	0.24700	0.15817
10.0	0.29450	0.22831
25.0	0.37800	0.38102
50.0	0.53150	0.59661
75.0	0.74600	0.84955
90.0	1.10050	1.10040
95.0	1.54700	1.25842
99.0	1.74100	1.56691

Output 3.22.4 provides two EDF goodness-of-fit tests for the Weibull distribution: the Anderson-Darling and the Cramér-von Mises tests. The p-values for the EDF tests are all less than 0.10, indicating that the data do not support a Weibull model.

Output 3.22.5. Summary of Fitted Gamma Distribution

```
                         Distribution of Plate Gaps

                        Fitted Distributions for Gap

                     Parameters for Gamma Distribution

                       Parameter   Symbol   Estimate

                       Threshold   Theta           0
                       Scale       Sigma    0.155198
                       Shape       Alpha    4.082646
                       Mean                  0.63362
                       Std Dev              0.313587


                Goodness-of-Fit Tests for Gamma Distribution

         Test                  ---Statistic----   -----p Value-----

         Kolmogorov-Smirnov    D     0.09695325   Pr > D     >0.250
         Cramer-von Mises      W-Sq  0.07398467   Pr > W-Sq  >0.250
         Anderson-Darling      A-Sq  0.58106613   Pr > A-Sq   0.137


                      Quantiles for Gamma Distribution

                                  ------Quantile------
                       Percent    Observed   Estimated

                           1.0     0.23100     0.13326
                           5.0     0.24700     0.21951
                          10.0     0.29450     0.27938
                          25.0     0.37800     0.40404
                          50.0     0.53150     0.58271
                          75.0     0.74600     0.80804
                          90.0     1.10050     1.05392
                          95.0     1.54700     1.22160
                          99.0     1.74100     1.57939
```

Output 3.22.5 provides three EDF goodness-of-fit tests for the gamma distribution: the Anderson-Darling, the Cramér-von Mises, and the Kolmogorov-Smirnov tests. At the $\alpha = 0.10$ significance level, all tests support the conclusion that the gamma distribution with scale parameter $\sigma = 0.16$ and shape parameter $\alpha = 4.08$ provides a good model for the distribution of plate gaps.

Based on this analysis, the fitted lognormal distribution and the fitted gamma distribution are both good models for the distribution of plate gaps. A sample program, uniex13.sas, for this example is available in the SAS Sample Library for Base SAS software.

Example 3.23. Computing Kernel Density Estimates

This example illustrates the use of kernel density estimates to visualize a nonnormal data distribution. This example uses the data set Channel, which is introduced in Example 3.15.

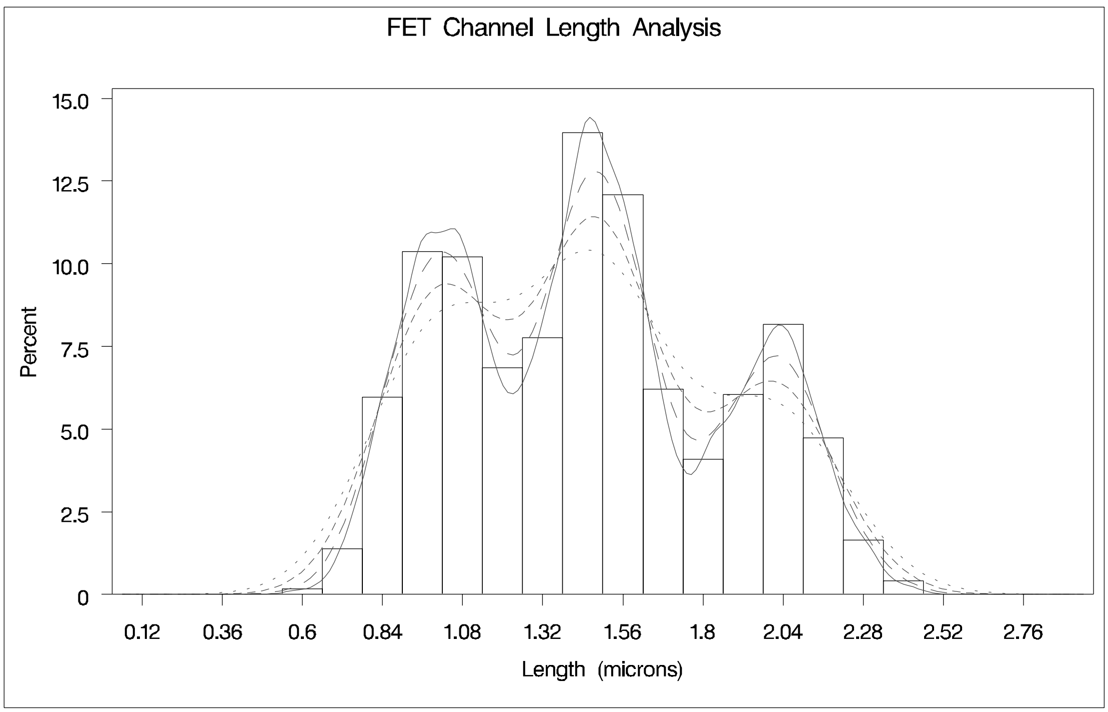
When you compute kernel density estimates, you should try several choices for the bandwidth parameter c since this determines the smoothness and closeness of the fit.

You can specify a list of up to five C= values with the KERNEL option to request multiple density estimates, as shown in the following statements:

```
title 'FET Channel Length Analysis';
proc univariate data=Channel noprint;
   histogram Length / kernel(c = 0.25 0.50 0.75 1.00
                             l = 1 20 2 34
                             color=red
                             noprint);
run;
```

The L= secondary option specifies distinct line types for the curves (the L= values are paired with the C= values in the order listed). Output 3.23.1 demonstrates the effect of c. In general, larger values of c yield smoother density estimates, and smaller values yield estimates that more closely fit the data distribution.

Output 3.23.1. Multiple Kernel Density Estimates



Output 3.23.1 reveals strong trimodality in the data, which is displayed with comparative histograms in Example 3.15.

A sample program, uniex09.sas, for this example is available in the SAS Sample Library for Base SAS software.

Example 3.24. Fitting a Three-Parameter Lognormal Curve

If you request a lognormal fit with the LOGNORMAL primary option, a two-parameter lognormal distribution is assumed. This means that the shape parameter σ and the scale parameter ζ are unknown (unless specified) and that the threshold θ is known (it is either specified with the THETA= option or assumed to be zero).

If it is necessary to estimate θ in addition to ζ and σ, the distribution is referred to as a three-parameter lognormal distribution. This example shows how you can request a three-parameter lognormal distribution.

A manufacturing process produces a plastic laminate whose strength must exceed a minimum of 25 psi. Samples are tested, and a lognormal distribution is observed for the strengths. It is important to estimate θ to determine whether the process meets the strength requirement. The following statements save the strengths for 49 samples in the data set Plastic:

```
data Plastic;
   label Strength = 'Strength in psi';
   input Strength @@;
   datalines;
30.26 31.23 71.96 47.39 33.93 76.15 42.21
81.37 78.48 72.65 61.63 34.90 24.83 68.93
43.27 41.76 57.24 23.80 34.03 33.38 21.87
31.29 32.48 51.54 44.06 42.66 47.98 33.73
25.80 29.95 60.89 55.33 39.44 34.50 73.51
43.41 54.67 99.43 50.76 48.81 31.86 33.88
35.57 60.41 54.92 35.66 59.30 41.96 45.32
;
run;
```

The following statements use the LOGNORMAL primary option in the HISTOGRAM statement to display the fitted three-parameter lognormal curve shown in Output 3.24.1:

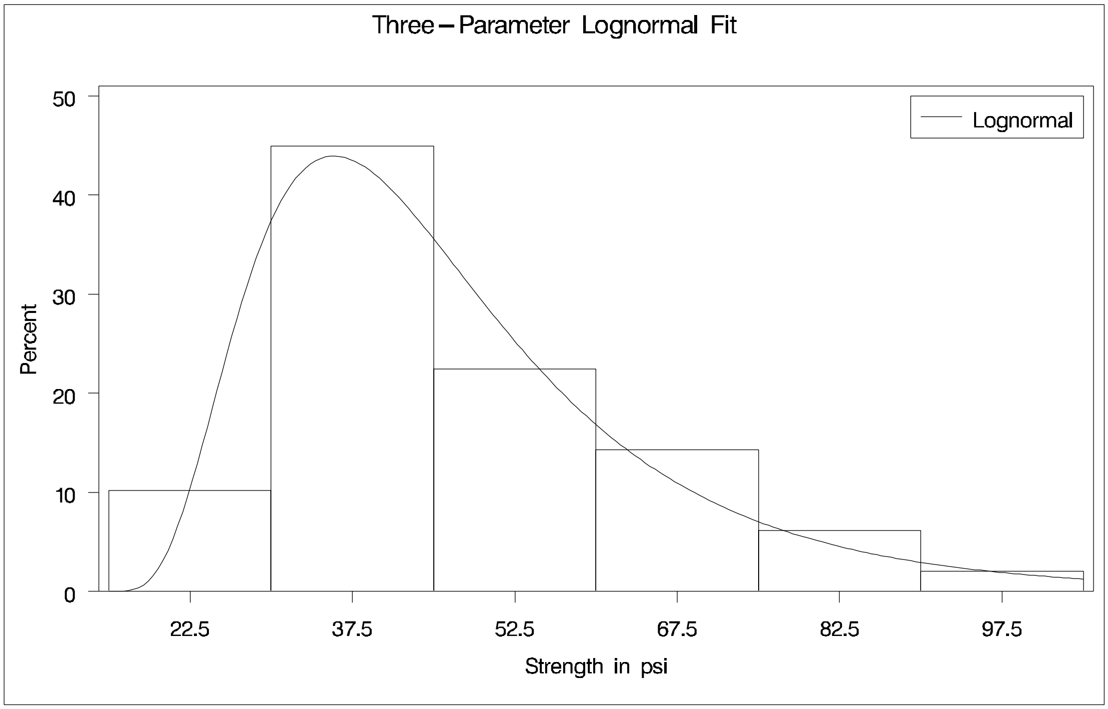
```
title 'Three-Parameter Lognormal Fit';
proc univariate data=Plastic noprint;
   histogram Strength / lognormal(fill theta = est noprint)
                        cfill = white;
   inset lognormal    / format=6.2 pos=ne;
run;
```

The NOPRINT option suppresses the tables of statistical output produced by default. Specifying THETA=EST requests a local maximum likelihood estimate (LMLE) for θ, as described by Cohen (1951). This estimate is then used to compute maximum likelihood estimates for σ and ζ.

Note: You can also specify THETA=EST with the WEIBULL primary option to fit a three-parameter Weibull distribution.

A sample program, uniex14.sas, for this example is available in the SAS Sample Library for Base SAS software.

Output 3.24.1. Three-Parameter Lognormal Fit



Example 3.25. Annotating a Folded Normal Curve

This example shows how to display a fitted curve that is not supported by the HISTOGRAM statement. The offset of an attachment point is measured (in mm) for a number of manufactured assemblies, and the measurements (Offset) are saved in a data set named Assembly. The following statements create the data set Assembly:

```
data Assembly;
   label Offset = 'Offset (in mm)';
   input Offset @@;
   datalines;
11.11 13.07 11.42  3.92 11.08  5.40 11.22 14.69  6.27  9.76
 9.18  5.07  3.51 16.65 14.10  9.69 16.61  5.67  2.89  8.13
 9.97  3.28 13.03 13.78  3.13  9.53  4.58  7.94 13.51 11.43
11.98  3.90  7.67  4.32 12.69  6.17 11.48  2.82 20.42  1.01
 3.18  6.02  6.63  1.72  2.42 11.32 16.49  1.22  9.13  3.34
 1.29  1.70  0.65  2.62  2.04 11.08 18.85 11.94  8.34  2.07
 0.31  8.91 13.62 14.94  4.83 16.84  7.09  3.37  0.49 15.19
 5.16  4.14  1.92 12.70  1.97  2.10  9.38  3.18  4.18  7.22
15.84 10.85  2.35  1.93  9.19  1.39 11.40 12.20 16.07  9.23
 0.05  2.15  1.95  4.39  0.48 10.16  4.81  8.28  5.68 22.81
 0.23  0.38 12.71  0.06 10.11 18.38  5.53  9.36  9.32  3.63
12.93 10.39  2.05 15.49  8.12  9.52  7.77 10.70  6.37  1.91
 8.60 22.22  1.74  5.84 12.90 13.06  5.08  2.09  6.41  1.40
15.60  2.36  3.97  6.17  0.62  8.56  9.36 10.19  7.16  2.37
12.91  0.95  0.89  3.82  7.86  5.33 12.92  2.64  7.92 14.06
;
run;
```

It is decided to fit a *folded normal distribution* to the offset measurements. A variable X has a folded normal distribution if $X = |Y|$, where Y is distributed as $N(\mu, \sigma)$. The fitted density is

$$h(x) = \frac{1}{\sqrt{2\pi}\sigma}\left[\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) + \exp\left(-\frac{(x+\mu)^2}{2\sigma^2}\right)\right] \quad , \ x \geq 0$$

You can use SAS/IML to compute preliminary estimates of μ and σ based on a method of moments given by Elandt (1961). These estimates are computed by solving equation (19) of Elandt (1961), which is given by

$$f(\theta) = \frac{\left(\frac{2}{\sqrt{2\pi}}e^{-\theta^2/2} - \theta\left[1 - 2\Phi(\theta)\right]\right)^2}{1+\theta^2} = A$$

where $\Phi(\cdot)$ is the standard normal distribution function, and

$$A = \frac{\bar{x}^2}{\frac{1}{n}\sum_{i=1}^{n} x_i^2}$$

Then the estimates of σ and μ are given by

$$\begin{aligned} \hat{\sigma}_0 &= \sqrt{\frac{\frac{1}{n}\sum_{i=1}^{n} x_i^2}{1+\hat{\theta}^2}} \\ \hat{\mu}_0 &= \hat{\theta} \cdot \hat{\sigma}_0 \end{aligned}$$

Begin by using PROC MEANS to compute the first and second moments and using the following DATA step to compute the constant A:

```
proc means data = Assembly noprint;
   var Offset;
   output out=stat mean=m1 var=var n=n min = min;
run;

* Compute constant A from equation (19) of Elandt (1961);
data stat;
   keep m2 a min;
   set stat;
   a  = (m1*m1);
   m2 = ((n-1)/n)*var + a;
   a  = a/m2;
run;
```

Next, use the SAS/IML subroutine NLPDD to solve equation (19) by minimizing $(f(\theta) - A)^2$, and compute $\hat{\mu}_0$ and $\hat{\sigma}_0$:

```
proc iml;
   use stat;
   read all var {m2}  into m2;
   read all var {a}   into a;
   read all var {min} into min;

   * f(t) is the function in equation (19) of Elandt (1961);
   start f(t) global(a);
     y = .39894*exp(-0.5*t*t);
     y = (2*y-(t*(1-2*probnorm(t))))**2/(1+t*t);
     y = (y-a)**2;
     return(y);
   finish;

   * Minimize (f(t)-A)**2 and estimate mu and sigma;
   if ( min < 0 ) then do;
      print "Warning: Observations are not all nonnegative.";
      print "     The folded normal is inappropriate.";
      stop;
      end;
   if ( a < 0.637 ) then do;
      print "Warning: the folded normal may be inappropriate";
      end;
   opt = { 0 0 };
   con = { 1e-6 };
   x0  = { 2.0 };
   tc  = { . . . . . . 1e-8 . . . . . . . .};
   call nlpdd(rc,etheta0,"f",x0,opt,con,tc);
   esig0 = sqrt(m2/(1+etheta0*etheta0));
   emu0  = etheta0*esig0;

   create prelim var {emu0 esig0 etheta0};
   append;
   close prelim;
```

The preliminary estimates are saved in the data set Prelim, as shown in Output 3.25.1:

Output 3.25.1. Preliminary Estimates of μ, σ, and θ

The Data Set Prelim

EMU0	ESIG0	ETHETA0
6.51735	6.54953	0.99509

Now, using $\hat{\mu}_0$ and $\hat{\sigma}_0$ as initial estimates, call the NLPDD subroutine to maximize the log likelihood, $l(\mu, \sigma)$, of the folded normal distribution, where, up to a constant,

$$l(\mu, \sigma) = -n \log \sigma + \sum_{i=1}^{n} \log \left[\exp \left(-\frac{(x_i - \mu)^2}{2\sigma^2} \right) + \exp \left(-\frac{(x_i + \mu)^2}{2\sigma^2} \right) \right]$$

```
* Define the log likelihood of the folded normal;
start g(p) global(x);
   y = 0.0;
   do i = 1 to nrow(x);
      z = exp( (-0.5/p[2])*(x[i]-p[1])*(x[i]-p[1]) );
      z = z + exp( (-0.5/p[2])*(x[i]+p[1])*(x[i]+p[1]) );
      y = y + log(z);
   end;
   y = y - nrow(x)*log( sqrt( p[2] ) );
   return(y);
finish;

* Maximize the log likelihood with subroutine NLPDD;
use assembly;
read all var {offset} into x;
esig0sq = esig0*esig0;
x0      = emu0||esig0sq;
opt     = { 1 0 };
con     = { . 0.0, .  .  };
call nlpdd(rc,xr,"g",x0,opt,con);
emu     = xr[1];
esig    = sqrt(xr[2]);
etheta  = emu/esig;

create parmest var{emu esig etheta};
append;
close parmest;
quit;
```

The data set ParmEst contains the maximum likelihood estimates $\hat{\mu}$ and $\hat{\sigma}$ (as well as $\hat{\mu}/\hat{\sigma}$), as shown in Output 3.25.2:

Output 3.25.2. Final Estimates of μ, σ, and θ

The Data Set ParmEst

EMU	ESIG	ETHETA
6.66761	6.39650	1.04239

To annotate the curve on a histogram, begin by computing the width and endpoints of the histogram intervals. The following statements save these values in a data set called OutCalc. Note that a plot is not produced at this point.

```
proc univariate data = Assembly noprint;
   histogram Offset / outhistogram = out normal(noprint) noplot;
run;

data OutCalc (drop = _MIDPT_);
   set out (keep = _MIDPT_) end = eof;
   retain _MIDPT1_ _WIDTH_;
   if _N_ = 1 then _MIDPT1_ = _MIDPT_;
   if eof then do;
      _MIDPTN_ = _MIDPT_;
   _WIDTH_ = (_MIDPTN_ - _MIDPT1_) / (_N_ - 1);
   output;
   end;
run;
```

Output 3.25.3 provides a listing of the data set OutCalc. The width of the histogram bars is saved as the value of the variable _WIDTH_; the midpoints of the first and last histogram bars are saved as the values of the variables _MIDPT1_ and _MIDPTN_.

Output 3.25.3. The Data Set OutCalc

```
                         Data Set OutCalc

                 _MIDPT1_     _WIDTH_     _MIDPTN_

                    1.5          3          22.5
```

The following statements create an annotate data set named Anno, which contains the coordinates of the fitted curve:

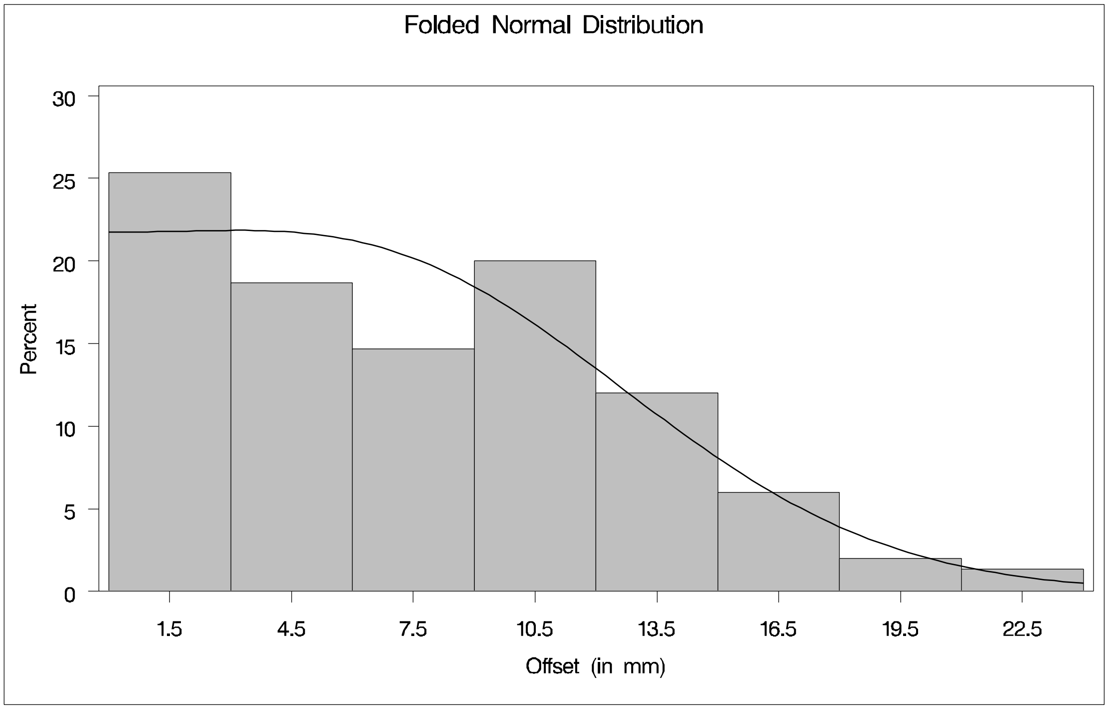
```
data Anno;
   merge ParmEst OutCalc;
   length function color $ 8;
   function = 'point';
   color    = 'black';
   size     =  2;
   xsys     = '2';
   ysys     = '2';
   when     = 'a';
   constant = 39.894*_width_;;
   left     =  _midpt1_ - .5*_width_;
   right    =  _midptn_ + .5*_width_;
   inc      = (right-left)/100;
   do x = left to right by inc;
      z1 = (x-emu)/esig;
      z2 = (x+emu)/esig;
      y  = (constant/esig)*(exp(-0.5*z1*z1)+exp(-0.5*z2*z2));
      output;
      function = 'draw';
   end;
run;
```

The following statements read the ANNOTATE= data set and display the histogram and fitted curve:

```
title 'Folded Normal Distribution';
proc univariate data=assembly noprint;
   histogram Offset / annotate = anno
                      cbarline = black
                      cfill    = ligr;
run;
```

Output 3.25.4 displays the histogram and fitted curve:

Output 3.25.4. Histogram with Annotated Folded Normal Curve



A sample program, uniex15.sas, for this example is available in the SAS Sample Library for Base SAS software.

Example 3.26. Creating Lognormal Probability Plots

This example is a continuation of the example explored in the section "Modeling a Data Distribution" on page 200.

In the normal probability plot shown in Figure 3.6, the nonlinearity of the point pattern indicates a departure from normality in the distribution of Deviation. Since the point pattern is curved with slope increasing from left to right, a theoretical distribution that is skewed to the right, such as a lognormal distribution, should provide a better fit than the normal distribution. See the section "Interpretation of Quantile-Quantile and Probability Plots" on page 299.

You can explore the possibility of a lognormal fit with a lognormal probability plot. When you request such a plot, you must specify the shape parameter σ for the lognormal distribution. This value must be positive, and typical values of σ range from 0.1 to 1.0. You can specify values for σ with the SIGMA= secondary option in the LOGNORMAL primary option, or you can specify that σ is to be estimated from the data.

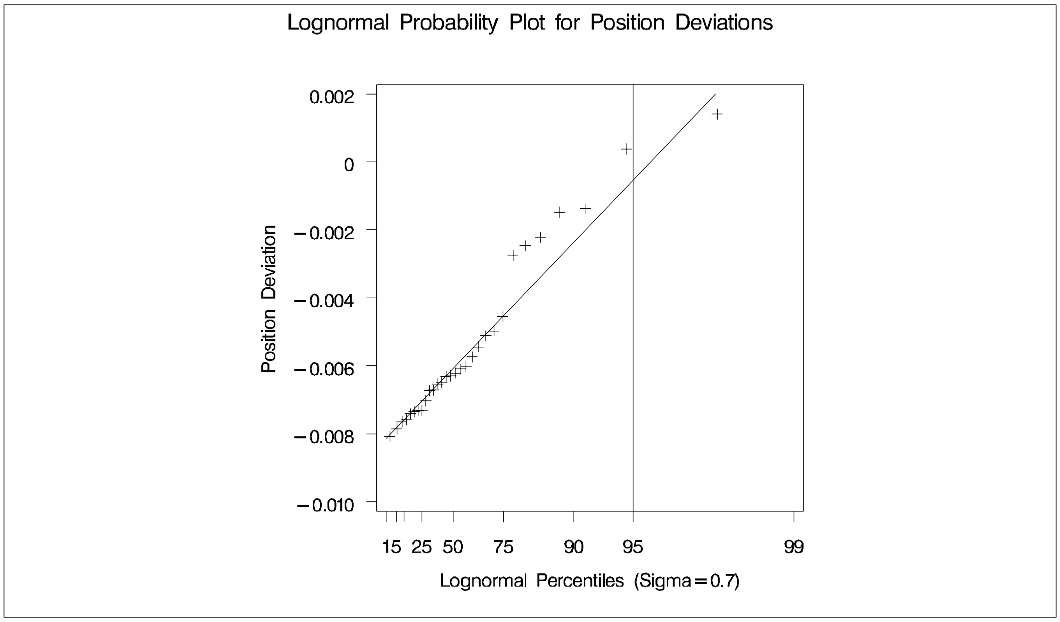
The following statements illustrate the first approach by creating a series of three lognormal probability plots for the variable Deviation introduced in the section "Modeling a Data Distribution" on page 200:

```
symbol v=plus height=3.5pct;
title 'Lognormal Probability Plot for Position Deviations';
proc univariate data=Aircraft noprint;
   probplot Deviation /
      lognormal(theta=est zeta=est sigma=0.7 0.9 1.1)
      href  = 95
      lhref = 1
      square;
run;
```

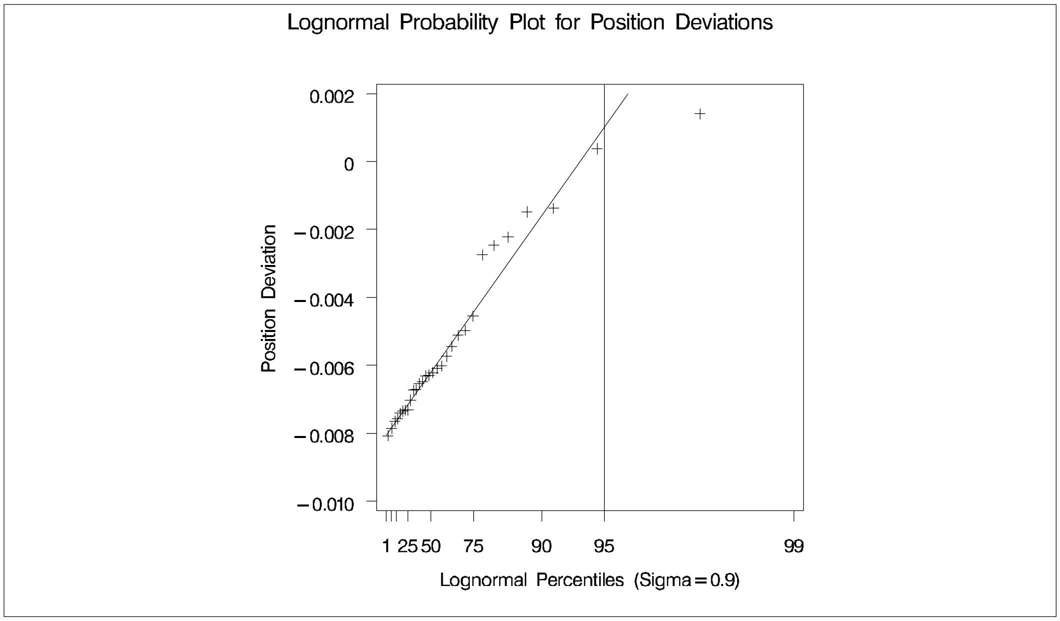
The LOGNORMAL primary option requests plots based on the lognormal family of distributions, and the SIGMA= secondary option requests plots for σ equal to 0.7, 0.9, and 1.1. These plots are displayed in Output 3.26.1, Output 3.26.2, and Output 3.26.3, respectively. Alternatively, you can specify σ to be estimated using the sample standard deviation by using the option SIGMA=EST.

The SQUARE option displays the probability plot in a square format, the HREF= option requests a reference line at the 95th percentile, and the LHREF= option specifies the line type for the reference line.

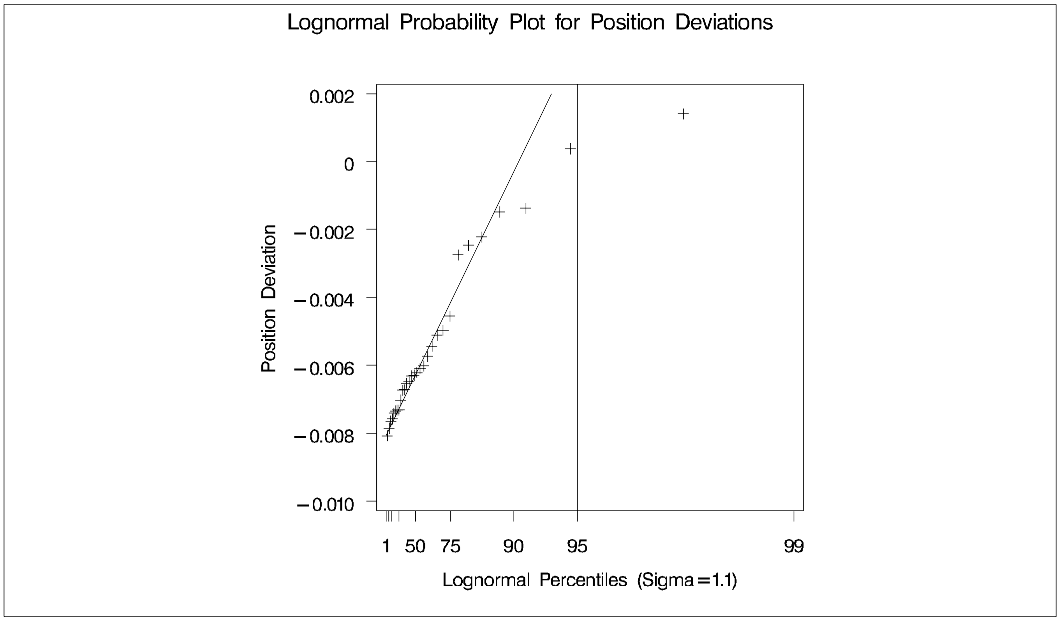
Output 3.26.1. Probability Plot Based on Lognormal Distribution with σ =0.7



Output 3.26.2. Probability Plot Based on Lognormal Distribution with σ =0.9



Output 3.26.3. Probability Plot Based on Lognormal Distribution with σ =1.1



The value $\sigma = 0.9$ in Output 3.26.2 most nearly linearizes the point pattern. The 95th percentile of the position deviation distribution seen in Output 3.26.2 is approximately 0.001, since this is the value corresponding to the intersection of the point pattern with the reference line.

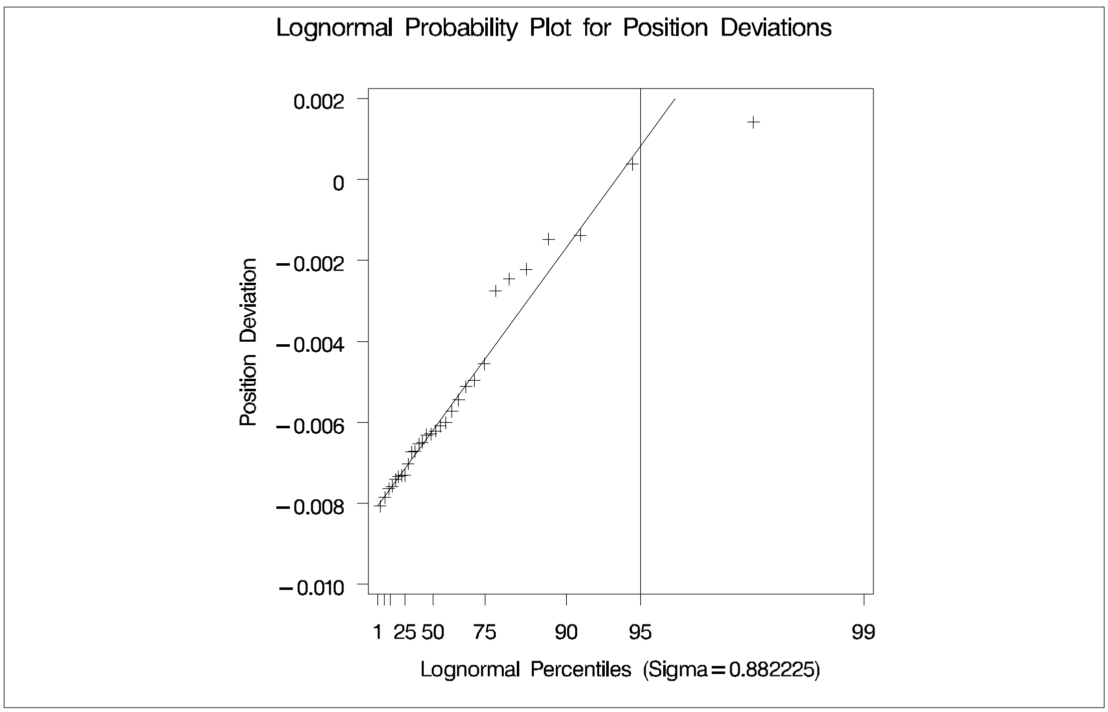
Note: Once the σ that produces the most linear fit is found, you can then estimate the threshold parameter θ and the scale parameter ζ. See Example 3.31.

The following statements illustrate how you can create a lognormal probability plot for Deviation using a local maximum likelihood estimate for σ.

```
symbol v=plus height=3.5pct;
title 'Lognormal Probability Plot for Position Deviations';
proc univariate data=Aircraft noprint;
   probplot Deviation / lognormal(theta=est zeta=est sigma=est)
                        href   = 95
                        lhref  = 1
                        square;
run;
```

The plot is displayed in Output 3.26.4. Note that the maximum likelihood estimate of σ (in this case 0.882) does not necessarily produce the most linear point pattern.

Output 3.26.4. Probability Plot Based on Lognormal Distribution with Estimated σ



A sample program, uniex16.sas, for this example is available in the SAS Sample Library for Base SAS software.

Example 3.27. Creating a Histogram to Display Lognormal Fit

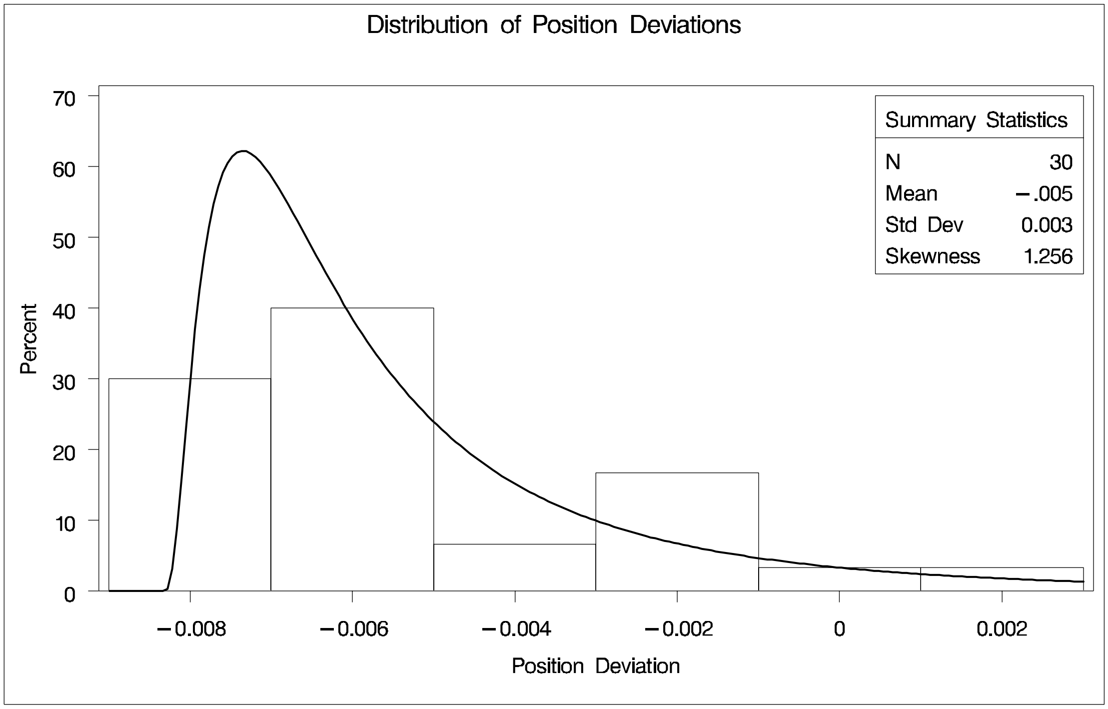
This example uses the data set Aircraft from the previous example to illustrate how to display a lognormal fit with a histogram. To determine whether the lognormal distribution is an appropriate model for a distribution, you should consider the graphical fit as well as conduct goodness-of-fit tests.

The following statements fit a lognormal distribution and display the density curve on a histogram:

```
title 'Distribution of Position Deviations';
ods select Lognormal.ParameterEstimates Lognormal.GoodnessOfFit MyPlot;
proc univariate data=Aircraft;
   var Deviation;
   histogram / lognormal(w=3 theta=est)
               vaxis = axis1
               name  = 'MyPlot';
   inset n mean (5.3) std='Std Dev' (5.3) skewness (5.3) /
         pos = ne  header = 'Summary Statistics';
   axis1 label=(a=90 r=0);
run;
```

The ODS SELECT statement restricts the output to the "ParameterEstimates" and "GoodnessOfFit" tables; see the section "ODS Table Names" on page 309. The LOGNORMAL primary option superimposes a fitted curve on the histogram in Output 3.27.1. The W= option specifies the line width for the curve. The INSET statement specifies that the mean, standard deviation, and skewness be displayed in an inset in the northeast corner of the plot. Note that the default value of the threshold parameter θ is zero. In applications where the threshold is not zero, you can specify θ with the THETA= option. The variable Deviation includes values that are less than the default threshold; therefore, the option THETA= EST is used.

Output 3.27.1. Normal Probability Plot Created with Graphics Device



Output 3.27.2 provides three EDF goodness-of-fit tests for the lognormal distribution: the Anderson-Darling, the Cramér-von Mises, and the Kolmogorov-Smirnov tests. The null hypothesis for the three tests is that a lognormal distribution holds for the sample data.

Output 3.27.2. Summary of Fitted Lognormal Distribution

```
                    Distribution of Position Deviations

                      Fitted Distribution for Deviation

                    Parameters for Lognormal Distribution

                        Parameter   Symbol   Estimate

                        Threshold   Theta    -0.00834
                        Scale       Zeta     -6.14382
                        Shape       Sigma    0.882225
                        Mean                 -0.00517
                        Std Dev              0.003438


               Goodness-of-Fit Tests for Lognormal Distribution

          Test                  ---Statistic----   -----p Value-----

          Kolmogorov-Smirnov    D     0.09419634   Pr > D      >0.500
          Cramer-von Mises      W-Sq  0.02919815   Pr > W-Sq   >0.500
          Anderson-Darling      A-Sq  0.21606642   Pr > A-Sq   >0.500
```

The p-values for all three tests are greater than 0.5, so the null hypothesis is not rejected. The tests support the conclusion that the two-parameter lognormal distribution with scale parameter $\hat{\zeta} = -6.14$, and shape parameter $\hat{\sigma} = 0.88$ provides a good model for the distribution of position deviations. For further discussion of goodness-of-fit interpretation, see the section "Goodness-of-Fit Tests" on page 292.

A sample program, uniex16.sas, for this example is available in the SAS Sample Library for Base SAS software.

Example 3.28. Creating a Normal Quantile Plot

This example illustrates how to create a normal quantile plot. An engineer is analyzing the distribution of distances between holes cut in steel sheets. The following statements save measurements of the distance between two holes cut into 50 steel sheets as values of the variable Distance in the data set Sheets:

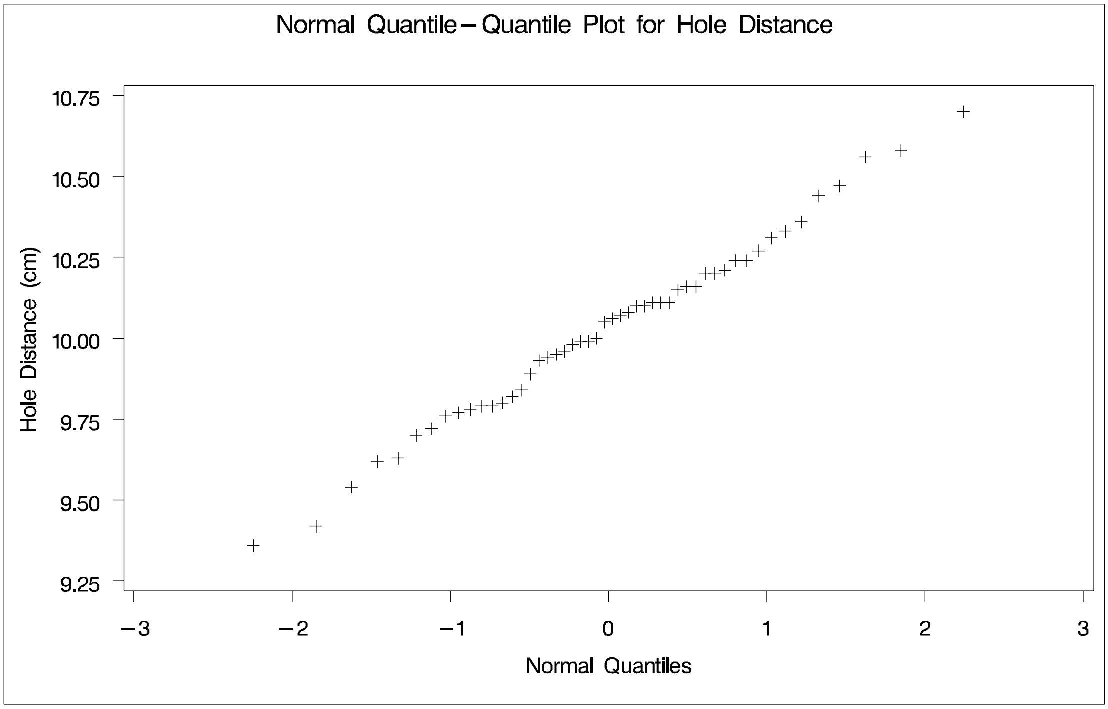
```
data Sheets;
   input Distance @@;
   label Distance = 'Hole Distance (cm)';
   datalines;
 9.80 10.20 10.27  9.70  9.76
10.11 10.24 10.20 10.24  9.63
 9.99  9.78 10.10 10.21 10.00
 9.96  9.79 10.08  9.79 10.06
10.10  9.95  9.84 10.11  9.93
10.56 10.47  9.42 10.44 10.16
10.11 10.36  9.94  9.77  9.36
 9.89  9.62 10.05  9.72  9.82
 9.99 10.16 10.58 10.70  9.54
10.31 10.07 10.33  9.98 10.15
;
run;
```

The engineer decides to check whether the distribution of distances is normal. The following statements create a Q-Q plot for Distance, shown in Output 3.28.1:

```
symbol v=plus;
title 'Normal Quantile-Quantile Plot for Hole Distance';
proc univariate data=Sheets noprint;
   qqplot Distance;
run;
```

The plot compares the ordered values of Distance with quantiles of the normal distribution. The linearity of the point pattern indicates that the measurements are normally distributed. Note that a normal Q-Q plot is created by default.

Output 3.28.1. Normal Quantile-Quantile Plot for Distance



A sample program, uniex17.sas, for this example is available in the SAS Sample Library for Base SAS software.

Example 3.29. Adding a Distribution Reference Line

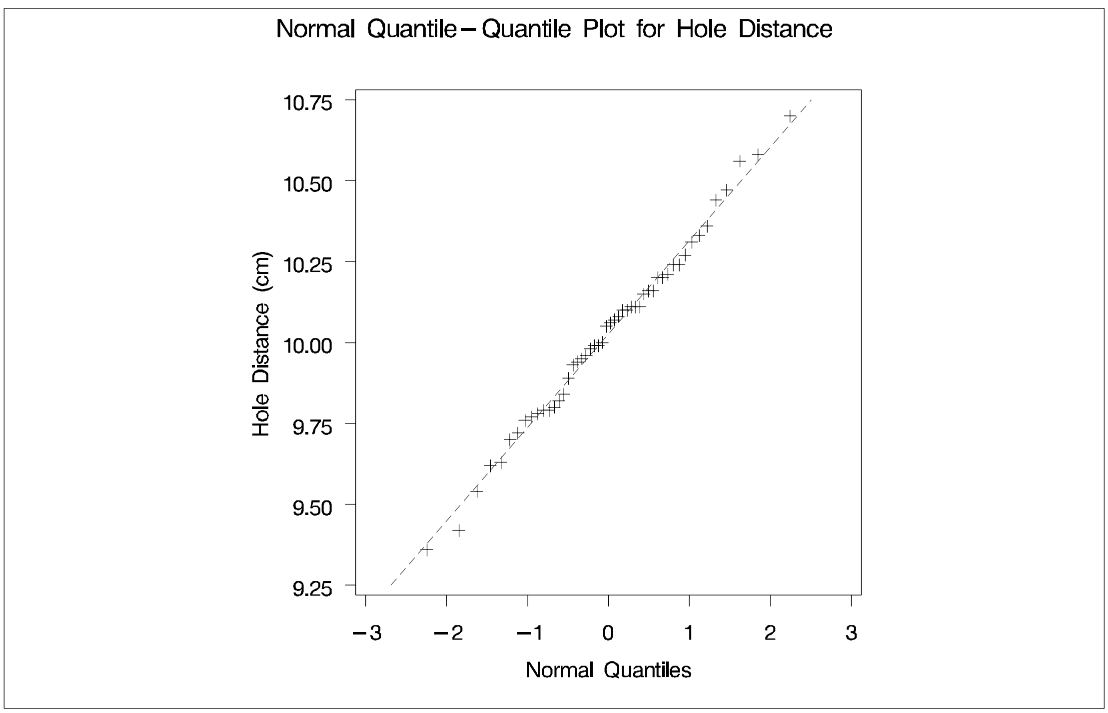
This example, which is a continuation of Example 3.28, illustrates how to add a reference line to a normal Q-Q plot, which represents the normal distribution with mean μ_0 and standard deviation σ_0. The following statements reproduce the Q-Q plot in Output 3.28.1 and add the reference line:

```
symbol v=plus;
title 'Normal Quantile-Quantile Plot for Hole Distance';
proc univariate data=Sheets noprint;
   qqplot Distance / normal(mu=est sigma=est color=red
                            l=2 noprint)
                     square;
run;
```

The plot is displayed in Output 3.29.1.

Specifying MU=EST and SIGMA=EST with the NORMAL primary option requests the reference line for which μ_0 and σ_0 are estimated by the sample mean and standard deviation. Alternatively, you can specify numeric values for μ_0 and σ_0 with the MU= and SIGMA= secondary options. The COLOR= and L= options specify the color and type of the line, and the SQUARE option displays the plot in a square format. The NOPRINT options in the PROC UNIVARIATE statement and after the NORMAL option suppress all the tables of statistical output produced by default.

Output 3.29.1. Adding a Distribution Reference Line to a Q-Q Plot



The data clearly follow the line, which indicates that the distribution of the distances is normal.

A sample program, uniex17.sas, for this example is available in the SAS Sample Library for Base SAS software.

Example 3.30. Interpreting a Normal Quantile Plot

This example illustrates how to interpret a normal quantile plot when the data are from a non-normal distribution. The following statements create the data set Measures, which contains the measurements of the diameters of 50 steel rods in the variable Diameter:

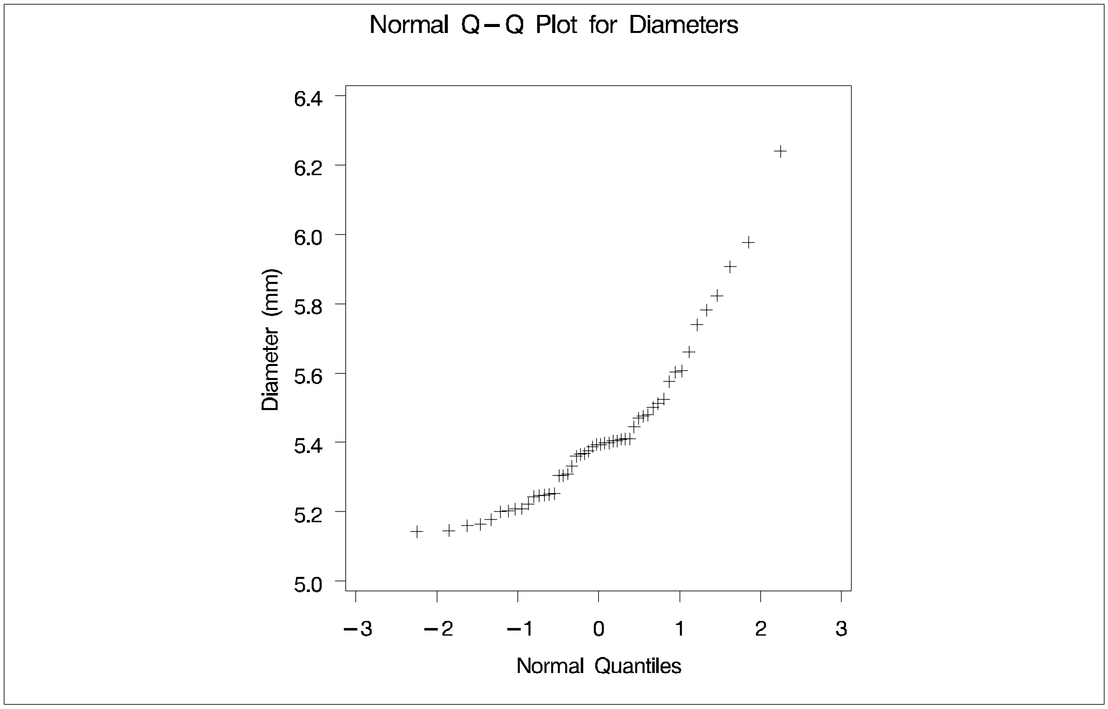
```
data Measures;
   input Diameter @@;
   label Diameter = 'Diameter (mm)';
   datalines;
5.501  5.251  5.404  5.366  5.445  5.576  5.607
5.200  5.977  5.177  5.332  5.399  5.661  5.512
5.252  5.404  5.739  5.525  5.160  5.410  5.823
5.376  5.202  5.470  5.410  5.394  5.146  5.244
5.309  5.480  5.388  5.399  5.360  5.368  5.394
5.248  5.409  5.304  6.239  5.781  5.247  5.907
5.208  5.143  5.304  5.603  5.164  5.209  5.475
5.223
;
run;
```

The following statements request the normal Q-Q plot in Output 3.30.1:

```
symbol v=plus;
title 'Normal Q-Q Plot for Diameters';
proc univariate data=Measures noprint;
   qqplot Diameter / normal(noprint)
                     square
                     vaxis=axis1;
   axis1 label=(a=90 r=0);
run;
```

The nonlinearity of the points in Output 3.30.1 indicates a departure from normality. Since the point pattern is curved with slope increasing from left to right, a theoretical distribution that is skewed to the right, such as a lognormal distribution, should provide a better fit than the normal distribution. The mild curvature suggests that you should examine the data with a series of lognormal Q-Q plots for small values of the shape parameter σ, as illustrated in Example 3.31. For details on interpreting a Q-Q plot, see the section "Interpretation of Quantile-Quantile and Probability Plots" on page 299.

Output 3.30.1. Normal Quantile-Quantile Plot of Nonnormal Data



A sample program, uniex18.sas, for this example is available in the SAS Sample Library for Base SAS software.

Example 3.31. Estimating Three Parameters from Lognormal Quantile Plots

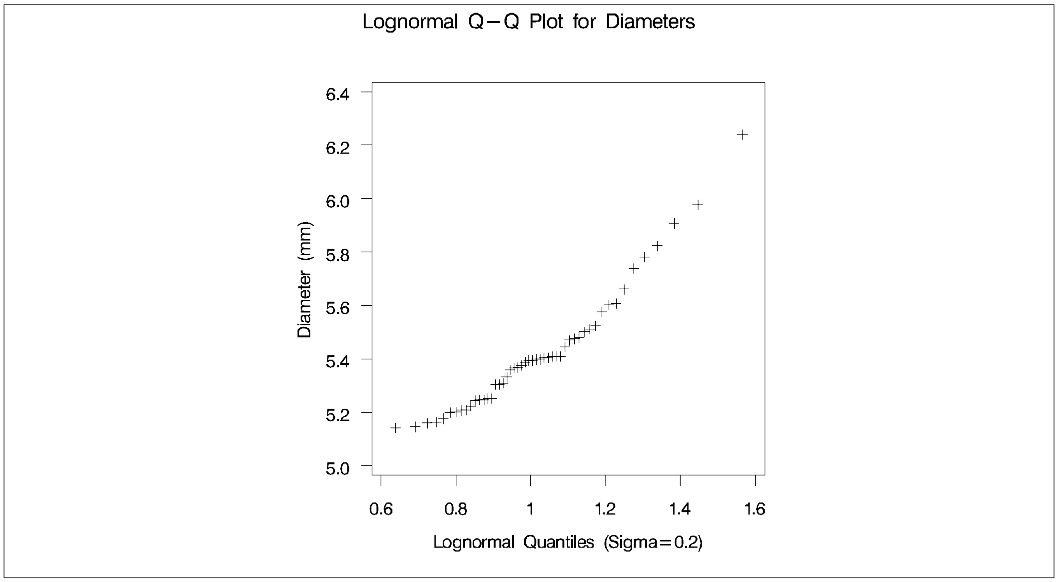
This example, which is a continuation of Example 3.30, demonstrates techniques for estimating the shape, location, and scale parameters, and the theoretical percentiles for a three-parameter lognormal distribution.

The three-parameter lognormal distribution depends on a threshold parameter θ, a scale parameter ζ, and a shape parameter σ. You can estimate σ from a series of lognormal Q-Q plots which use the SIGMA= secondary option to specify different values of σ; the estimate of σ is the value that linearizes the point pattern. You can then estimate the threshold and scale parameters from the intercept and slope of the point pattern. The following statements create the series of plots in Output 3.31.1, Output 3.31.2, and Output 3.31.3 for σ values of 0.2, 0.5, and 0.8, respectively:

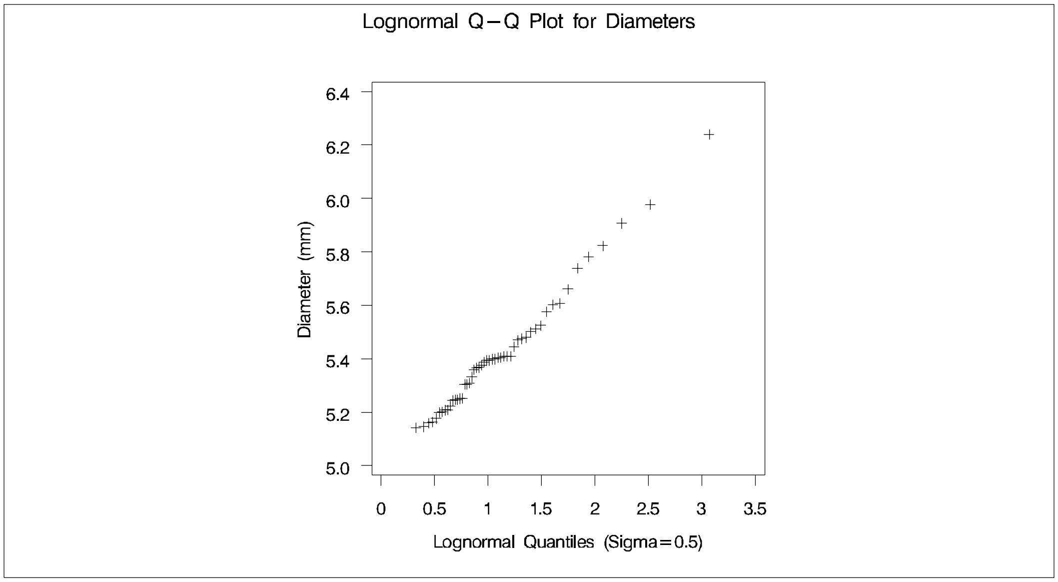
```
symbol v=plus;
title 'Lognormal Q-Q Plot for Diameters';
proc univariate data=Measures noprint;
   qqplot Diameter / lognormal(sigma=0.2 0.5 0.8 noprint)
                     square;
run;
```

Note: You must specify a value for the shape parameter σ for a lognormal Q-Q plot with the SIGMA= option or its alias, the SHAPE= option.

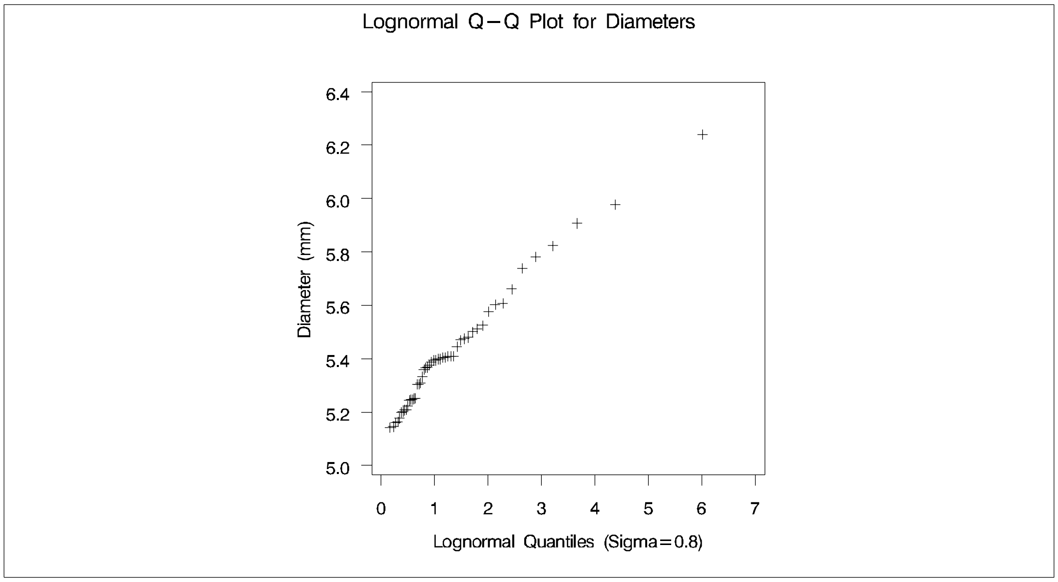
Output 3.31.1. Lognormal Quantile-Quantile Plot (σ =0.2)



Output 3.31.2. Lognormal Quantile-Quantile Plot (σ =0.5)



Output 3.31.3. Lognormal Quantile-Quantile Plot (σ =0.8)



The plot in Output 3.31.2 displays the most linear point pattern, indicating that the lognormal distribution with $\sigma = 0.5$ provides a reasonable fit for the data distribution.

Data with this particular lognormal distribution have the following density function:

$$p(x) = \begin{cases} \frac{\sqrt{2}}{\sqrt{\pi}(x-\theta)} \exp\left(-2(\log(x-\theta)-\zeta)^2\right) & \text{for } x > \theta \\ 0 & \text{for } x \leq \theta \end{cases}$$

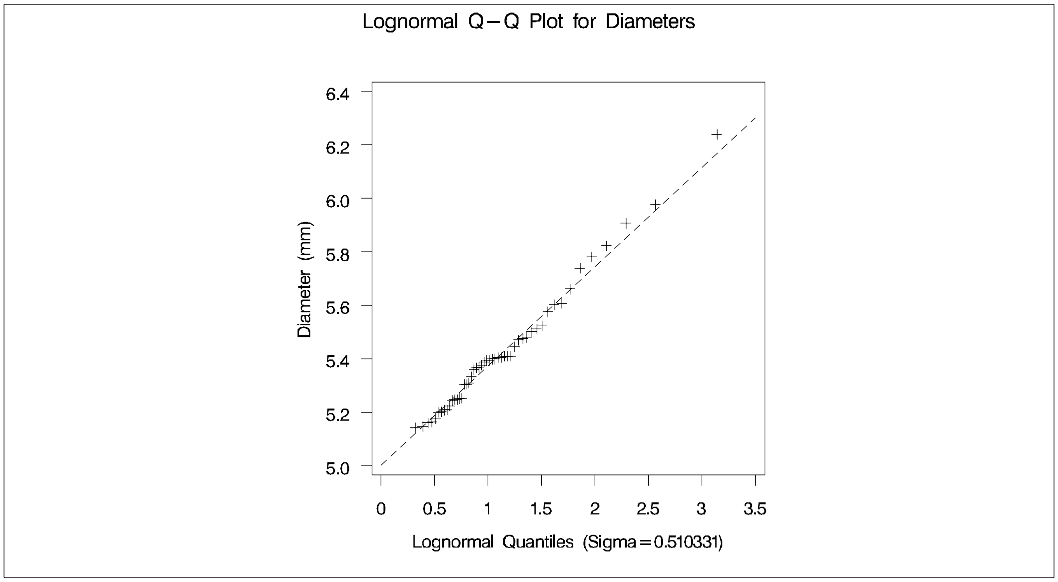
The points in the plot fall on or near the line with intercept θ and slope $\exp(\zeta)$. Based on Output 3.31.2, $\theta \approx 5$ and $\exp(\zeta) \approx \frac{1.2}{3} = 0.4$, giving $\zeta \approx \log(0.4) \approx -0.92$.

You can also request a reference line using the SIGMA=, THETA=, and ZETA= options together. The following statements produce the lognormal Q-Q plot in Output 3.31.4:

```
symbol v=plus;
title 'Lognormal Q-Q Plot for Diameters';
proc univariate data=Measures noprint;
   qqplot Diameter / lognormal(theta=5 zeta=est sigma=est
                               color=black l=2 noprint)
                     square;
run;
```

Output 3.31.1 through Output 3.31.3 show that the threshold parameter θ is not equal to zero. Specifying THETA=5 overrides the default value of zero. The SIGMA=EST and ZETA=EST secondary options request estimates for σ and $\exp \zeta$ using the sample mean and standard deviation.

Output 3.31.4. Lognormal Quantile-Quantile Plot (σ =est, ζ =est, θ =5)



From the plot in Output 3.31.2, σ can be estimated as 0.51, which is consistent with the estimate of 0.5 derived from the plot in Output 3.31.2. The next example illustrates how to estimate percentiles using lognormal Q-Q plots.

A sample program, uniex18.sas, for this example is available in the SAS Sample Library for Base SAS software.

Example 3.32. Estimating Percentiles from Lognormal Quantile Plots

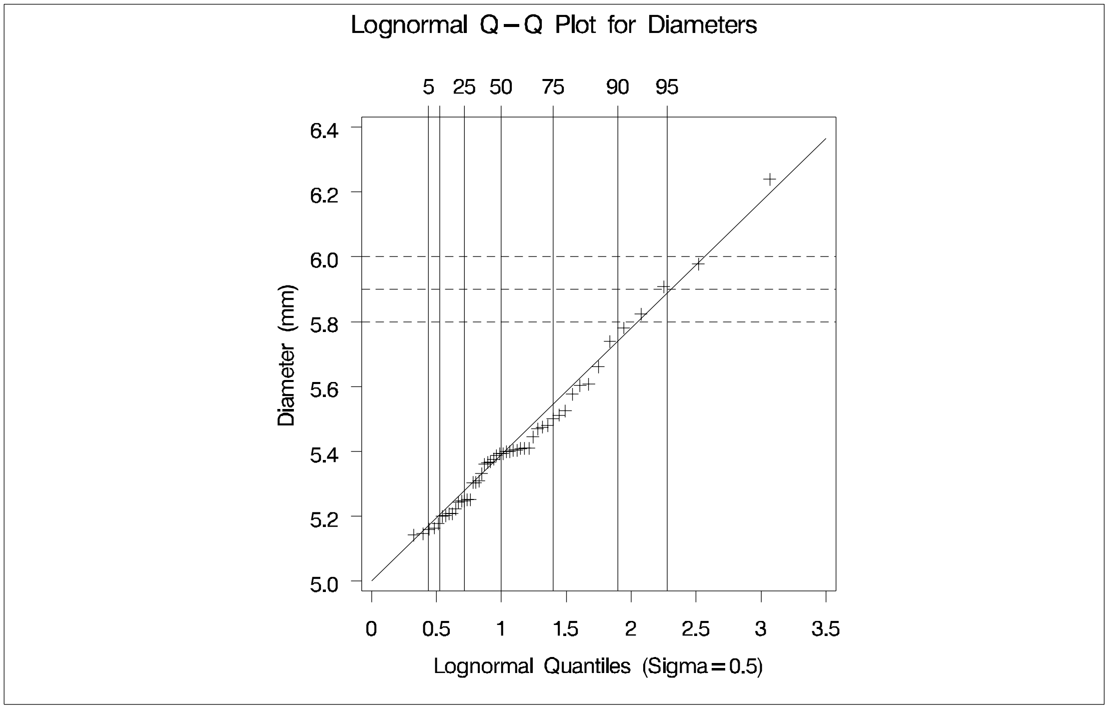
This example, which is a continuation of the previous example, shows how to use a Q-Q plot to estimate percentiles such as the 95th percentile of the lognormal distribution. A probability plot can also be used for this purpose, as illustrated in Example 3.26.

The point pattern in Output 3.31.4 has a slope of approximately 0.39 and an intercept of 5. The following statements reproduce this plot, adding a lognormal reference line with this slope and intercept:

```
symbol v=plus;
title 'Lognormal Q-Q Plot for Diameters';
proc univariate data=Measures noprint;
   qqplot Diameter / lognormal(sigma=0.5 theta=5 slope=0.39 noprint)
                     pctlaxis(grid)
                     vref  = 5.8 5.9 6.0
                     square;
run;
```

The result is shown in Output 3.32.1:

Output 3.32.1. Lognormal Q-Q Plot Identifying Percentiles



The PCTLAXIS option labels the major percentiles, and the GRID option draws percentile axis reference lines. The 95th percentile is 5.9, since the intersection of the distribution reference line and the 95th reference line occurs at this value on the vertical axis.

Alternatively, you can compute this percentile from the estimated lognormal parameters. The αth percentile of the lognormal distribution is

$$P_\alpha = \exp(\sigma\Phi^{-1}(\alpha) + \zeta) + \theta$$

where $\Phi^{-1}(\cdot)$ is the inverse cumulative standard normal distribution. Consequently,

$$\hat{P}_{0.95} \quad = \quad \exp\left(\tfrac{1}{2}\Phi^{-1}(0.95) + \log(0.39)\right) + 5 \quad = \quad 5.89$$

A sample program, uniex18.sas, for this example is available in the SAS Sample Library for Base SAS software.

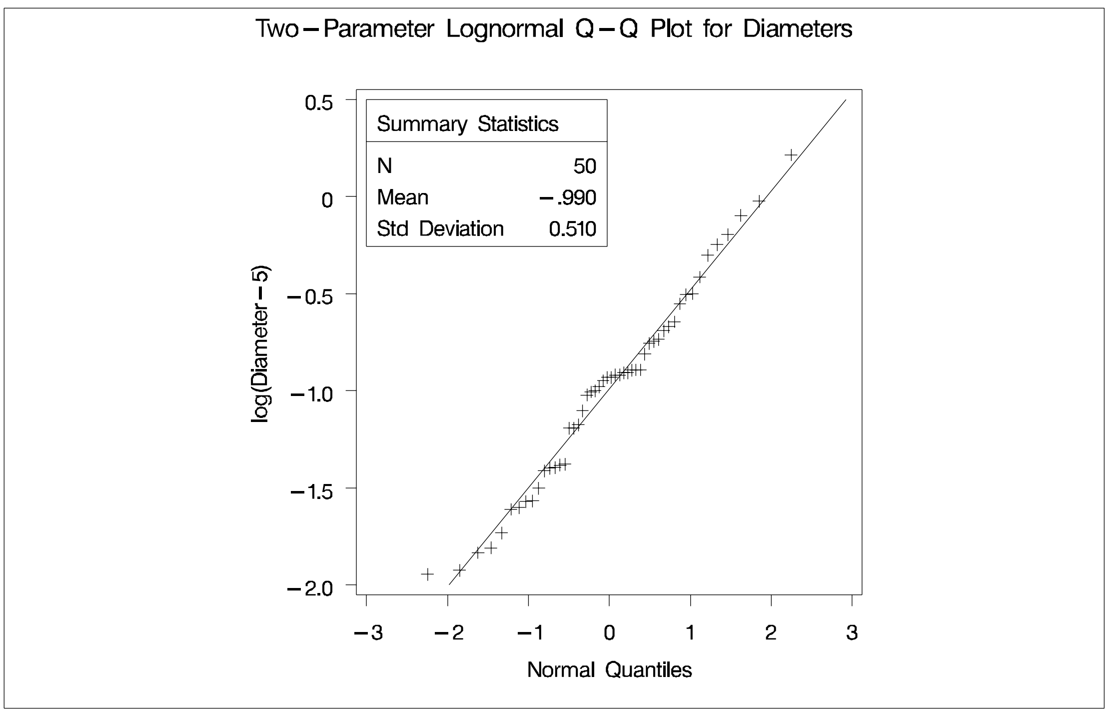
Example 3.33. Estimating Parameters from Lognormal Quantile Plots

This example, which is a continuation of Example 3.31, demonstrates techniques for estimating the shape, location, and scale parameters, and the theoretical percentiles for a two-parameter lognormal distribution.

If the threshold parameter is known, you can construct a two-parameter lognormal Q-Q plot by subtracting the threshold from the data values and making a normal Q-Q plot of the log-transformed differences, as illustrated in the following statements:

```
data ModifiedMeasures;
   set Measures;
   LogDiameter = log(Diameter-5);
   label LogDiameter = 'log(Diameter-5)';
run;

symbol v=plus;
title 'Two-Parameter Lognormal Q-Q Plot for Diameters';
proc univariate data=ModifiedMeasures noprint;
   qqplot LogDiameter / normal(mu=est sigma=est noprint)
                        square
                        vaxis=axis1;
   inset n mean (5.3) std (5.3)
           / pos = nw header = 'Summary Statistics';
   axis1 label=(a=90 r=0);
run;
```

Output 3.33.1. Two-Parameter Lognormal Q-Q Plot for Diameters



Because the point pattern in Output 3.33.1 is linear, you can estimate the lognormal parameters ζ and σ as the normal plot estimates of μ and σ, which are -0.99 and 0.51. These values correspond to the previous estimates of -0.92 for ζ and 0.5 for σ from Example 3.31. A sample program, uniex18.sas, for this example is available in the SAS Sample Library for Base SAS software.

Example 3.34. Comparing Weibull Quantile Plots

This example compares the use of three-parameter and two-parameter Weibull Q-Q plots for the failure times in months for 48 integrated circuits. The times are assumed to follow a Weibull distribution. The following statements save the failure times as the values of the variable Time in the data set Failures:

```
data Failures;
   input Time @@;
   label Time = 'Time in Months';
   datalines;
29.42 32.14 30.58 27.50 26.08 29.06 25.10 31.34
29.14 33.96 30.64 27.32 29.86 26.28 29.68 33.76
29.32 30.82 27.26 27.92 30.92 24.64 32.90 35.46
30.28 28.36 25.86 31.36 25.26 36.32 28.58 28.88
26.72 27.42 29.02 27.54 31.60 33.46 26.78 27.82
29.18 27.94 27.66 26.42 31.00 26.64 31.44 32.52
;
run;
```

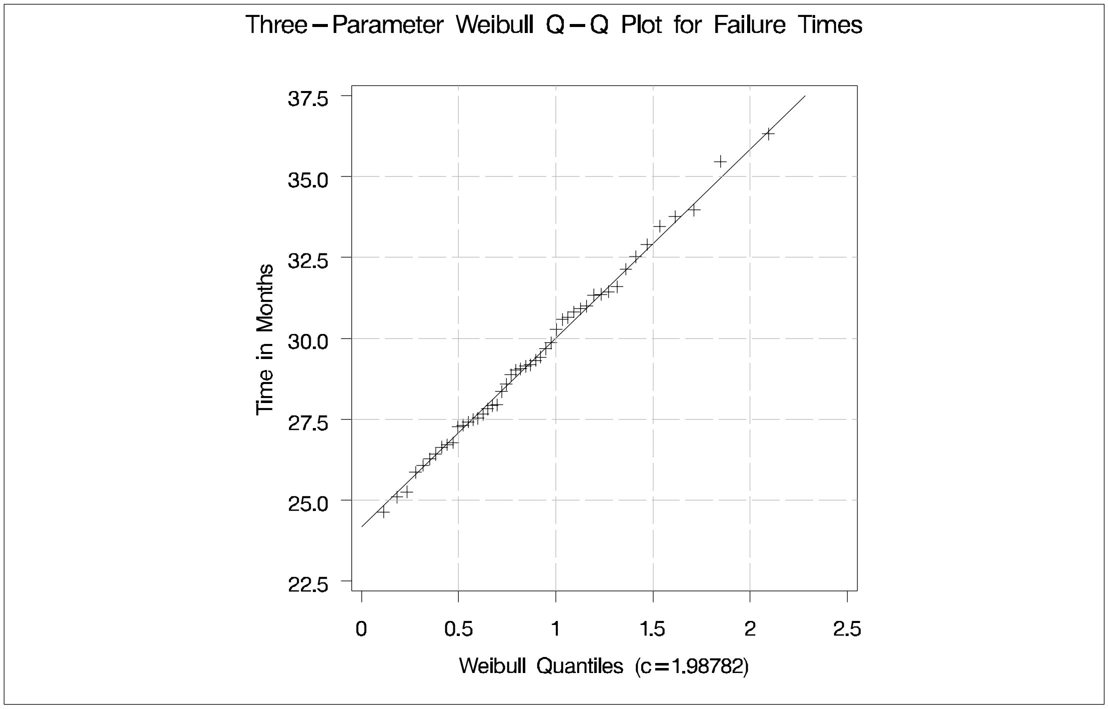
If no assumption is made about the parameters of this distribution, you can use the WEIBULL option to request a three-parameter Weibull plot. As in the previous example, you can visually estimate the shape parameter c by requesting plots for different values of c and choosing the value of c that linearizes the point pattern. Alternatively, you can request a maximum likelihood estimate for c, as illustrated in the following statements:

```
symbol v=plus;
title 'Three-Parameter Weibull Q-Q Plot for Failure Times';
proc univariate data=Failures noprint;
   qqplot Time / weibull(c=est theta=est sigma=est noprint)
                 square
                 href=0.5 1 1.5 2
                 vref=25 27.5 30 32.5 35
                 lhref=4 lvref=4
                 chref=tan cvref=tan;
run;
```

Note: When using the WEIBULL option, you must either specify a list of values for the Weibull shape parameter c with the C= option, or you must specify C=EST.

Output 3.34.1 displays the plot for the estimated value $\hat{c} = 1.99$. The reference line corresponds to the estimated values for the threshold and scale parameters of $\hat{\theta}_0 = 24.19$ and $\hat{\sigma}_0 = 5.83$, respectively.

Output 3.34.1. Three-Parameter Weibull Q-Q Plot

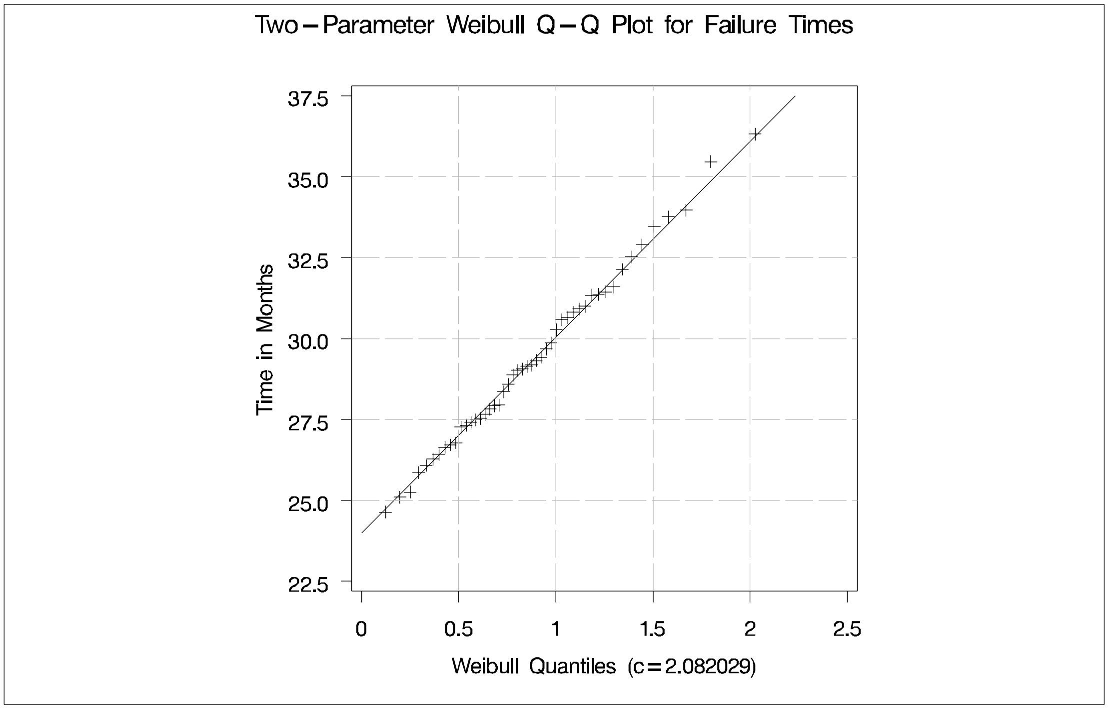


Now, suppose it is known that the circuit lifetime is at least 24 months. The following statements use the known threshold value $\theta_0 = 24$ to produce the two-parameter Weibull Q-Q plot shown in Output 3.31.4:

```
symbol v=plus;
title 'Two-Parameter Weibull Q-Q Plot for Failure Times';
proc univariate data=Failures noprint;
   qqplot Time / weibull(theta=24 c=est sigma=est noprint)
                 square
                 vref= 25 to 35 by 2.5
                 href= 0.5 to 2.0 by 0.5
                 lhref=4 lvref=4
                 chref=tan cvref=tan;
run;
```

The reference line is based on maximum likelihood estimates $\hat{c} = 2.08$ and $\hat{\sigma} = 6.05$.

Output 3.34.2. Two-Parameter Weibull Q-Q Plot for $\theta_0 = 24$



A sample program, uniex19.sas, for this example is available in the SAS Sample Library for Base SAS software.

References

Blom, G. (1958), *Statistical Estimates and Transformed Beta-Variables*, New York: John Wiley & Sons, Inc.

Chambers, J. M., Cleveland, W. S., Kleiner, B., and Tukey, P. A. (1983), *Graphical Methods for Data Analysis*, Belmont, Calif.: Wadsworth International Group.

Cohen, A. C., Jr. (1951), "Estimating Parameters of Logarithmic-Normal Distributions by Maximum Likelihood," *Journal of the American Statistical Association*, 46, 206–212.

Conover, W. J. (1999), *Practical Nonparametric Statistics*, Third Edition, New York: John Wiley & Sons, Inc.

Croux, C. and Rousseeuw, P. J. (1992), "Time-Efficient Algorithms for Two Highly Robust Estimators of Scale," *Computational Statistics*, Vol. 1, 411–428.

D'Agostino, R. B. and Stephens, M. (1986), *Goodness-of-Fit Techniques*, New York: Marcel Dekker, Inc.

Dixon, W. J. and Tukey, J. W. (1968), "Approximate Behavior of the Distribution of Winsorized t (Trimming/Winsorization 2)," *Technometrics*, 10, 83–98.

Elandt, R. C. (1961), “The Folded Normal Distribution: Two Methods of Estimating Parameters from Moments,” *Technometrics*, 3, 551–562.

Fisher, R. A. (1973), *Statistical Methods and Scientific Inference*, New York: Hafner Press.

Fowlkes, E. B. (1987), *A Folio of Distributions: A Collection of Theoretical Quantile-Quantile Plots*, New York: Marcel Dekker, Inc.

Hahn, G. J. and Meeker, W. Q. (1991), *Statistical Intervals: A Guide for Practitioners*, New York: John Wiley & Sons, Inc.

Hampel, F. R. (1974), “The Influence Curve and Its Role in Robust Estimation,” *Journal of the American Statistical Association*, 69, 383–393.

Iman, R. L. (1974), “Use of a t-statistic as an Approximation to the Exact Distribution of the Wilcoxon Signed Ranks Test Statistic,” *Communications in Statistics*, 3, 795–806.

Johnson, N. L., Kotz, S., and Balakrishnan, N. (1994), *Continuous Univariate Distributions*, Vol. 1, New York: John Wiley & Sons, Inc.

Johnson, N. L., Kotz, S., and Balakrishnan, N. (1995), *Continuous Univariate Distributions*, Vol. 2, New York: John Wiley & Sons, Inc.

Lehmann, E. L. (1998), *Nonparametrics: Statistical Methods Based on Ranks*, New Jersey: Prentice Hall.

Odeh, R. E. and Owen, D. B. (1980), *Tables for Normal Tolerance Limits, Sampling Plans, and Screening*, New York: Marcel Dekker, Inc.

Owen, D. B. and Hua, T. A. (1977), “Tables of Confidence Limits on the Tail Area of the Normal Distribution,” *Communication and Statistics, Part B–Simulation and Computation*, 6, 285–311.

Rousseeuw, P. J. and Croux, C. (1993), “Alternatives to the Median Absolute Deviation,” *Journal of the American Statistical Association*. 88, 1273–1283.

Royston, J. P. (1992), “Approximating the Shapiro-Wilk W-Test for Non-normality,” *Statistics and Computing*, 2, 117–119.

Shapiro, S. S. and Wilk, M. B. (1965), “An Analysis of Variance Test for Normality (complete samples),” *Biometrika*, 52, 591–611.

Silverman, B. W. (1986), *Density Estimation for Statistics and Data Analysis*, New York: Chapman and Hall.

Terrell, G. R. and Scott, D. W. (1985), “Oversmoothed Nonparametric Density Estimates,” *Journal of the American Statistical Association*, 80, 209–214.

Tukey, J. W. (1977), *Exploratory Data Analysis*, Reading, Massachusetts: Addison-Wesley.

Tukey, J. W. and McLaughlin, D. H. (1963), “Less Vulnerable Confidence and Significance Procedures for Location Based on a Single Sample: Trimming/Winsorization 1,” *Sankhya A*, 25, 331–352.

Subject Index

Q

R

Syntax Index

O

P

V

W

Z