THE POWER TO KNOW.

Base SAS® 9.2 Procedures Guide

SAS® Documentation

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2009. *Base SAS® 9.2 Procedures Guide*. Cary, NC: SAS Institute Inc.

Base SAS® 9.2 Procedures Guide

ISBN 978–1-59994-714-3

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

1st electronic printing, February 2009

2nd electronic printing, September 2009

3rd electronic printing, May 2010

1st printing, June 2009

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

Contents

What's New ***xiii***
Overview **xiii**
New Base SAS Procedures **xiv**
Enhanced Base SAS Procedures **xv**
Documentation Enhancements **xxiv**

PART 1 Concepts 1

Chapter 1 △ Choosing the Right Procedure 3
Functional Categories of Base SAS Procedures **3**
Report-Writing Procedures **5**
Statistical Procedures **6**
Utility Procedures **8**
Brief Descriptions of Base SAS Procedures **10**

Chapter 2 △ Fundamental Concepts for Using Base SAS Procedures 17
Language Concepts **17**
Procedure Concepts **20**
Output Delivery System **33**

Chapter 3 △ Statements with the Same Function in Multiple Procedures 35
Overview **35**
Statements **36**

Chapter 4 △ In-Database Processing of Base Procedures 49
Base Procedures That Are Enhanced for In-Database Processing **49**

PART 2 Procedures 51

Chapter 5 △ The APPEND Procedure 55
Overview: APPEND Procedure **55**
Syntax: APPEND Procedure **55**

Chapter 6 △ The CALENDAR Procedure 57
Overview: CALENDAR Procedure **59**
Syntax: CALENDAR Procedure **64**
Concepts: CALENDAR Procedure **82**
Results: CALENDAR Procedure **92**
Examples: CALENDAR Procedure **93**

Chapter 7 △ The CALLRFC Procedure 129
Information about the CALLRFC Procedure **129**

Chapter 8 △ The CATALOG Procedure 131
Overview: CATALOG Procedure **131**
Syntax: CATALOG Procedure **132**
Concepts: CATALOG Procedure **142**
Results: CATALOG Procedure **146**
Examples: CATALOG Procedure **147**

Chapter 9 △ The CHART Procedure 155
Overview: CHART Procedure **155**
Syntax: CHART Procedure **160**
Concepts: CHART Procedure **173**
Results: CHART Procedure **174**
Examples: CHART Procedure **175**
References **189**

Chapter 10 △ The CIMPORT Procedure 191
Overview: CIMPORT Procedure **191**
Syntax: CIMPORT Procedure **192**
CIMPORT Problems: Importing Transport Files **198**
Examples: CIMPORT Procedure **203**

Chapter 11 △ The COMPARE Procedure 207
Overview: COMPARE Procedure **208**
Syntax: COMPARE Procedure **211**
Concepts: COMPARE Procedure **222**
Results: COMPARE Procedure **226**
Examples: COMPARE Procedure **239**

Chapter 12 △ The CONTENTS Procedure 259
Overview: CONTENTS Procedure **259**
Syntax: CONTENTS Procedure **259**

Chapter 13 △ The COPY Procedure 261
Overview: COPY Procedure **261**
Syntax: COPY Procedure **261**
Concepts: COPY Procedure **262**
Example: COPY Procedure **263**

Chapter 14 △ The CORR Procedure 267
Information about the CORR Procedure **267**

Chapter 15 △ The CPORT Procedure 269
Overview: CPORT Procedure **269**
Syntax: CPORT Procedure **270**
READ= Data Set Option in the PROC CPORT Statement **278**
Results: CPORT Procedure **279**
Examples: CPORT Procedure **279**

Chapter 16 △ The CV2VIEW Procedure 285
Information about the CV2VIEW Procedure **285**

Chapter 17 △ The DATASETS Procedure 287
Overview: DATASETS Procedure **288**
Syntax: DATASETS Procedure **291**
Concepts: DATASETS Procedure **353**
Results: DATASETS Procedure **359**
Examples: DATASETS Procedure **372**

Chapter 18 △ The DBCSTAB Procedure 399
Information about the DBCSTAB Procedure **399**

Chapter 19 △ The DISPLAY Procedure 401
Overview: DISPLAY Procedure **401**
Syntax: DISPLAY Procedure **401**
Example: DISPLAY Procedure **402**

Chapter 20 △ The DOCUMENT Procedure 405
Information about the DOCUMENT Procedure **405**

Chapter 21 △ The EXPLODE Procedure 407
Information about the EXPLODE Procedure **407**

Chapter 22 △ The EXPORT Procedure 409
Overview: EXPORT Procedure **409**
Syntax: EXPORT Procedure **409**
Examples: EXPORT Procedure **412**

Chapter 23 △ The FCMP Procedure 417
Overview: FCMP Procedure **420**
Syntax: FCMP Procedure **420**
Concepts: FCMP Procedure **431**
PROC FCMP and DATA Step Differences **435**
Working with Arrays **438**
Reading Arrays and Writing Arrays to a Data Set **439**
Using Macros with PROC FCMP Routines **442**
Variable Scope in PROC FCMP Routines **442**
Recursion **443**
Directory Transversal **445**
Identifying the Location of Compiled Functions and Subroutines: The CMPLIB= System Option **448**
Special Functions and CALL Routines: Overview **451**
Special Functions and CALL Routines: Matrix CALL Routines **451**
Special Functions and CALL Routines: C Helper Functions and CALL Routines **463**
Special Functions and CALL Routines: Other Functions **467**
Functions for Calling SAS Code from Within Functions **472**
The FCmp Function Editor **477**

Examples: FCMP Procedure **488**

Chapter 24 △ The FONTREG Procedure 497
Overview: FONTREG Procedure **497**
Syntax: FONTREG Procedure **498**
Concepts: FONTREG Procedure **504**
Examples: FONTREG Procedure **506**

Chapter 25 △ The FORMAT Procedure 511
Overview: FORMAT Procedure **512**
Syntax: FORMAT Procedure **513**
Informat and Format Options **534**
Specifying Values or Ranges **536**
Concepts: FORMAT Procedure **537**
Results: FORMAT Procedure **541**
Examples: FORMAT Procedure **546**

Chapter 26 △ The FORMS Procedure 573
Information about the FORMS Procedure **573**

Chapter 27 △ The FREQ Procedure 575
Information about the FREQ Procedure **575**

Chapter 28 △ The FSLIST Procedure 577
Overview: FSLIST Procedure **577**
Syntax: FSLIST Procedure **577**
Using the FSLIST Window **582**

Chapter 29 △ The HTTP Procedure 589
Overview: HTTP Procedure **589**
Syntax: HTTP Procedure **589**
Using Hypertext Transfer Protocol Secure (HTTPS) **591**
Examples: HTTP Procedure **592**

Chapter 30 △ The IMPORT Procedure 595
Overview: IMPORT Procedure **595**
Syntax: IMPORT Procedure **596**
Examples: IMPORT Procedure **599**

Chapter 31 △ The INFOMAPS Procedure 605
Information about the INFOMAPS Procedure **605**

Chapter 32 △ The JAVAINFO Procedure 607
Overview: JAVAINFO Procedure **607**
Syntax: JAVAINFO Procedure **607**

Chapter 33 △ The MEANS Procedure 609
Overview: MEANS Procedure **610**
Syntax: MEANS Procedure **612**

Concepts: MEANS Procedure **637**
In-Database Processing for PROC MEANS **640**
Statistical Computations: MEANS Procedure **641**
Results: MEANS Procedure **644**
Examples: MEANS Procedure **646**
References **675**

Chapter 34 △ The METADATA Procedure 677
Information about the METADATA Procedure **677**

Chapter 35 △ The METALIB Procedure 679
Information about the METALIB Procedure **679**

Chapter 36 △ The METAOPERATE Procedure 681
Information about the METAOPERATE Procedure **681**

Chapter 37 △ The MIGRATE Procedure 683
Overview: MIGRATE Procedure **683**
Syntax: MIGRATE Procedure **685**
Concepts: MIGRATE Procedure **687**
Migrating a Library with Validation Tools **693**
Using the SLIBREF= Option **693**
Examples **695**

Chapter 38 △ The OPTIONS Procedure 701
Overview: OPTIONS Procedure **701**
Syntax: OPTIONS Procedure **707**
Results: OPTIONS Procedure **710**
Examples: OPTIONS Procedure **710**

Chapter 39 △ The OPTLOAD Procedure 715
Overview: OPTLOAD Procedure **715**
Syntax: OPTLOAD Procedure **715**

Chapter 40 △ The OPTSAVE Procedure 717
Overview: OPTSAVE Procedure **717**
Syntax: OPTSAVE Procedure **717**

Chapter 41 △ The PLOT Procedure 719
Overview: PLOT Procedure **720**
Syntax: PLOT Procedure **722**
Concepts: PLOT Procedure **738**
Results: PLOT Procedure **743**
Examples: PLOT Procedure **744**

Chapter 42 △ The PMENU Procedure 777
Overview: PMENU Procedure **777**
Syntax: PMENU Procedure **778**

Concepts: PMENU Procedure **792**
Examples: PMENU Procedure **794**

Chapter 43 △ The PRINT Procedure 815
Overview: PRINT Procedure **815**
Syntax: PRINT Procedure **818**
Results: Print Procedure **832**
Examples: PRINT Procedure **835**

Chapter 44 △ The PRINTTO Procedure 887
Overview: PRINTTO Procedure **887**
Syntax: PRINTTO Procedure **888**
Concepts: PRINTTO Procedure **891**
Examples: PRINTTO Procedure **892**

Chapter 45 △ The PROTO Procedure 903
Overview: PROTO Procedure **903**
Syntax: PROTO Procedure **904**
Concepts: PROTO Procedure **906**
C Helper Functions and CALL Routines **916**
Results: PROTO Procedure **918**
Examples: PROTO Procedure **919**

Chapter 46 △ The PRTDEF Procedure 921
Overview: PRTDEF Procedure **921**
Syntax: PRTDEF Procedure **921**
Input Data Set: PRTDEF Procedure **923**
Examples: PRTDEF Procedure **928**

Chapter 47 △ The PRTEXP Procedure 933
Overview: PRTEXP Procedure **933**
Syntax: PRTEXP Procedure **933**
Concepts: PRTEXP Procedure **935**
Examples: PRTEXP Procedure **935**

Chapter 48 △ The PWENCODE Procedure 937
Overview: PWENCODE Procedure **937**
Syntax: PWENCODE Procedure **937**
Concepts: PWENCODE Procedure **938**
Examples: PWENCODE Procedure **939**

Chapter 49 △ The RANK Procedure 943
Overview: RANK Procedure **943**
Syntax: RANK Procedure **945**
Concepts: RANK Procedure **951**
In-Database Processing for PROC RANK **953**
Results: RANK Procedure **954**

Examples: RANK Procedure **955**
References **961**

Chapter 50 △ The REGISTRY Procedure 963
Overview: REGISTRY Procedure **963**
Syntax: REGISTRY Procedure **963**
Creating Registry Files with the REGISTRY Procedure **968**
Examples: REGISTRY Procedure **971**

Chapter 51 △ The REPORT Procedure 979
Overview: REPORT Procedure **981**
Concepts: REPORT Procedure **986**
In-Database Processing for PROC REPORT **1003**
Syntax: REPORT Procedure **1004**
REPORT Procedure Windows **1052**
How PROC REPORT Builds a Report **1075**
Examples: REPORT Procedure **1087**

Chapter 52 △ The SCAPROC Procedure 1143
Overview: SCAPROC Procedure **1143**
Syntax: SCAPROC Procedure **1144**
Results: SCAPROC Procedure **1145**
Examples: SCAPROC Procedure **1148**

Chapter 53 △ The SOAP Procedure 1153
Overview: SOAP Procedure **1153**
Syntax: SOAP Procedure **1154**
Concepts: SOAP Procedure **1157**
WS-Security: Client Configuration **1157**
Using PROC SOAP with Secure Socket Layer (SSL) **1158**
Methods of Calling SAS Web Services **1159**
Examples: SOAP Procedure **1160**

Chapter 54 △ The SORT Procedure 1165
Overview: SORT Procedure **1165**
Syntax: SORT Procedure **1167**
Concepts: SORT Procedure **1181**
In-Database Processing: PROC SORT **1184**
Integrity Constraints: SORT Procedure **1185**
Results: SORT Procedure **1186**
Examples: SORT Procedure **1187**

Chapter 55 △ The SQL Procedure 1197
Overview: SQL Procedure **1199**
Syntax: SQL Procedure **1201**
SQL Procedure Component Dictionary **1247**
PROC SQL and the ANSI Standard **1293**

Examples: SQL Procedure **1296**

Chapter 56 △ The STANDARD Procedure 1335
Overview: STANDARD Procedure **1335**
Syntax: STANDARD Procedure **1337**
Results: STANDARD Procedure **1343**
Statistical Computations: STANDARD Procedure **1343**
Examples: STANDARD Procedure **1344**

Chapter 57 △ The SUMMARY Procedure 1351
Overview: SUMMARY Procedure **1351**
Syntax: SUMMARY Procedure **1351**

Chapter 58 △ The TABULATE Procedure 1355
Overview: TABULATE Procedure **1356**
Terminology: TABULATE Procedure **1359**
Syntax: TABULATE Procedure **1362**
Concepts: TABULATE Procedure **1391**
In-Database Processing for PROC TABULATE **1400**
Results: TABULATE Procedure **1401**
Examples: TABULATE Procedure **1413**
References **1474**

Chapter 59 △ The TEMPLATE Procedure 1475
Information about the TEMPLATE Procedure **1475**

Chapter 60 △ The TIMEPLOT Procedure 1477
Overview: TIMEPLOT Procedure **1477**
Syntax: TIMEPLOT Procedure **1479**
Results: TIMEPLOT Procedure **1487**
Examples: TIMEPLOT Procedure **1489**

Chapter 61 △ The TRANSPOSE Procedure 1501
Overview: TRANSPOSE Procedure **1501**
Syntax: TRANSPOSE Procedure **1504**
Results: TRANSPOSE Procedure **1510**
Examples: TRANSPOSE Procedure **1512**

Chapter 62 △ The TRANTAB Procedure 1525
Information about the TRANTAB Procedure **1525**

Chapter 63 △ The UNIVARIATE Procedure 1527
Information about the UNIVARIATE Procedure **1527**

Chapter 64 △ The XSL Procedure (Preproduction) 1529
Overview: XSL Procedure **1529**
Syntax: XSL Procedure **1530**
Examples: XSL Procedure **1531**

PART 3 Appendixes 1533

Appendix 1 △ SAS Elementary Statistics Procedures 1535
Overview **1535**
Keywords and Formulas **1536**
Statistical Background **1544**
References **1569**

Appendix 2 △ Operating Environment-Specific Procedures 1571
Descriptions of Operating Environment-Specific Procedures **1571**

Appendix 3 △ Raw Data and DATA Steps 1573
Overview **1574**
CENSUS **1574**
CHARITY **1575**
CONTROL Library **1577**
CUSTOMER_RESPONSE **1602**
DJIA **1604**
EDUCATION **1605**
EMPDATA **1606**
ENERGY **1608**
EXP Library **1609**
EXPREV **1610**
GROC **1611**
MATCH_11 **1612**
PROCLIB.DELAY **1613**
PROCLIB.EMP95 **1614**
PROCLIB.EMP96 **1615**
PROCLIB.INTERNAT **1616**
PROCLIB.LAKES **1616**
PROCLIB.MARCH **1617**
PROCLIB.PAYLIST2 **1618**
PROCLIB.PAYROLL **1618**
PROCLIB.PAYROLL2 **1621**
PROCLIB.SCHEDULE **1622**
PROCLIB.STAFF **1625**
PROCLIB.SUPERV **1628**
RADIO **1629**
SALES **1641**

Appendix 4 △ ICU License 1643
ICU License - ICU 1.8.1 and later **1643**

Appendix 5 △ Recommended Reading 1645
Recommended Reading **1645**

Index 1647

What's New

Overview

The following Base SAS procedures are new:

- CALLRFC
- FCMP
- HTTP
- JAVAINFO
- PROTO
- SCAPROC
- SOAP
- XSL

The following Base SAS procedures have been enhanced:

- APPEND
- CIMPORT
- CONTENTS
- COPY
- CPORT
- CORR
- DATASETS
- FREQ
- MEANS
- MIGRATE
- OPTIONS
- PRINT
- PWENCODE
- RANK
- REPORT
- SORT

- SQL
- SUMMARY
- TABULATE
- UNIVARIATE

New Base SAS Procedures

The CALLRFC Procedure

The CALLRFC procedure enables you to invoke Remote Function Call (RFC) or RFC-compatible functions on an SAP System from a SAS program. You must license and configure SAS/ACCESS Interface to R/3 to use the CALLRFC procedure.

The FCMP Procedure

The FCMP procedure is new for 9.2. The SAS Function Compiler Procedure (FCMP) enables you to create, test, and store SAS functions and subroutines before you use them in other SAS procedures. PROC FCMP accepts slight variations of DATA step statements, and most features of the SAS programming language can be used in functions and subroutines that are processed by PROC FCMP.

The JAVAINFO Procedure

The JAVAINFO procedure conveys diagnostic information to the user about the Java environment that SAS is using. The diagnostic information can be used to confirm that the SAS Java environment has been configured correctly, and can be helpful when reporting problems to SAS technical support. Also, PROC JAVAINFO is often used to verify that the SAS Java environment is working correctly because PROC JAVAINFO uses Java to report its diagnostics.

The PROTO Procedure

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

The SCAPROC Procedure

The SCAPROC procedure enables you to specify a filename or fileref that will contain the output of the SAS Code Analyzer and to write the output to the file. The SAS Code Analyzer captures information about the job step, input and output information such as file dependencies, and information about macro symbol usage from a running SAS job. The SCAPROC procedure also can generate a grid-enabled job that can simultaneously run independent pieces of a SAS job.

The CONCATMEM comment has been added to the SCAPROC procedure output for the third maintenance release for SAS 9.2. The CONCATMEM comment specifies the name of a concatenated library that contains a specified libref.

The SOAP Procedure

The SOAP procedure reads XML input from a file that has a fileref and writes XML output to another file that has a fileref. The envelope and headings are part of the content of the fileref.

The HTTP Procedure

The HTTP procedure invokes a Web service that issues requests.

The XSL Procedure

The XSL procedure is new for the third maintenance release for SAS 9.2. The XSL procedure transforms an XML document into another format, such as HTML, text, or another XML document type. The procedure reads an input XML document, transforms it by using an XSL style sheet, and then writes an output file.

Enhanced Base SAS Procedures

The APPEND Procedure

The NOWARN option has been added to the APPEND procedure. The NOWARN option suppresses the warning message when it is used with the FORCE option to concatenate two data sets with different variables.

The CIMPORT Procedure

The following enhancement has been made to the CIMPORT procedure:

- ISFILEUTF8= is a new option that specifies whether the encoding of the transport file is UTF-8. This feature is useful when you import a transport file whose UTF-8 encoding identity is known to you but is not stored in the transport file. SAS releases before SAS 9.2 do not store any encodings in the transport file.
- New warning and error messages are available to alert you to transport problems and recovery actions.

The CONTENTS Procedure

The WHERE option of the CONTENTS procedure has been restricted. You cannot use the WHERE option to affect the output because PROC CONTENTS does not process any observations.

The COPY Procedure

The PROC COPY option of the COPY procedure ignores concatenations with catalogs. Use PROC CATALOG COPY to copy concatenated catalogs.

The CPORT Procedure

The documentation about the READ= data set option (used in the DATA statement of PROC CPORT) was enhanced to explain when a read-only password might be required. You can create a transport file for a read-only data set only when you also specify the data set's password using the READ= option in PROC CPORT. Clear-text and encoded passwords are supported.

The CORR Procedure

The new ID statement for the CORR procedure specifies one or more additional tip variables to identify observations in scatter plots and scatter plot matrices.

The DATASETS Procedure

The following options are new or enhanced in the DATASETS procedure:

- The new REBUILD option specifies whether to correct or delete disabled indexes and integrity constraints. When a data set is damaged in some way and the DLDMGACTION=NOINDEX data set or system option is used, the data set is repaired, the indexes and integrity constraint are disabled, and the index file is deleted. The data set is then limited to INPUT mode only until the REBUILD option is executed. This option enables you to continue with production without waiting for the indexes to be repaired, which can take a long time on large data sets.
- Here is a list of enhancements for the COPY statement:
  - The COPY statement with the NOCLONE option specified supports the OUTREP= and ENCODING= LIBNAME options for SQL views, DATA step views, and some SAS/ACCESS views (Oracle and Sybase).
  - You can use the COPY statement, along with the XPORT engine or a REMOTE engine, to transport SAS data sets between hosts.
- Here is a list of enhancements for the CONTENTS statement:
  - When using the OUT2 option, indexes and integrity constraints are labeled if disabled.

The FCMP Procedure

In the third maintenance release for SAS 9.2, the following statements have been added to the FCMP procedure:

LISTFUNC | LISTSUBR	causes the source code for a function to be written to the SAS listing.
DELETEFUNC | DELETESUBR	causes a specified function to be deleted from the library that is specified in the OUTLIB option.

In the third maintenance release for SAS 9.2, the following option has been added:

LISTFUNCS enables you to list the prototypes for all visible FCMP procedure functions in the SAS listing.

The FREQ Procedure

The FREQ procedure can now produce frequency plots, cumulative frequency plots, deviation plots, odds ratio plots, and kappa plots by using ODS Graphics. The crosstabulation table now has an ODS template that you can customize using the TEMPLATE procedure. Equivalence and noninferiority tests are now available for the binomial proportion and the proportion difference. New confidence limits for the binomial proportion include Agresti-Coull, Jeffreys, and Wilson (score) confidence limits. The RISKDIFF option in the EXACT statement provides unconditional exact confidence limits for the proportion (risk) difference. The EQOR option in the EXACT statement provides Zelen's exact test for equal odds ratios.

In the third maintenance release for SAS 9.2, the FREQ procedure has been enhanced to run inside the Teradata Enterprise Data Warehouse (EDW), DB2 under UNIX, and Oracle. Using conventional processing, a SAS procedure, by means of the SAS/ACCESS engine, receives all the rows of the table from the database. All processing is done by the procedure. Large tables mean that a significant amount of data must be transferred. Using the new in-database technology, the procedures that are enabled for processing inside the database generate more sophisticated queries that allow the aggregations and analytics to be run inside the database. For most in-database procedures, a much smaller result set is returned for the remaining analysis that is required to produce the final output. As a result of using the in-database procedures, more work is done inside the database and less data movement can occur. Using in-database procedures can result in significant performance improvements.

The MEANS Procedure

The following enhancements have been made to the MEANS procedure:

- The PRT statistic is now an alias for the PROBT statistic.
- The MODE statistic can now be used with PROC MEANS.

In the third maintenance release for SAS 9.2, the MEANS procedure has been enhanced to run inside the Teradata Enterprise Data Warehouse (EDW), DB2 under UNIX, and Oracle. Using conventional processing, a SAS procedure, by means of the SAS/ACCESS engine, receives all the rows of the table from the database. All processing is done by the procedure. Large tables mean that a significant amount of data must be transferred. Using the new in-database technology, the procedures that are enabled for processing inside the database generate more sophisticated queries that allow the aggregations and analytics to be run inside the database. For most in-database procedures, a much smaller result set is returned for the remaining analysis that is required to produce the final output. As a result of using the in-database procedures, more work is done inside the database and less data movement can occur. Using in-database procedures can result in significant performance improvements.

The MIGRATE Procedure

The MIGRATE procedure now supports more cross-environment migrations. You can migrate a SAS 8.2 data library from almost every SAS 8.2 operating environment to

any SAS 9.2 operating environment. Most SAS 6 operating environments are also supported, but not for cross-environment migration.

The OPTIONS Procedure

The following enhancements have been made to the OPTIONS procedure:

- Restricted options are now supported in all operating environments.
- The value of environment variables can be displayed by using the EXPAND option.
- System options that have a character value can be displayed as a hexadecimal value by using the HEXVALUE option.
- You can display a list of SAS system option groups by using the LISTGROUPS option.
- To display the options in multiple groups, you can list more than one group in the GROUP= option.
- The following system option groups are new and can be specified on the GROUP= option: CODEGEN, LOGCONTROL, LISTCONTROL, SMF, SQL, and SVG.

The PRINT Procedure

The following new options have been added to the PRINT procedure:

SUMLABEL
enables you to display the label of the BY variable on the summary line.

BLANKLINE
enables you to insert a blank line after every *n* observations.

The PWENCODE Procedure

The following enhancements have been made to the PWENCODE procedure:

- Encoded passwords are now supported for SAS data sets.
- The sas003 encoding method, which uses a 256-bit key to generate encoded passwords, is now supported. The sas003 encoding method supports the AES (Advanced Encryption Standard), which is a new security algorithm for SAS/SECURE.
-

The RANK Procedure

The TIES= option of the RANK procedure has a new value, DENSE, which computes scores and ranks by treating tied values as a single-order statistic.

In the third maintenance release for SAS 9.2, the RANK procedure has been enhanced to run inside the Teradata Enterprise Data Warehouse (EDW), DB2 under UNIX, and Oracle. Using conventional processing, a SAS procedure, by means of the SAS/ACCESS engine, receives all the rows of the table from the database. All processing is done by the procedure. Large tables mean that a significant amount of data must be transferred. Using the new in-database technology, the procedures that are enabled for processing inside the database generate more sophisticated queries that allow the aggregations and analytics to be run inside the database. For most in-database procedures, a much smaller result set is returned for the remaining analysis that is required to produce the final output. As a result of using the in-database

procedures, more work is done inside the database and less data movement can occur. Using in-database procedures can result in significant performance improvements.

The REPORT Procedure

The following enhancements have been made to the REPORT procedure:

- The PROBT statistic is now an alias for the PRT statistic.
- The MODE statistic can now be used with PROC REPORT.
- The STYLE/MERGE attribute name option has been added so that styles can be concatenated. Currently, there is no way to concatenate styles using a CALL DEFINE statement. Each time the CALL DEFINE statement is executed, it replaces any previous style information.
- The BY statement is now available when requesting an output data set with the OUT= option in the PROC REPORT statement.
- The new Table of Contents (TOC) now supports the CONTENTS= option in the BREAK, RBREAK, and DEFINE statements.
- The BYPAGENO=n option has been added to reset the page number between BY groups.
- The SPANROWS option has been added for the PROC REPORT statement. This option permits the GROUP and ORDER variables to be contained in a box rather than in blank cells appearing underneath the GROUP or ORDER variable values.
- The SPANROWS option also permits GROUP and ORDER variable values to repeat when the values break across pages in PDF, PS, and RTF destinations.
- PROC REPORT now supports the ODS DOCUMENT and ODS OUTPUT destinations.
- PROC REPORT now supports style attributes BORDERBOTTOMSTYLE, BORDERBOTTOMWIDTH, BORDERBOTTOMCOLOR, BORDERTOPSTYLE, BORDERTOPWIDTH, and BORDERTOPCOLOR.
- In the third maintenance release for SAS 9.2, the REPORT procedure has been enhanced to run inside the Teradata Enterprise Data Warehouse (EDW), DB2 under UNIX, and Oracle. Using conventional processing, a SAS procedure, by means of the SAS/ACCESS engine, receives all the rows of the table from the database. All processing is done by the procedure. Large tables mean that a significant amount of data must be transferred. Using the new in-database technology, the procedures that are enabled for processing inside the database generate more sophisticated queries that allow the aggregations and analytics to be run inside the database. For most in-database procedures, a much smaller result set is returned for the remaining analysis that is required to produce the final output. As a result of using the in-database procedures, more work is done inside the database and less data movement can occur. Using in-database procedures can result in significant performance improvements.

The SOAP Procedure

In the third maintenance release for SAS 9.2, the following options have been added to the SOAP procedure:

ENVFILE
specifies the location of the SAS environments file.

ENVIRONMENT
specifies to use the environment that is defined in the SAS environments file.

SERVICE
specifies the SAS Web service to use.

In the third maintenance release for SAS 9.2, you can call SAS Web services by using one of two methods. The first method requires that you know the URL of the Service Registry Service and the URL of the endpoint of the service you are calling. You must set the URL of the Service Registry Service on the SRSURL option. The URL option indicates the endpoint of the service that you are calling.

The second method that is used to call SAS Web services uses the SAS environments file to specify the endpoint of the service you are calling. Using this method, you can indicate the location of the SAS environments file in one of two ways:

- use the ENVFILE option in PROC SOAP
- define the Java property **`env.definition.location`** in JREOPTIONS on the SAS command line or in the SAS configuration file

The SORT Procedure

The following options and statements are new or enhanced in the SORT procedure :

- The new PRESORTED option causes PROC SORT to check within the input data set to determine whether the observations are in order before sorting. Use the PRESORTED option when you know or strongly suspect that a data set is already in order according to the key variables specified in the BY statement. By specifying this option, you avoid the cost of sorting the data set.
- The SORTSEQ= option is enhanced. New suboptions have been added as follows:
  - The LINGUISTIC suboption specifies linguistic collation, which sorts characters according to rules of language. The rules and default collating sequence options are based on the language specified in the current locale setting. You can modify the default collating rules of linguistic collation. The following are the collating rules that can be used to modify the LINGUISTIC collation suboption:
    - ALTERNATE_HANDLING=
    - CASE_FIRST=
    - COLLATION=
    - LOCALE=
    - NUMERIC_COLLATION=
    - STRENGTH=
  - You can now specify all possible encoding values. The result is the same as a binary collation of the character data represented in the specified encoding. The encoding values available are found in the *SAS National Language Support (NLS): Reference Guide*.
  - The KEY statement has been added to PROC SORT. You can specify multiple KEY statements and multiple variables per KEY statement. You can specify the DESCENDING option to change the default collating direction from ascending to descending.

In the third maintenance release for SAS 9.2, the SORT procedure has been enhanced to run inside the Teradata Enterprise Data Warehouse (EDW), DB2 under UNIX, and Oracle. Using conventional processing, a SAS procedure, by means of the SAS/ACCESS engine, receives all the rows of the table from the database. All processing is done by the procedure. Large tables mean that a significant amount of data must be transferred. Using the new in-database technology, the procedures that are enabled for

processing inside the database generate more sophisticated queries that allow the aggregations and analytics to be run inside the database. For most in-database procedures, a much smaller result set is returned for the remaining analysis that is required to produce the final output. As a result of using the in-database procedures, more work is done inside the database and less data movement can occur. Using in-database procedures can result in significant performance improvements.

The SUMMARY Procedure

The following enhancements have been made to the SUMMARY procedure:

In the third maintenance release after SAS 9.2, the SUMMARY procedure has been enhanced to run inside the Teradata Enterprise Data Warehouse (EDW), DB2 under UNIX, and Oracle. Using conventional processing, a SAS procedure, by means of the SAS/ACCESS engine, receives all the rows of the table from the database. All processing is done by the procedure. Large tables mean that a significant amount of data must be transferred. Using the new in-database technology, the procedures that are enabled for processing inside the database generate more sophisticated queries that allow the aggregations and analytics to be run inside the database. For most in-database procedures, a much smaller result set is returned for the remaining analysis that is required to produce the final output. As a result of using the in-database procedures, more work is done inside the database and less data movement can occur. Using in-database procedures can result in significant performance improvements.

The SQL Procedure

The following enhancements have been made to the SQL procedure:

- A number of features have been added which enable you to optimize queries.
  - Depending on which engine type the query uses, you can replace the PUT function with a logically equivalent expression.
  - You can replace references to the DATE, TIME, DATETIME, and TODAY functions in a query to their equivalent constant values before the query executes.
  - You can specify the minimum number of rows that must be in a table or the maximum number of SAS format values that can exist in a PUT function in order for PROC SQL to consider optimizing the PUT function.
  - You can bypass the remerging process when a summary function is used in a SELECT clause or a HAVING clause.
  - If indexing is present, PROC SQL now uses the index files when processing SELECT DISTINCT statements.
  - Semicolons can now be used in explicit queries for pass–through.
- You can use custom functions that are created with PROC FCMP in PROC SQL.
- The DICTIONARY.EXTFILES table will now include the access method and device type information.
- Three new DICTIONARY tables have been added. The FUNCTIONS table contains information about currently accessible functions. The INFOMAPS table returns information on all known information maps. The DESTINATIONS table contains information about all known ODS destinations.
- The DESCRIBE TABLE CONSTRAINTS statement will not display the names of password-protected foreign key data set variables that reference the primary key constraint.

- The TRANSCODE=NO argument is not supported by some SAS Workspace Server clients. In SAS 9.2, if the argument is not supported, column values with TRANSCODE=NO are replaced (masked) with asterisks (*). Before SAS 9.2, column values with TRANSCODE=NO were transcoded.
- The SAS/ACCESS CONNECT statement has a new AUTHDOMAIN option that supports lookup of security credentials (user ID and password) without your having to explicitly specify the credentials.

The following new options have been added to the PROC SQL statement:

CONSTDATETIME | NOCONSTDATETIME
specifies whether the SQL procedure replaces references to the DATE, TIME, DATETIME, and TODAY functions in a query with their equivalent constant values before the query executes.

Note: The CONSTDATETIME option provides the same functionality as the new SQLCONSTDATETIME system option. △

EXITCODE
specifies whether PROC SQL clears an error code for any SQL statement.

IPASSTHRU | NOIPASSTHRU
specifies whether implicit pass-through is enabled or disabled.

REDUCEPUT
specifies the engine type that a query uses for which optimization is performed by replacing a PUT function in a query with a logically equivalent expression.

Note: The REDUCEPUT option provides the same functionality as the new SQLREDUCEPUT system option. △

REMERGE | NOREMERGE
specifies that the SQL procedure does not process queries that use remerging of data.

Note: The REMERGE option provides the same functionality as the new SQLREMERGE system option. △

The following new global system options affect SQL processing and performance:

DBIDIRECTEXEC (SAS/ACCESS)
controls SQL optimization for SAS/ACCESS engines.

SQLCONSTANTDATETIME
specifies whether the SQL procedure replaces references to the DATE, TIME, DATETIME, and TODAY functions in a query with their equivalent constant values before the query executes.

SQLMAPPUTTO (SAS/ACCESS)
for SAS 9.2 Phase 2 and later, specifies whether the PUT function in the SQL procedure is processed by SAS or by the SAS_PUT( ) function inside the Teradata database.

SQLREDUCEPUT
for the SQL procedure, specifies the engine type that a query uses for which optimization is performed by replacing a PUT function in a query with a logically equivalent expression.

SQLREDUCEPUTOBS
for the SQL procedure when the SQLREDUCEPUT= system option is set to NONE, specifies the minimum number of observations that must be in a table in order for PROC SQL to consider optimizing the PUT function in a query.

SQLREDUCEPUTVALUES
for the SQL procedure when the SQLREDUCEPUT= system option is set to NONE, specifies the maximum number of SAS format values that can exist in a PUT function expression in order for PROC SQL to consider optimizing the PUT function in a query.

SQLREMERGE
specifies whether the SQL procedure can process queries that use remerging of data.

SQLUNDOPOLICY
specifies whether the SQL procedure keeps or discards updated data if errors occur while the data is being updated.

The TABULATE Procedure

The following enhancements have been made to the TABULATE procedure:

- The PROBT statistic is now an alias for the PRT statistic.
- The MODE statistic can now be used with PROC TABULATE.
- You can specify variable name list shortcuts within the TABLE statement.
- PROC TABULATE now supports style attributes BORDERBOTTOMSTYLE, BORDERBOTTOMWIDTH, BORDERBOTTOMCOLOR, BORDERTOPSTYLE, BORDERTOPWIDTH, and BORDERTOPCOLOR.
- In the third maintenance release for SAS 9.2, the TABULATE procedure has been enhanced to run inside the Teradata Enterprise Data Warehouse (EDW), DB2 under UNIX, and Oracle. Using conventional processing, a SAS procedure, by means of the SAS/ACCESS engine, receives all the rows of the table from the database. All processing is done by the procedure. Large tables mean that a significant amount of data must be transferred. Using the new in-database technology, the procedures that are enabled for processing inside the database generate more sophisticated queries that allow the aggregations and analytics to be run inside the database. For most in-database procedures, a much smaller result set is returned for the remaining analysis that is required to produce the final output. As a result of using the in-database procedures, more work is done inside the database and less data movement can occur. Using in-database procedures can result in significant performance improvements.

The UNIVARIATE Procedure

The UNIVARIATE procedure now produces graphs that conform to ODS styles, so that creating consistent output is easier. Also, you now have two methods for producing graphs. With traditional graphics, you can control every detail of a graph through familiar procedure syntax and the GOPTION and SYMBOL statements. With ODS Graphics (experimental for the UNIVARIATE procedure in SAS 9.2), you can obtain the highest quality output with minimal syntax. You also now have full compatibility with graphics that are produced by the SAS/STAT and SAS/ETS procedures.

The new UNIVARIATE procedure CDFPLOT statement plots the observed cumulative distribution function (cdf) of a variable and enables you to superimpose a fitted theoretical distribution on the graph. The new PPPLOT statement creates a probability-probability plot (also referred to as a P-P plot or percent plot). This statement compares the empirical cumulative distribution function (ecdf) of a variable with a specified theoretical cumulative distribution function. The beta, exponential, gamma, lognormal, normal, and Weibull distributions are available in both statements.

Documentation Enhancements

The following Base SAS Procedures have had part or all of their documentation relocated to other SAS documents.

The CV2VIEW Procedure

Documentation for the CV2VIEW procedure is now in the *SAS/ACCESS for Relational Databases: Reference*.

The DBCSTAB Procedure

Documentation for the DBCSTAB procedure is now in the *SAS National Language Support (NLS): Reference Guide*.

The EXPORT Procedure

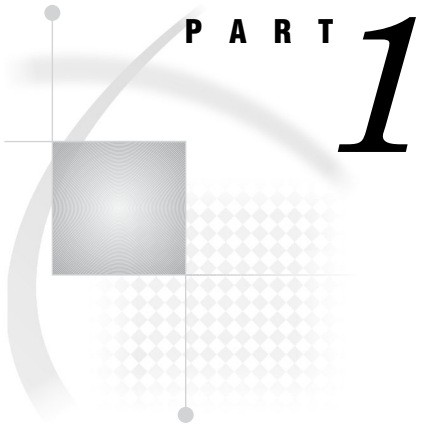
The *Base SAS Procedures Guide* contains only overview and common syntax information for the EXPORT procedure. Information that is specific to PC Files is now in the *SAS/ACCESS Interface to PC Files: Reference*.

The IMPORT Procedure

The *Base SAS Procedures Guide* contains only overview and common syntax information for the IMPORT procedure. Information that is specific to PC Files is now in the *SAS/ACCESS Interface to PC Files: Reference*.

The TRANTAB Procedure

Documentation for the TRANTAB procedure is now in the *SAS National Language Support (NLS): Reference Guide*.



PART 1

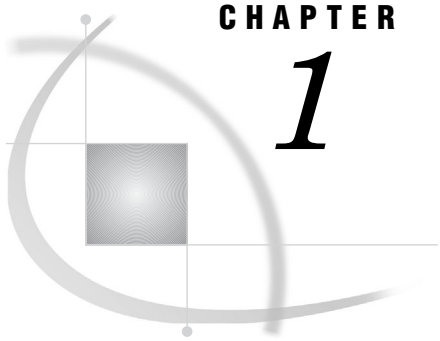
Concepts

Chapter **1** . . . . . **Choosing the Right Procedure** *3*

Chapter **2** . . . . . **Fundamental Concepts for Using Base SAS Procedures** *17*

Chapter **3** . . . . . **Statements with the Same Function in Multiple Procedures** *35*

Chapter **4** . . . . . **In-Database Processing of Base Procedures** *49*



CHAPTER

1

Choosing the Right Procedure

Functional Categories of Base SAS Procedures **3**
Report Writing **3**
Statistics **3**
Utilities **4**
Report-Writing Procedures **5**
Statistical Procedures **6**
Available Statistical Procedures **6**
Efficiency Issues **7**
Quantiles **7**
Computing Statistics for Groups of Observations **7**
Additional Information about the Statistical Procedures **8**
Utility Procedures **8**
Brief Descriptions of Base SAS Procedures **10**

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,

crosstabulations, correlations, and distribution tests. They also rank and standardize data.

CHART	RANK	SUMMARY
CORR	REPORT	TABULATE
FREQ	SQL	UNIVARIATE
MEANS	STANDARD	

Utilities

These procedures perform 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	FONTREG	PRINTTO
BMDP*	FORMAT	PROTO
CATALOG	FSLIST	PRTDEF
CIMPORT	IMPORT	PRTEXP
COMPARE	INFOMAPS++	PWENCODE
CONTENTS	JAVAINFO	REGISTRY
CONVERT*	METADATA@@	RELEASE*
COPY	METALIB@@	SORT
CPORT	METAOPERATE@@	SOURCE*
CV2VIEW@	MIGRATE	SQL
DATASETS	OPTIONS	TAPECOPY*
DBCSTAB#	OPTLOAD	TAPELABEL*
DISPLAY	OPTSAVE	TEMPLATE+
DOCUMENT+	PDS*	TRANSPOSE
EXPORT	PDSCOPY*	TRANTAB#
FCMP	PMENU	XSL

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

Report-Writing Procedures

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

Table 1.1 Report-Writing Procedures by Task

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

Report Type	Procedure	Description
	PLOT	produces scatter diagrams that plot one variable against another.
	TIMEPLOT	produces plots of one or more variables over time intervals.

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

Statistical Procedures

Available Statistical Procedures

The following table lists statistical procedures according to task. Table A1.1 on page 1537 lists the most common statistics and the procedures that compute them.

Table 1.2 Elementary Statistical Procedures by Task

Report type	Procedure...	Description
Descriptive statistics	CORR	computes simple descriptive statistics.
	MEANS or SUMMARY	computes descriptive statistics; can produce printed output and output data sets. By default, PROC MEANS produces printed output, and PROC SUMMARY creates an output data set.
	REPORT	computes most of the same statistics as PROC TABULATE; allows customization of format.
	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 computes test 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 measures of dependence (D) and Cronbach's coefficient alpha.
Distribution analysis	UNIVARIATE	computes tests for location and tests for normality.

Report type	Procedure...	Description
	FREQ	computes a test for the binomial proportion for one-way tables; computes a goodness-of-fit test for one-way tables; computes a chi-square test of equal distribution for two-way tables.
Robust estimation	UNIVARIATE	computes robust estimates of scale, trimmed means, and Winsorized means.
Data transformation		
Computing ranks	RANK	computes ranks for one or more numeric variables across the observations of a SAS data set and creates an output data set; can produce normal scores or other rank scores.
Standardizing data	STANDARD	creates an output data set that contains variables that are standardized to a given mean and standard deviation.
Low-resolution graphics*		
	CHART	produces a graphical report that can show one of the following statistics for the chart variable: frequency counts, percentages, cumulative frequencies, cumulative percentages, totals, or averages.
	UNIVARIATE	produces descriptive plots such as stem-and-leaf plot, box plots, and normal probability plots.

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

Efficiency Issues

Quantiles

For a large sample size n, the calculation of quantiles, including the median, requires computing time proportional to $n\log(n)$. Therefore, a procedure, such as UNIVARIATE, that automatically calculates quantiles might require more time than other data summarization procedures. Furthermore, because data is held in memory, the procedure also requires more storage space to perform the computations. By default, the report procedures PROC MEANS, PROC SUMMARY, and PROC TABULATE require less memory because they do not automatically compute quantiles. These procedures also provide an option to use a new fixed-memory, quantiles estimation method that is usually less memory-intense. See “Quantiles” on page 643 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 might require substantial computer resources. A more efficient way to compute statistics within groups without sorting is to use a CLASS statement with one of the following procedures: MEANS, SUMMARY, or TABULATE.

Additional Information about the Statistical Procedures

Appendix 1, "SAS Elementary Statistics Procedures," on page 1535, lists standard keywords, statistical notation, and formulas for the statistics that Base SAS procedures compute frequently. The sections on the individual statistical procedures discuss the statistical concepts that are useful to interpret a procedure output.

Utility Procedures

The following table groups utility procedures according to task.

Table 1.3 Utility Procedures by Task

Tasks	Procedure	Description
Supply information	COMPARE	compares the contents of two SAS data sets.
	CONTENTS	describes the contents of a SAS library or specific library members.
	JAVAINFO	conveys diagnostic information about the Java environment that SAS is using.
	OPTIONS	lists the current values of all SAS system options.
	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	FCMP	enables creation, testing, and storage of SAS functions and subroutines before they are used in other SAS procedures.
	FSLIST	browses external files such as files that contain SAS source lines or SAS procedure output.

Tasks	Procedure	Description
	INFOMAPS[++]	creates or updates a SAS Information Map.
	SQL	creates SAS data sets using Structured Query Language and SAS features.
Transform data	DBCSTAB[#]	produces conversion tables for the double-byte character sets that SAS supports.
	FORMAT	creates user-defined informats to read data and user-defined formats to display data.
	SORT	sorts SAS data sets by one or more variables.
	SQL	sorts SAS data sets by one or more variables.
	TRANSPOSE	transforms SAS data sets so that observations become variables and variables become observations.
	TRANTAB[#]	creates, edits, and displays customized translation tables.
	XSL	transforms an XML document into another format.
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.
	CATALOG	manages SAS catalog entries.
	CIMPORT	restores a transport sequential file that PROC CPORT creates (usually in another operating environment) to its original form as a SAS catalog, a SAS data set, or a SAS library.
	CONVERT[*]	converts BMDP system files, OSIRIS system files, and SPSS portable files to SAS data sets.
	COPY	copies a SAS library or specific members of the library.
	CPORT	converts a SAS catalog, a SAS data set, or a SAS library to a transport sequential file that PROC CIMPORT can restore (usually in another operating environment) to its original form.
	CV2VIEW[@]	converts SAS/ACCESS view descriptors to PROC SQL views.
	DATASETS	manages SAS files.
	EXPORT	reads data from a SAS data set and writes them to an external data source.
	IMPORT	reads data from an external data source and writes them to a SAS data set.
	MIGRATE	migrates members in a SAS library forward to the most current release of SAS.
	PDS[*]	lists, deletes, and renames the members of a partitioned data set.
	PDSCOPY[*]	copies partitioned data sets from disk to tape, disk to disk, tape to tape, or tape to disk.
	PROTO	enables registration, in batch mode, of external functions that are written in the C or C++ programming languages.

Tasks	Procedure	Description
	REGISTRY	imports registry information to the USER portion of the SAS registry.
	RELEASE*	releases unused space at the end of a disk data set under the z/OS environment.
	SOURCE*	provides an easy way to back up and process source library data sets.
	SQL	concatenates SAS data sets.
	TAPECOPY*	copies an entire tape volume or files from one or more tape volumes to one output tape volume.
	TAPELABEL*	lists the label information of an IBM standard-labeled tape volume in the z/OS environment.
Control windows	PMENU	creates customized menus for SAS applications.
Miscellaneous	DISPLAY	executes SAS/AF applications.
	PWENCODE	encodes passwords for use in SAS programs.
Manage metadata in a SAS Metadata Repository	METADATA@@	sends a method call, in the form of an XML string, to a SAS Metadata Server.
	METALIB@@	updates metadata to match the tables in a library.
	METAOPERATE@@	performs administrative tasks on a metadata server.

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

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

CHART procedure

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

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

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

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

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

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

CORR procedure
computes Pearson product-moment and weighted product-moment correlation coefficients between variables and descriptive statistics for these variables. In addition, PROC CORR can compute three nonparametric measures of association (Spearman's rank-order correlation, Kendall's tau-b, and Hoeffding's measure of dependence, D), partial correlations (Pearson's partial correlation, Spearman's partial rank-order correlation, and Kendall's partial tau-b), and Cronbach's coefficient alpha.

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

CV2VIEW procedure
converts SAS/ACCESS view descriptors to PROC SQL views. Starting in SAS System 9, conversion of SAS/ACCESS view descriptors to PROC SQL views is recommended because PROC SQL views are platform-independent and enable you to use the LIBNAME statement. See the *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 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. See the *SAS National Language Support (NLS): Reference Guide* for details.

DISPLAY procedure
executes SAS/AF applications. See the *SAS Guide to Applications Development* for information on building SAS/AF applications.

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.

FCMP procedure
enables you to create, test, and store SAS functions and subroutines before you use them in other SAS procedures. PROC FCMP accepts slight variations of DATA step statements. Most features of the SAS programming language can be used in functions and subroutines that are processed by PROC FCMP.

FONTREG procedure
adds system fonts to the SAS registry.

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

FREQ procedure
produces one-way to *n*-way frequency tables and reports frequency counts. PROC FREQ can compute chi-square tests for one-way to *n*-way tables; for tests and measures of association and of agreement for two-way to *n*-way 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.

INFOMAPS
creates or updates a SAS Information Map. See the *Base SAS Guide to Information Maps* for details.

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

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

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

METALIB procedure

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

METAOPERATE procedure
performs administrative tasks on a metadata server.

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

OPTIONS procedure
lists the current values of all SAS system options.

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

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

PDS procedure
lists, deletes, and renames the members of a partitioned data set. See the *SAS Companion for z/OS* 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 Companion for z/OS* 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.

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

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

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

PWENCODE procedure
encodes passwords for use in SAS programs.

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 Companion for z/OS* for more information.

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

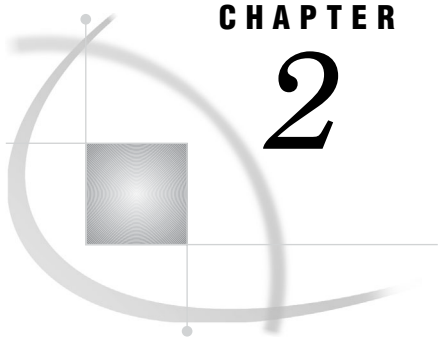
produces plots of one or more variables over time intervals.

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

TRANTAB procedure
creates, edits, and displays customized translation tables. See *SAS National Language Support (NLS): Reference Guide* for more information.

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

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



CHAPTER

2

Fundamental Concepts for Using Base SAS Procedures

Language Concepts **17**
Temporary and Permanent SAS Data Sets **18**
Naming SAS Data Sets **18**
USER Library **18**
SAS System Options **18**
Data Set Options **19**
Global Statements **20**
Procedure Concepts **20**
Input Data Sets **20**
RUN-Group Processing **21**
Creating Titles That Contain BY-Group Information **21**
BY-Group Processing **21**
Suppressing the Default BY Line **21**
Inserting BY-Group Information into a Title **21**
Example: Inserting a Value from Each BY Variable into the Title **22**
Example: Inserting the Name of a BY Variable into a Title **24**
Example: Inserting the Complete BY Line into a Title **24**
Error Processing of BY-Group Specifications **25**
Shortcuts for Specifying Lists of Variable Names **25**
Formatted Values **26**
Using Formatted Values **26**
Example: Printing the Formatted Values for a Data Set **26**
Example: Grouping or Classifying Formatted Data **28**
Example: Temporarily Associating a Format with a Variable **29**
Example: Temporarily Dissociating a Format from a Variable **30**
Formats and BY-Group Processing **31**
Formats and Error Checking **31**
Processing All the Data Sets in a Library **31**
Operating Environment-Specific Procedures **31**
Statistic Descriptions **32**
Computational Requirements for Statistics **33**
Output Delivery System **33**

Language Concepts

Temporary and Permanent SAS Data Sets

Naming SAS Data Sets

SAS data sets can have a one-level name or a two-level name. Typically, names of temporary SAS data sets have only one level and are stored in the WORK library. The WORK library is defined automatically at the beginning of the SAS session and is deleted automatically at the end of the SAS session. Procedures assume that SAS data sets that are specified with a one-level name are to be read from or written to the WORK library. To indicate otherwise, you specify a USER library (see "USER Library" on page 18). For example, the following PROC PRINT steps are equivalent. The second PROC PRINT step assumes that the DEBATE data set is in the WORK library.

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

proc print data=debate;
run;
```

The SAS system options WORK=, WORKINIT, and WORKTERM affect how you work with temporary and permanent libraries. See the *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 library*. A SAS library is an external storage location that stores SAS data sets in your operating environment. A LIBNAME statement associates the libref with the SAS library. In the following PROC PRINT step, PROCLIB is the libref and EMP is the SAS data set within the library:

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

USER Library

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

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

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

SAS System Options

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

BYLINE | NOBYLINE

DATE | NODATE
DETAILS | NODETAILS
FMTERR | NOFMTERR
FORMCHAR=
FORMDLIM=
LABEL | NOLABEL
LINESIZE=
NUMBER | NONUMBER
PAGENO=
PAGESIZE=
REPLACE | NOREPLACE
SOURCE | NOSOURCE

For a complete description of SAS system options, see the *SAS 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 as follows:

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

KEEP=	WRITE=
LABEL=	

For a complete description of SAS data set options, see the *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 the *SAS Language Reference: Dictionary*. For information about the ODS statement, refer to the "Output Delivery System" on page 33 and to *SAS Output Delivery System: User's Guide*.

Procedure Concepts

Input Data Sets

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

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

If you omit the DATA= option, the procedure uses the value of the SAS system option _LAST_=. The default of _LAST_= is the most recently created SAS data set in the current SAS job or session. _LAST_= is described in detail in the *SAS 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 suppress the default BY line. To suppress it, use the SAS system option NOBYLINE.

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

#BY-specification<.suffix>

BY-specification

is one of the following specifications:

BYVAL*n* | BYVAL(*BY-variable*)
places the value of the specified BY variable in the title. You specify the BY variable with one of the following options:

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

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

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

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

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

BYLINE
inserts the complete default BY line into the title.

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

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

This example demonstates these actions:

1. creates a data set, GROC, that contains data for stores from four regions. Each store has four departments. See "GROC" on page 1611 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 does these actions.

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

```
options nobyline nodate pageno=1
        linesize=64 pagesize=20;  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;  ❸
```

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

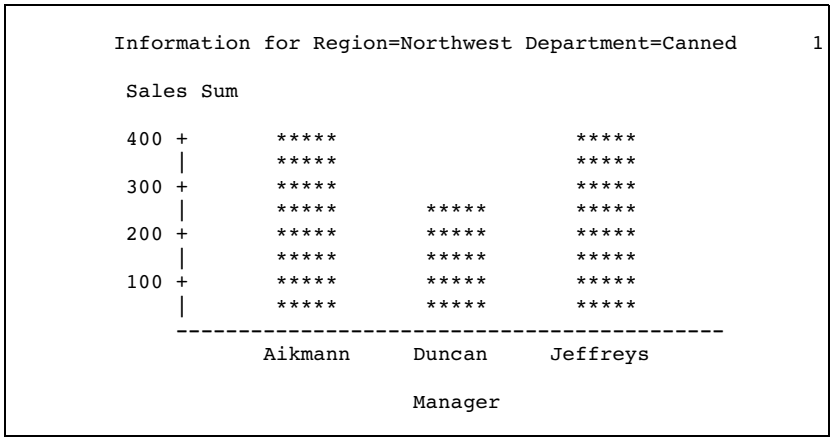
This example inserts the complete BY line into the title. The program does these actions:

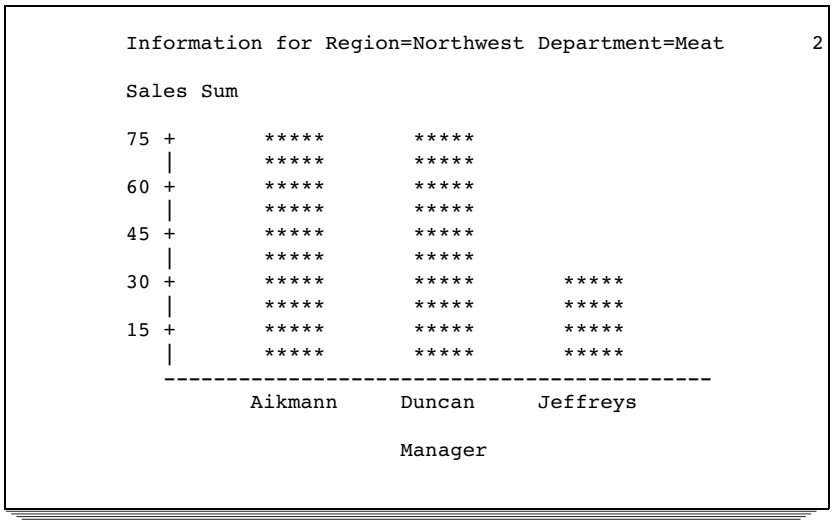
1. uses the SAS system option NOBYLINE to suppress the BY line that normally appears in output that is produced with BY-group processing.
2. uses PROC CHART to chart sales by Region and Department. In the TITLE statement, #BYLINE inserts the complete BY line into the title.
3. uses the SAS system option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

```
options nobyline nodate pageno=1
        linesize=64 pagesize=20;  ❶
```

```
proc chart data=groc;  ❷
   by region department;
   vbar manager / type=sum sumvar=sales;
   title 'Information for #byline';
run;
options byline;  ❸
```

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





Error Processing of BY-Group Specifications

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

Shortcuts for Specifying Lists of Variable Names

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

Notation	Meaning
x1-x*n*	specifies variables X1 through X*n*. The numbers must be consecutive.
x:	specifies all variables that begin with the letter X.
x--a	specifies all variables between X and A, inclusive. This notation uses the position of the variables in the data set.
x-numeric-a	specifies all numeric variables between X and A, inclusive. This notation uses the position of the variables in the data set.
x-character-a	specifies all character variables between X and A, inclusive. This notation uses the position of the variables in the data set.
numeric	specifies all numeric variables.
character	specifies all character variables.
all	specifies all variables.

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

See the *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 SAS procedures use formatted values.

Example: Printing the Formatted Values for a Data Set

The following example prints the formatted values of the data set PROCLIB.PAYROLL. (See "PROCLIB.PAYROLL" on page 1618 for details about 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-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;
```

The following example 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 determine whether a format is assigned to the BY variable. If it is, then the procedure adds observations to the current BY groups until the formatted value changes. If *nonconsecutive* internal values of the BY variables have the same formatted value, then the values are grouped into different BY group. Thus, two BY groups are created with the same formatted value. Further, if different and *consecutive* internal values of the BY variables have the same formatted value, then they are included in the same BY group.

Formats and Error Checking

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

Note: To ensure that SAS can find user-written formats, use the SAS system option FMTSEARCH=. How to store formats is described in "Storing Informats and Formats" on page 539. △

Processing All the Data Sets in a Library

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

Example 9 on page 882 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 1571 contains a table with additional information. These procedures are described in more detail in the SAS documentation for operating environments.

Statistic Descriptions

The following table identifies common descriptive statistics that are available in several Base SAS procedures. See "Keywords and Formulas" on page 1536 for more detailed information about available statistics and theoretical information.

Table 2.1 Common Descriptive Statistics That Base SAS 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 computational requirements are for the statistics that are listed in Table 2.1 on page 32. They do not describe recommended sample sizes.

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

Statistics are reported as missing if they cannot be computed.

Output Delivery System

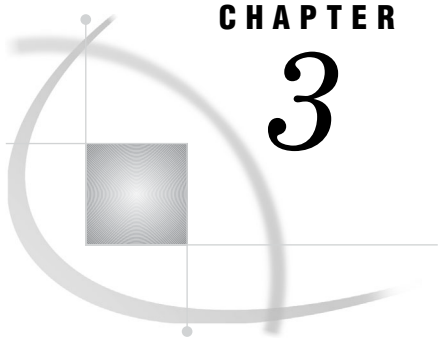
The Output Delivery System (ODS) gives you greater flexibility in generating, storing, and reproducing SAS procedure and DATA step output, with a wide range of formatting options. ODS provides formatting functionality that is not available from individual procedures or from the DATA step alone. ODS overcomes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

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

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

procedure, then you relied on PROC PRINTTO and the DATA step to retrieve results.

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



CHAPTER

3

Statements with the Same Function in Multiple Procedures

Overview **35**
Statements **36**
BY **36**
FREQ **39**
QUIT **41**
WEIGHT **41**
WHERE **46**

Overview

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

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

FORMAT
affects the procedure output and the output data set. The FORMAT statement does not permanently alter the variables in the input data set. The DEFAULT= option is not valid. See the *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 39.

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

QUIT
executes any statements that have not executed and ends the procedure. See "QUIT" on page 41.

WEIGHT
specifies weights for analysis variables in the statistical calculations. See "WEIGHT" on page 41.

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 46.

Statements

BY

Orders the output according to the BY groups.

See also: "Creating Titles That Contain BY-Group Information" on page 21

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

Required Arguments

variable
specifies the variable that the procedure uses to form BY groups. You can specify more than one variable. If you do not use the NOTSORTED option in the BY statement, then either the observations in the data set must be sorted by all the variables that you specify, or they must be indexed appropriately. Variables in a BY statement are called *BY variables*.

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 might need to use the NOTSORTED or DESCENDING option in the BY statement in the PROC step.

- For more information on the BY statement, see the *SAS Language Reference: Dictionary*.
- For more information on PROC SORT, see Chapter 54, "The SORT Procedure," on page 1165.
- For more information on creating indexes, see "INDEX CREATE Statement" on page 339.

Formatting BY-Variable Values

When a procedure is submitted with a BY statement, the following actions are taken with respect to processing of BY groups:

1. The procedure determines whether the data is sorted by the internal (unformatted) values of the BY variable(s).
2. The procedure determines whether a format has been applied to the BY variable(s). If the BY variable is numeric and has no user-applied format, then the BEST12. format is applied for the purpose of BY-group processing.
3. The procedure continues adding observations to the current BY group until both the internal and the formatted values of the BY variable or variables change.

This process can have unexpected results if, for example, nonconsecutive internal BY values share the same formatted value. In this case, the formatted value is represented in different BY groups. Alternatively, if different consecutive internal BY values share the same formatted value, then these observations are grouped into the same BY group.

Base SAS Procedures That Support the BY Statement

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

FREQ	TABULATE
MEANS	TIMEPLOT
PLOT	TRANSPOSE
PRINT	UNIVARIATE
RANK	

Note: In the SORT procedure, the BY statement specifies how to sort the data. In the other procedures, the BY statement specifies how the data is currently sorted. △

Example

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

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

```
                    Printing of Team Members                    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 represents a ship's course and speed (in nautical miles per hour), recorded every hour. The frequency variable Hours represents the number of hours that the ship maintained the same course and speed. Each of the following PROC MEANS steps calculates average course and speed. The different results demonstrate the effect of using Hours as a frequency variable.

The following PROC MEANS step does not use a frequency variable:

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

data track;
   input Course Speed Hours @@;
   datalines;
30  4  8 50 7 20
75 10 30 30 8 10
80  9 22 20 8 25
83 11  6 20 6 20
;


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

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

```
                  Average Course and Speed                    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. 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	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.

Before Version 7 of SAS, no Base SAS procedure excluded the observations with missing weights from the analysis. Most SAS/STAT procedures, such as PROC GLM, have always excluded not only missing weights but also negative and zero weights from the analysis. You can achieve this same behavior in a Base SAS procedure that supports the WEIGHT statement by using the EXCLNPWGT option in the PROC statement.

The procedure substitutes the value of the WEIGHT variable for w_i, which appears in "Keywords and Formulas" on page 1536.

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 ith observation, because this variance involves the observation's weight, which varies from observation to observation.

If the values of your variable are counts that represent the number of occurrences of each observation, then use this variable in the FREQ statement rather than in the WEIGHT statement. In this case, because the values are counts, they should be integers. (The FREQ statement truncates any noninteger values.) The variance that is computed with a FREQ variable is an estimate of the common variance σ^2 of the observations.

Note: If your data comes from a stratified sample where the weights w_i represent the strata weights, then neither the WEIGHT statement nor the FREQ statement provides appropriate stratified estimates of the mean, variance, or variance of the mean. To perform the appropriate analysis, consider using PROC SURVEYMEANS, which is a SAS/STAT procedure that is documented in the *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;
```

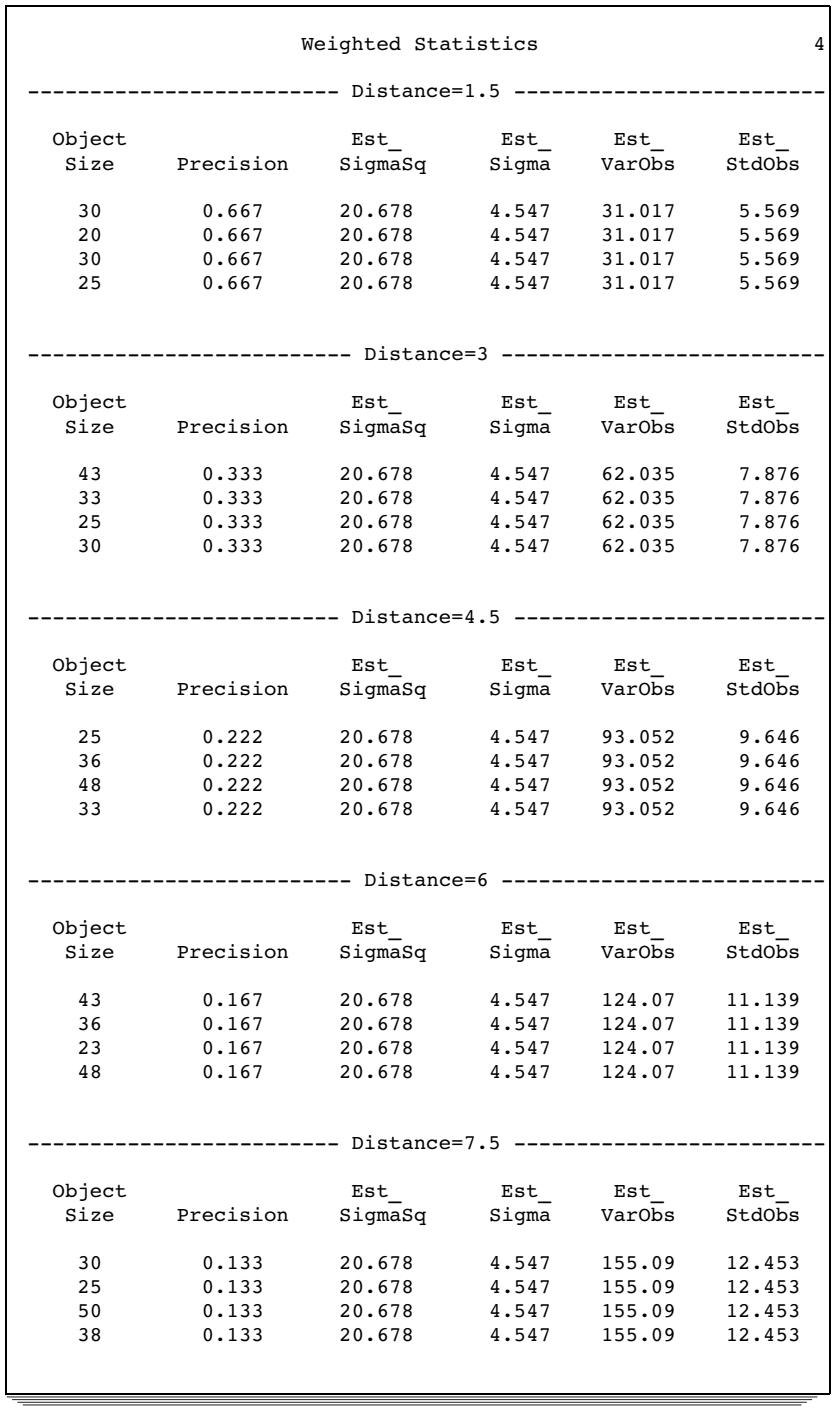
The variance of the *i*th observation is assumed to be $var\left(x_i\right) = \sigma^2/w_i$ and w_i is the weight for the *i*th observation. In the first PROC MEANS step, the computed variance is an estimate of σ^2. In the second PROC MEANS step, the computed variance is an estimate of $\left(n - 1/n\right)\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

```
          Weighted Analysis Using Default VARDEF=DF          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 Size	Precision	Est_ SigmaSq	Est_ Sigma	Est_ VarObs	Est_ StdObs
30	0.667	20.678	4.547	31.017	5.569
20	0.667	20.678	4.547	31.017	5.569
30	0.667	20.678	4.547	31.017	5.569
25	0.667	20.678	4.547	31.017	5.569

Distance=3

Object Size	Precision	Est_ SigmaSq	Est_ Sigma	Est_ VarObs	Est_ StdObs
43	0.333	20.678	4.547	62.035	7.876
33	0.333	20.678	4.547	62.035	7.876
25	0.333	20.678	4.547	62.035	7.876
30	0.333	20.678	4.547	62.035	7.876

Distance=4.5

Object Size	Precision	Est_ SigmaSq	Est_ Sigma	Est_ VarObs	Est_ StdObs
25	0.222	20.678	4.547	93.052	9.646
36	0.222	20.678	4.547	93.052	9.646
48	0.222	20.678	4.547	93.052	9.646
33	0.222	20.678	4.547	93.052	9.646

Distance=6

Object Size	Precision	Est_ SigmaSq	Est_ Sigma	Est_ VarObs	Est_ StdObs
43	0.167	20.678	4.547	124.07	11.139
36	0.167	20.678	4.547	124.07	11.139
23	0.167	20.678	4.547	124.07	11.139
48	0.167	20.678	4.547	124.07	11.139

Distance=7.5

Object Size	Precision	Est_ SigmaSq	Est_ Sigma	Est_ VarObs	Est_ StdObs
30	0.133	20.678	4.547	155.09	12.453
25	0.133	20.678	4.547	155.09	12.453
50	0.133	20.678	4.547	155.09	12.453
38	0.133	20.678	4.547	155.09	12.453

WHERE

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 the *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;
```

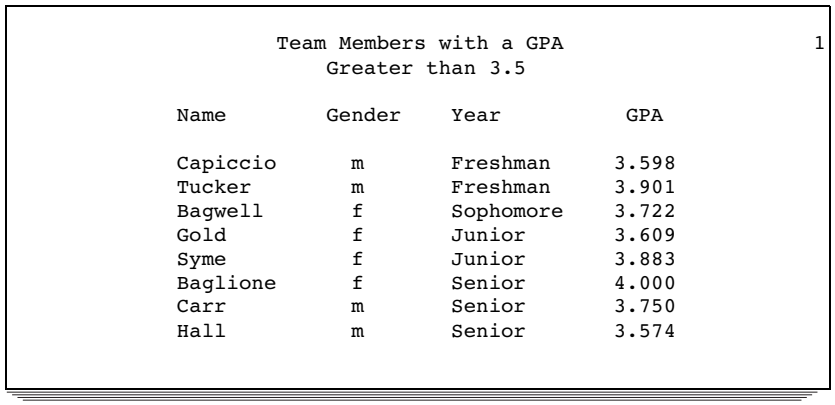
For more information on WHERE=, see the *SAS Language Reference: Dictionary*.

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 30.

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


proc print data=debate noobs;
   where gpa>3.5;
   title 'Team Members with a GPA';
```

```
   title2 'Greater than 3.5';
run;
```



Team Members with a GPA 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

CHAPTER

4

In-Database Processing of Base Procedures

Base Procedures That Are Enhanced for In-Database Processing **49**

Base Procedures That Are Enhanced for In-Database Processing

In the third maintenance release for SAS 9.2, Base SAS procedures have been enhanced so they can process data inside the Teradata Enterprise Data Warehouse (EDW), DB2 under UNIX, and Oracle. The in-database procedures are used to generate more sophisticated queries that allow the aggregations and analytics to be run inside the database. These in-database procedures all generate SQL queries. You use SAS/ACCESS or SQL as the interface to the Teradata EDW.

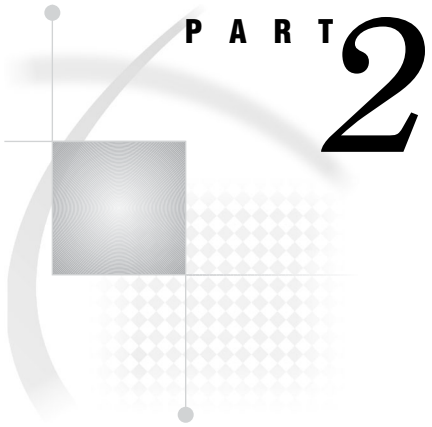
The following Base SAS procedures have been enhanced for in-database processing.

Table 4.1 In-Database Base Procedures

Procedure	Description
PROC FREQ in *Base SAS Procedures Guide: Statistical Procedures*	produces one-way to *n*-way tables; reports frequency counts; computes test and measures of association and agreement for two-way to *n*-way crosstabulation tables; can compute exact tests and asymptotic tests; can create output data sets.
PROC MEANSChapter 33, “The MEANS Procedure,” on page 609	computes descriptive statistics; can produce printed output and output data sets. By default, PROC MEANS produces printed output.
PROC RANKChapter 49, “The RANK Procedure,” on page 943	computes ranks for one or more numeric variables across the observations of a SAS data set; can produce some rank scores.
PROC REPORTChapter 51, “The REPORT Procedure,” on page 979	combines features of the PRINT, MEANS, and TABULATE procedures with features of the DATA step in a single report-writing tool that can produce a variety of reports.
PROC SORTChapter 54, “The SORT Procedure,” on page 1165	orders SAS data set observations by the values of one or more character or numeric variables.

Procedure	Description
PROC SUMMARYChapter 57, "The SUMMARY Procedure," on page 1351	computes descriptive statistics; can produce a printed report and create an output data set. By default, PROC SUMMARY creates an output data set.
PROC TABULATEChapter 58, "The TABULATE Procedure," on page 1355	displays descriptive statistics in tabular format, using some or all of the variables in a data set.

For more information, see the documentation for each procedure and "Overview of In-Database Procedures" in *SAS/ACCESS for Relational Databases: Reference*.



PART 2

Procedures

Chapter ***5*** . . . . . **The APPEND Procedure** *55*

Chapter ***6*** . . . . . **The CALENDAR Procedure** *57*

Chapter ***7*** . . . . . **The CALLRFC Procedure** *129*

Chapter ***8*** . . . . . **The CATALOG Procedure** *131*

Chapter ***9*** . . . . . **The CHART Procedure** *155*

Chapter ***10*** . . . . . **The CIMPORT Procedure** *191*

Chapter ***11*** . . . . . **The COMPARE Procedure** *207*

Chapter ***12*** . . . . . **The CONTENTS Procedure** *259*

Chapter ***13*** . . . . . **The COPY Procedure** *261*

Chapter ***14*** . . . . . **The CORR Procedure** *267*

Chapter ***15*** . . . . . **The CPORT Procedure** *269*

Chapter ***16*** . . . . . **The CV2VIEW Procedure** *285*

Chapter ***17*** . . . . . **The DATASETS Procedure** *287*

Chapter ***18*** . . . . . **The DBCSTAB Procedure** *399*

Chapter ***19*** . . . . . **The DISPLAY Procedure** *401*

Chapter ***20*** . . . . . **The DOCUMENT Procedure** *405*

Chapter ***21*** . . . . . **The EXPLODE Procedure** *407*

Chapter ***22*** . . . . . **The EXPORT Procedure** *409*

Chapter ***23*** . . . . . **The FCMP Procedure** *417*

Chapter ***24*** . . . . . **The FONTREG Procedure** *497*

Chapter ***25*** . . . . . **The FORMAT Procedure** *511*

Chapter ***26*** . . . . . **The FORMS Procedure** *573*

Chapter ***27*** . . . . . **The FREQ Procedure** *575*

Chapter ***28*** . . . . . **The FSLIST Procedure** *577*

Chapter ***29*** . . . . . **The HTTP Procedure** *589*

Chapter ***30*** . . . . . **The IMPORT Procedure** *595*

Chapter ***31*** . . . . . **The INFOMAPS Procedure** *605*

Chapter ***32*** . . . . . **The JAVAINFO Procedure** *607*

Chapter ***33*** . . . . . **The MEANS Procedure** *609*

Chapter ***34*** . . . . . **The METADATA Procedure** *677*

Chapter ***35*** . . . . . **The METALIB Procedure** *679*

Chapter ***36*** . . . . . **The METAOPERATE Procedure** *681*

Chapter ***37*** . . . . . **The MIGRATE Procedure** *683*

Chapter ***38*** . . . . . **The OPTIONS Procedure** *701*

Chapter ***39*** . . . . . **The OPTLOAD Procedure** *715*

Chapter ***40*** . . . . . **The OPTSAVE Procedure** *717*

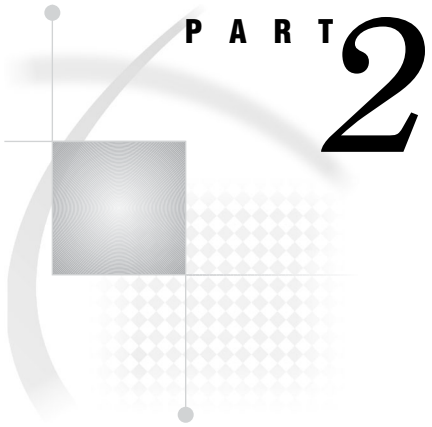
Chapter ***41*** . . . . . **The PLOT Procedure** *719*

Chapter ***42*** . . . . . **The PMENU Procedure** *777*

Chapter ***43*** . . . . . **The PRINT Procedure** *815*

Chapter ***44*** . . . . . **The PRINTTO Procedure** *887*

Chapter ***45*** . . . . . **The PROTO Procedure** *903*



PART 2

Procedures

Chapter ***46*** . . . . . **The PRTDEF Procedure** *921*

Chapter ***47*** . . . . . **The PRTEXP Procedure** *933*

Chapter ***48*** . . . . . **The PWENCODE Procedure** *937*

Chapter ***49*** . . . . . **The RANK Procedure** *943*

Chapter ***50*** . . . . . **The REGISTRY Procedure** *963*

Chapter ***51*** . . . . . **The REPORT Procedure** *979*

Chapter ***52*** . . . . . **The SCAPROC Procedure** *1143*

Chapter ***53*** . . . . . **The SOAP Procedure** *1153*

Chapter ***54*** . . . . . **The SORT Procedure** *1165*

Chapter ***55*** . . . . . **The SQL Procedure** *1197*

Chapter ***56*** . . . . . **The STANDARD Procedure** *1335*

Chapter ***57*** . . . . . **The SUMMARY Procedure** *1351*

Chapter ***58*** . . . . . **The TABULATE Procedure** *1355*

Chapter ***59*** . . . . . **The TEMPLATE Procedure** *1475*

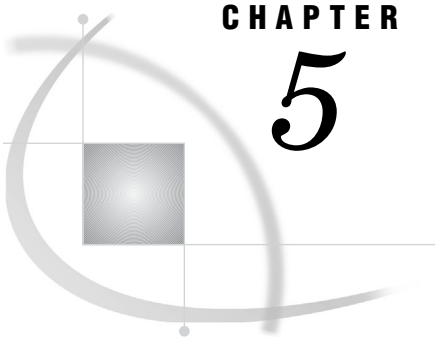
Chapter ***60*** . . . . . **The TIMEPLOT Procedure** *1477*

Chapter ***61*** . . . . . **The TRANSPOSE Procedure** *1501*

Chapter ***62*** . . . . . **The TRANTAB Procedure** *1525*

Chapter ***63*** . . . . . **The UNIVARIATE Procedure** *1527*

Chapter ***64*** . . . . . **The XSL Procedure (Preproduction)** *1529*



CHAPTER

5

The APPEND Procedure

Overview: APPEND Procedure **55**
Syntax: APPEND Procedure **55**

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

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

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

Tip: Complete documentation for the APPEND statement and the APPEND procedure is in “APPEND Statement” on page 302 .

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

Note: The links in the following table are to the DATASETS procedure documentation, which explains these options.
△

Task	Option
Add observations from one SAS data set to the end of another SAS data set	"APPEND Statement" on page 302
Add observations to the data set one at a time	APPENDVER=V6 on page 303
Name of destination data set	BASE= on page 302 (required)
Name of source data set	DATA= on page 303
Forces the append when variables are different	FORCE on page 303
Copies the sort indicator that was established by using PROC SORT from the DATA= data set to the BASE= data set	GETSORT on page 303
Suppresses the warning message when used with the FORCE option to concatenate two data sets with different variables	NOWARN on page 304

CHAPTER

6

The CALENDAR Procedure

Overview: CALENDAR Procedure **59**
What Does the CALENDAR Procedure Do? **59**
What Types of Calendars Can PROC CALENDAR Produce? **59**
Advanced Scheduling and Project Management Tasks **63**
Syntax: CALENDAR Procedure **64**
PROC CALENDAR Statement **65**
BY Statement **72**
CALID Statement **73**
DUR Statement **74**
FIN Statement **75**
HOLIDUR Statement **75**
HOLIFIN Statement **76**
HOLISTART Statement **76**
HOLIVAR Statement **77**
MEAN Statement **78**
OUTDUR Statement **78**
OUTFIN Statement **79**
OUTSTART Statement **79**
START Statement **80**
SUM Statement **80**
VAR Statement **81**
Concepts: CALENDAR Procedure **82**
Type of Calendars **82**
Schedule Calendar **82**
Definition **82**
Required Statements **82**
Examples **83**
Summary Calendar **83**
Definition **83**
Required Statements **83**
Multiple Events on a Single Day **83**
Examples **83**
The Default Calendars **83**
Description **83**
When You Unexpectedly Produce a Default Calendar **84**
Examples **84**
Calendars and Multiple Calendars **84**
Definitions **84**
Why Create Multiple Calendars **84**
How to Identify Multiple Calendars **85**
Using Holidays or Calendar Data Sets with Multiple Calendars **85**

Types of Reports That Contain Multiple Calendars **85**
*How to Identify Calendars with the CALID Statement and the Special Variable _CAL_* **86**
When You Use Holidays or Calendar Data Sets **86**
Examples **86**
Input Data Sets **86**
Activities Data Set **87**
Purpose **87**
Requirements and Restrictions **87**
Structure **87**
Multiple Activities per Day in Summary Calendars **88**
Examples **88**
Holidays Data Set **88**
Purpose **88**
Structure **88**
No Sorting Needed **88**
Using SAS Date Versus SAS Datetime Values **88**
Create a Generic Holidays Data Set **89**
Holidays and Nonwork Periods **89**
Examples **89**
Calendar Data Set **89**
Purpose **89**
Structure **89**
Using Default Work Shifts Instead of a Workdays Data Set **90**
Examples **90**
Workdays Data Set **90**
Purpose **90**
Use Default Work Shifts or Create Your Own? **91**
Structure **91**
How Missing Values Are Treated **91**
Examples **91**
Missing Values in Input Data Sets **91**
Results: CALENDAR Procedure **92**
What Affects the Quantity of PROC CALENDAR Output **92**
How Size Affects the Format of PROC CALENDAR Output **92**
What Affects the Lines That Show Activity Duration **93**
Customizing the Calendar Appearance **93**
Portability of ODS Output with PROC CALENDAR **93**
Examples: CALENDAR Procedure **93**
Example 1: Schedule Calendar with Holidays: 5-Day Default **93**
Example 2: Schedule Calendar Containing Multiple Calendars **97**
Example 3: Multiple Schedule Calendars with Atypical Work Shifts (Separated Output) **100**
Example 4: Multiple Schedule Calendars with Atypical Work Shifts (Combined and Mixed Output) **105**
Example 5: Schedule Calendar, Blank or with Holidays **110**
Example 6: Calculating a Schedule Based on Completion of Predecessor Tasks **114**
Example 7: Summary Calendar with MEAN Values By Observation **119**
Example 8: Multiple Summary Calendars with Atypical Work Shifts (Separated Output) **123**

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

The following output 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 6.1.

```
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 93.

Output 6.1 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

The following output 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 105.

Output 6.2 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

The following output 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 the following output. The data set that is shown in this calendar is created in Example 7 on page 119.

Output 6.3 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 114.

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: Basic Concepts in *SAS Output Delivery System: User's Guide* for details.

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

Table of Contents:

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.

Task	Statement
Create summary calendar	"MEAN Statement" on page 78
	"SUM Statement" on page 80
Create schedule calendar	"DUR Statement" on page 74
	"FIN Statement" on page 75
Create multiple calendars	"CALID Statement" on page 73

Task	Statement
Specify holidays	"HOLISTART Statement" on page 76
	"HOLIDUR Statement" on page 75
	"HOLIFIN Statement" on page 76
	"HOLIVAR Statement" on page 77
Control display	"OUTSTART Statement" on page 79
	"OUTDUR Statement" on page 78
	"OUTFIN Statement" on page 79
Specify grouping	"BY Statement" on page 72
	"CALID Statement" on page 73

PROC CALENDAR Statement

PROC CALENDAR <*option(s)*>;

Task	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	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	

Task	Option
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 83.

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 89

Featured in: Example 3 on page 100

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 87

Featured in: All examples. See "Examples: CALENDAR Procedure" on page 93

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 100

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 work week of five half-days. In a work week with varying day lengths, you need to set a standard day length to use in calculating duration times. For example, an activity with a duration of 3.0 workdays lasts 24 hours if DAYLENGTH=8:00 or 30 hours if DAYLENGTH=10:00.

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 89 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 110

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 6.1 on page 68 shows the formatting characters that PROC CALENDAR uses. Figure 6.1 on page 69 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 example, the following option assigns an asterisk (*) to the 12th position, assigns a single dash (-) to the 13th, and does not alter remaining characters:

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

These new settings change the activity line from this:

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

to this:

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

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

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

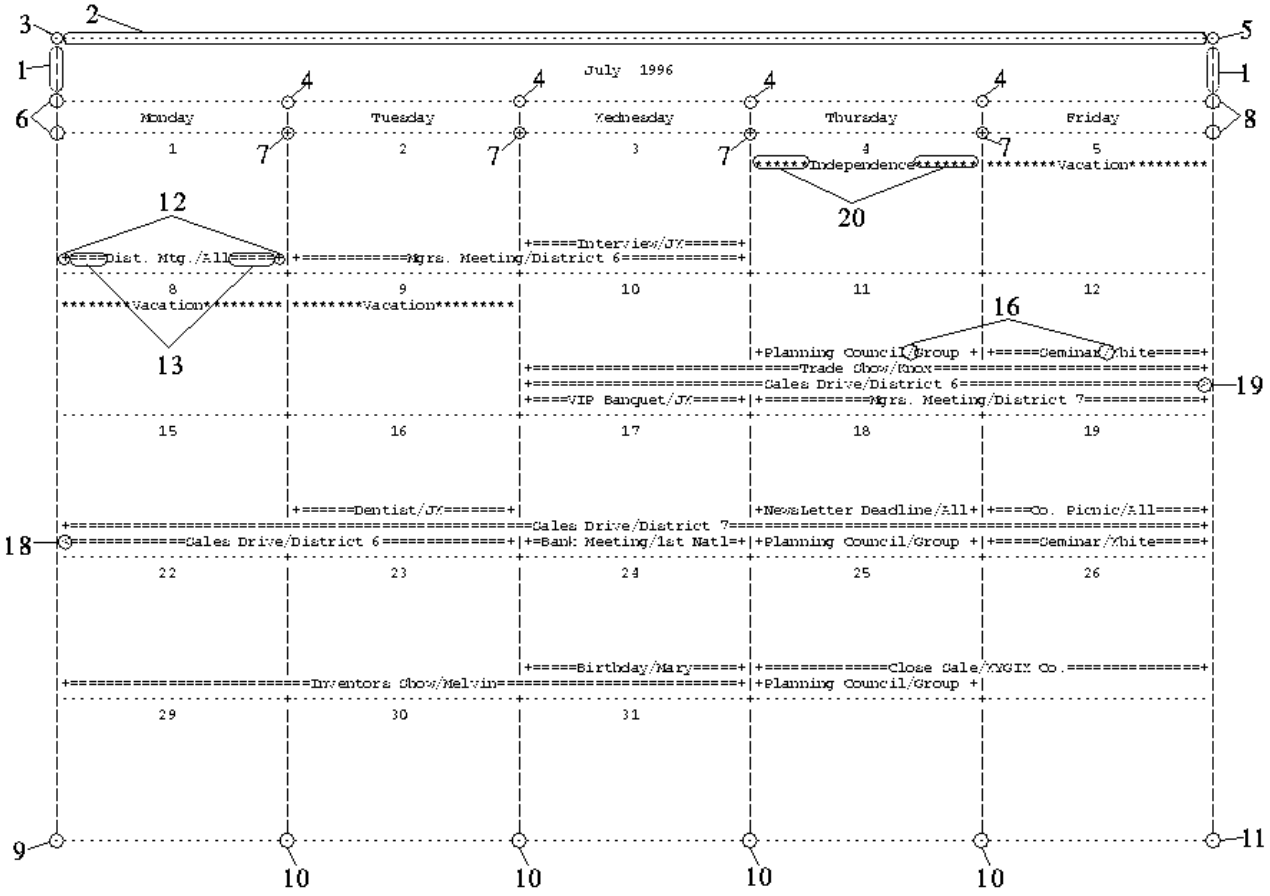
```
formchar(3,7)='2D7C'x
```

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

Table 6.1 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
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 6.1 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 88

Featured in: All examples. See "Examples: CALENDAR Procedure" on page 93

INTERVAL=DAY | WORKDAY
specifies the units of the DUR and HOLIDUR variables to one of two default daylengths:

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

WORKDAY
specifies the values of the DUR and HOLIDUR variables in units of 8-hour days 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 eight hours long, then a DUR value of 3.0 is treated as 24 hours.

Default: DAY

Interaction: If there is no 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 84 and "Calendar Data Set" on page 89 for more information on the INTERVAL= option and the specification of working days; "The Default Calendars" on page 83

Featured in: Example 5 on page 110

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 123

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

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

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

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

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

Default: NOBS

Restriction: MEANTYPE= applies only to summary calendars.

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

Featured in: Example 7 on page 119

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 91 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 6.1 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 93

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

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

See also: "Workdays Data Set" on page 90 and "Calendar Data Set" on page 89

Featured in: Example 3 on page 100

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 36

See also: "CALID Statement" on page 73

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 89

Featured in: Example 2 on page 97, Example 3 on page 100, and Example 6 on page 114

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 89

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 97 and Example 4 on page 105

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 105

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 100 and Example 8 on page 123

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 93)

DUR *variable*;

Required Arguments

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

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

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

See also: For more information on activity durations, see "Activities Data Set" on page 87 and "Calendar Data Set" on page 89

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:

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

- You can override the default length of a duration unit by using
  - 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 114

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 93 through Example 5 on page 110

HOLIDUR *variable*;

Required Arguments

variable
contains the duration of each holiday.

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

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

Featured in: Example 3 on page 100 and Example 8 on page 123

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 93 through Example 5 on page 110

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 93 through Example 5 on page 110

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 119

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 119

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 100 and Example 8 on page 123

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 100 and Example 8 on page 123

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 119 and Example 8 on page 123

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 119 and Example 8 on page 123

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.

Type of Calendar	Task	Restriction
schedule calendar	schedule activities around holidays and nonwork periods	cannot calculate sums and means
schedule calendar	schedule activities that last more than one day	
summary calendar	calculate sums and means	activities can last only one day

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

Schedule Calendar

Definition

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

Required Statements

You must supply a START statement and either a DUR or FIN statement.

Statement	Variable Value
START	starting date of an activity
DUR*	duration of an activity
FIN*	ending date of an activity

* Choose one of the following statements. 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 59, "Advanced Schedule Calendar" on page 60, as well as Example 1 on page 93, Example 2 on page 97, Example 3 on page 100, Example 4 on page 105, Example 5 on page 110, and Example 6 on page 114

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 61, Example 7 on page 119, and Example 8 on page 123

The Default Calendars

Description

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

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

Table 6.2 Default Calendar Settings and Examples

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

When You Unexpectedly Produce a Default Calendar

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

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

Examples

See the 7-day default calendar in Output 6.1 and the 5-day default calendar in Example 1 on page 93

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 114.

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 97, Example 3 on page 100, Example 4 on page 105, and Example 8 on page 123

Input Data Sets

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

Table 6.3 Four Possible Input Data Sets for PROC CALENDAR

Data Set	Description	Option
activities	Each *observation* contains information about a single activity.	DATA=
holidays	Each *observation* contains information about a holiday	HOLIDATA=
calendar	Each *observation* defines one weekly work schedule.	CALEDATA=
workdays	Each *variable* represents one daily schedule of alternating work and nonwork periods.	WORKDATA=

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.

Variable Content	Statement	Calendar Type
starting date	START	Schedule
		Summary
duration	DUR	Schedule
finishing date	FIN	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.

Variable Content	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 work shifts. Valid values for work shifts are

  - **WORKDAY** (the default work shift)

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

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

If the CALID variable contains a missing value, then the character or numeric value for the default calendar (**DEFAULT** or 0) is used. See "The Default Calendars" on page 83 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 eight hours if INTERVAL=WORKDAY.

Using Default Work Shifts Instead of a Workdays Data Set

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

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

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

Examples

Example 3 on page 100, Example 4 on page 105, and Example 7 on page 119 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 work shifts named in a CALEDATA= data set.

Use Default Work Shifts or Create Your Own?

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

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

See the INTERVAL= option on page 70.

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 100

Missing Values in Input Data Sets

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

Table 6.4 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

Data set	Variable	Treatment of Missing Values
	FIN	START value + daylength is used
	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 the length by specifying

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

Customizing the Calendar Appearance

PROC CALENDAR uses 17 of the 20 SAS formatting characters to construct the outline of the calendar and to print activity lines and to indicate holidays. You can use the FORMCHAR= option to customize the appearance of your PROC CALENDAR output by substituting your own characters for the default. See Table 6.1 on page 68 and Figure 6.1 on page 69.

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 67.

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: Listing

Output 6.4 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 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 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: Listing

Output 6.5 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 Work Shifts (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 105 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-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 work shifts for an entire week. The _CAL_ variable identifies to which calendar the work shifts apply. CAL1 uses the default 8-hour work shifts for Monday through Friday. CAL2 uses a half day on Saturday and the default 8-hour work shift for Monday through Friday.

```
data well.cal;
   input _sun_ $ _sat_ $ _mon_ $ _tue_ $ _wed_ $ _thu_ $
         _fri_ $ _cal_ $;
   datalines;
Holiday Holiday  Workday Workday Workday Workday Workday CAL1
Holiday Halfday  Workday Workday Workday Workday Workday CAL2
;
```

Create the workdays data set. This data set defines the daily work shifts that are named in the calendar data set. Each variable (not observation) contains one daily schedule of alternating work and nonwork periods. The HALFDAY work shift lasts 4 hours.

```
data well.wor;
   input halfday time5.;
   datalines;
08:00
12:00
;
```

Sort the activities data set by the variables that contain the calendar identification and the starting date, respectively. You are not required to sort the holidays data set.

```
proc sort data=well.act;
   by _cal_ date;
```

```
run;
```

Set 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; 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;
```

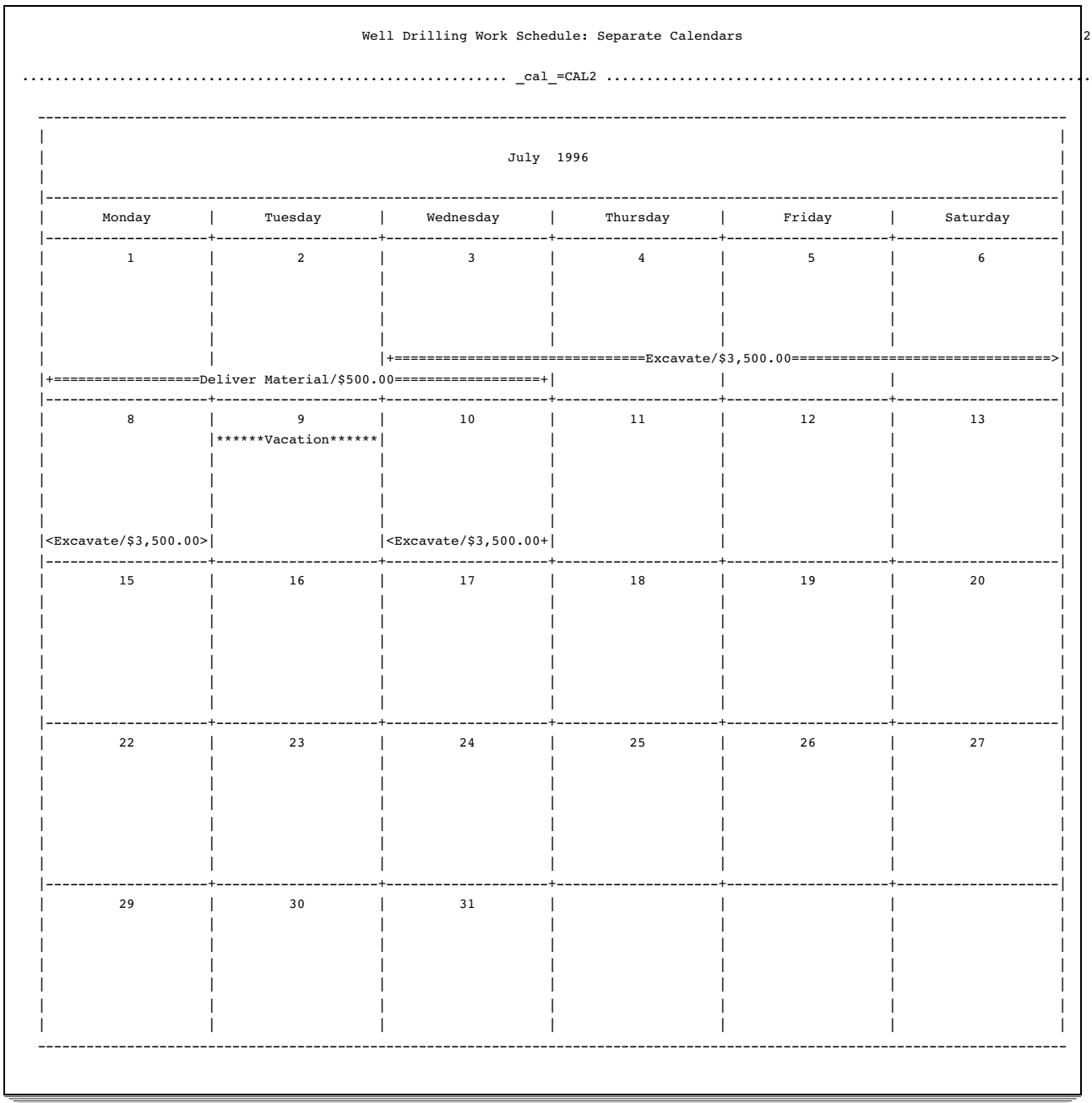
Specify the title and format the Cost variable.

```
    title1 'Well Drilling Work Schedule: Separate Calendars';
    format cost dollar9.2;
run;
```

Output: Listing

Output 6.6 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 Work Shifts (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 101, WELL.HOL on page 102, WELL.CAL on page 102, WEL.WOR on page 102.

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 100 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 library where the activities data set is stored.

```
libname well 'SAS-library';
```

Sort the activities data set by the variable that contains the starting date. Do not sort by the CALID variable when producing combined calendar output.

```
proc sort data=well.act;
   by date;
run;
```

Set 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 6.7 Multiple Schedule Calendars with Atypical Work Shifts (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 6.8 Multiple Schedule Calendar with Atypical Work Shifts (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: Listing

Output 6.9 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 ***New Year's****	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	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 *Christmas Break*	26 *Christmas Break*	27
28	29 *Christmas Break*	30 *Christmas Break*	31 *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 work shifts with holidays, calendar, and work-shift data sets.)
2. indicate which activities are successors to others (precedence relationships).
3. define resource limitations *if* you want them considered in the schedule.
4. provide an initial starting date.

PROC CPM can process your data to generate a data set that contains the start and end dates for each activity. PROC CPM schedules the activities, based on the duration information, weekly work patterns, work shifts, as well as holidays and nonwork days that interrupt the schedule. You can generate several views of the schedule that is computed by PROC CPM, from a simple listing of start and finish dates to a calendar, a Gantt chart, or a network diagram.

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 line size is not long enough to print the variable values, then PROC CALENDAR either truncates the values or produces no calendar output. A longer line size 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 jobs are milestones.

```
data grant;
   input jobnum Task $ 4-22 Days Succ1 $ 27-45 aldate : date7. altype $
         _cal_ $;
   format aldate date7.;
   datalines;
1  Run Exp 1          11  Analyze Exp 1       .      .   Student
2  Analyze Exp 1       5  Send Report 1       .      .   Prof.
3  Send Report 1       0  Run Exp 2           .      .   Prof.
```

```
4  Run Exp 2          11  Analyze Exp 2       .       .   Student
5  Analyze Exp 2       4  Send Report 2       .       .   Prof.
6  Send Report 2       0  Write Final Report  .       .   Prof.
7  Write Final Report  4  Send Final Report   .       .   Prof.
8  Send Final Report   0                      .       .   Student
9  Site Visit          1                    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 6.10.

```
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 6.11.

```
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: Listing

Output 6.10 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

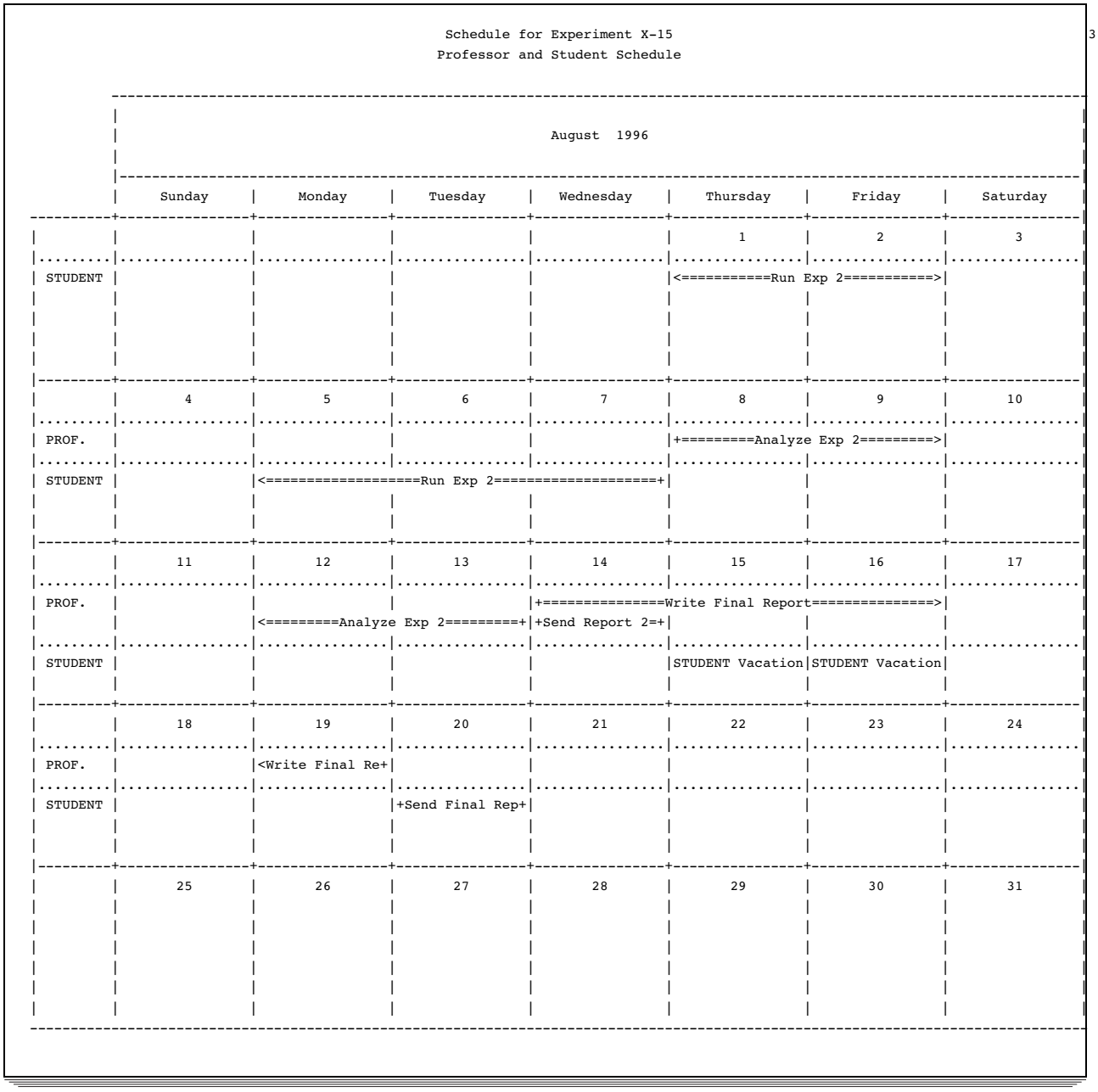
Output 6.11 Schedule Calendar Based on Output from PROC CPM

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 24 and August 14, 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: Listing

Output 6.12 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 Work Shifts (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 101 and WELL.HOL on page 102.

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 library where the activities data set is stored.

```
libname well 'SAS-library';
run;
```

Sort the activities data set by the variables containing the calendar identification and the starting date, respectively.

```
proc sort data=well.act;
   by _cal_ date;
run;
```

Set 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: Listing

Output 6.13 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 |
                                    -------------------------
```

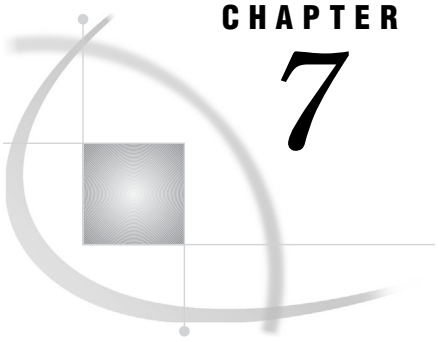
Well Drilling Cost Summary 2
Separate Calendars

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

July 1996

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

Legend	Sum
task	
dur	
cost	$4,000.00



CHAPTER 7

The CALLRFC Procedure

Information about the CALLRFC Procedure **129**

Information about the CALLRFC Procedure

See: For documentation about the CALLRCF procedure, see *SAS/ACCESS Interface to R/3: User's Guide*.

CHAPTER 8

The CATALOG Procedure

Overview: CATALOG Procedure **131**
Syntax: CATALOG Procedure **132**
PROC CATALOG Statement **133**
CHANGE Statement **135**
CONTENTS Statement **135**
COPY Statement **136**
DELETE Statement **138**
EXCHANGE Statement **139**
EXCLUDE Statement **140**
MODIFY Statement **140**
SAVE Statement **141**
SELECT Statement **142**
Concepts: CATALOG Procedure **142**
Interactive Processing with RUN Groups **142**
Definition **143**
How to End a PROC CATALOG Step **143**
Error Handling and RUN Groups **143**
Specifying an Entry Type **143**
Four Ways to Supply an Entry Type **144**
Why Use the ENTRYTYPE= Option? **144**
Avoid a Common Error **144**
The ENTRYTYPE= Option **144**
Catalog Concatenation **145**
Restrictions **145**
Results: CATALOG Procedure **146**
Examples: CATALOG Procedure **147**
Example 1: Copying, Deleting, and Moving Catalog Entries from Multiple Catalogs **147**
Example 2: Displaying Contents, Changing Names, and Changing a Description **151**
Example 3: Using the FORCE Option with the KILL Option **153**

Overview: CATALOG Procedure

The CATALOG procedure manages entries in SAS catalogs. PROC CATALOG is an interactive, statement-driven procedure that enables you to do the following:

- create a listing of the contents of a catalog
- copy a catalog or selected entries within a catalog
- rename, exchange, or delete entries within a catalog
- change the name of a catalog entry
- modify, by changing or deleting, the description of a catalog entry

For more information on SAS libraries and catalogs, refer to *SAS Language Reference: Concepts*.

To learn how to use the SAS windowing environment to manage entries in a SAS catalog, see the SAS online Help for the SAS Explorer window. You might prefer to use the Explorer window instead of using PROC CATALOG. The window can do most of what the procedure does.

Syntax: CATALOG Procedure

Tip: Supports RUN-group processing.

Tip: 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 55, "The SQL Procedure," on page 1197.

See: CATALOG Procedure OpenVMS 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-1* <*...entry-n*> </ ENTRYTYPE=*etype*>;
EXCLUDE *entry-1* <*...entry-n*> </ ENTRYTYPE=*etype*>;

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

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

DELETE *entry-1* <*...entry-n*> </ ENTRYTYPE=*etype*>;

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

SAVE *entry-1* <*...entry-n*> </ ENTRYTYPE=*etype*>;

Task	Statement
Copy entries from one SAS catalog to another	"PROC CATALOG Statement" on page 133
Copy or move all entries	"COPY Statement" on page 136 (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 Statement" on page 142
Copy all *except* the entries specified	COPY, "EXCLUDE Statement" on page 140

Task	Statement
Delete entries from a SAS catalog	
Delete *all* entries	"PROC CATALOG Statement" on page 133 (with KILL option)
Delete *all* entries in catalog opened by another resource environment	PROC CATALOG (with FORCE and KILL options)
Delete specified entries	"DELETE Statement" on page 138
Delete all *except* the entries specified	"SAVE Statement" on page 141
Alter names and descriptions	
Change the names of catalog entries	"CHANGE Statement" on page 135
Switch the names of two catalog entries	"EXCHANGE Statement" on page 139
Change the description of a catalog entry	"MODIFY Statement" on page 140
Print	
Print the contents of a catalog	"CONTENTS Statement" on page 135

PROC CATALOG Statement

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

Task	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 143

Featured in:

Example 1 on page 147

Example 2 on page 151

FORCE
forces statements to execute on a catalog that is opened by another resource environment.

Some CATALOG statements require exclusive access to the catalog that they operate on if the statement can radically change the contents of a catalog. If exclusive access cannot be obtained, then the action fails. Here are the statements and the catalogs that are affected by FORCE:

KILL	affects the specified catalog
COPY	affects the OUT= catalog
COPY MOVE	affects the IN= and the OUT= catalogs
SAVE	affects the specified catalog

Tip: Use FORCE to execute the statement, even if exclusive access cannot be obtained.

Featured in: Example 3 on page 153

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 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 153

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 151

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 144

See also: "Specifying an Entry Type" on page 143

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 151

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 the following order:

LIBNAME	the libref
MEMNAME	the catalog name
NAME	the names of entries
TYPE	the types of entries
DESC	the descriptions of entries
DATE	the dates entries were last modified.

COPY Statement

Copies some or all of the entries in one catalog to another catalog.

Restriction: A COPY statement's effect ends at a RUN statement or at the beginning of a statement other than the SELECT or EXCLUDE statement.

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 147

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

Task	Option
Restrict processing to one type of entry	ENTRYTYPE=
Copy from a different catalog in the same step	IN=
Enables concurrent users to copy to the same catalog at the same time	LOCKCAT=
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 144

See also: "Specifying an Entry Type" on page 143

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 147

LOCKCAT=EXCLUSIVE | SHARE
specifies whether to enable more than one user to copy to the same catalog at the same time. Using LOCKCAT=SHARE locks individual entries rather than the entire catalog, which enables greater throughput. The default is LOCKCAT=EXCLUSIVE, which locks the entire catalog to one user.

Note: Using the LOCKCAT=SHARE option can lessen performance if used in a single-user environment because of the overhead associated with locking and unlocking each entry. △

MOVE
deletes the original catalog or entries after the new copy is made.

Interaction: When MOVE removes all entries from a catalog, the procedure deletes the catalog from the library.

NEW

overwrites the destination (specified by OUT=) if it already exists. If you omit NEW, PROC CATALOG updates the destination. For information about using the NEW option with concatenated catalogs, see "Catalog Concatenation" on page 145.

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 147

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 141

Featured in: Example 1 on page 147

DELETE *entry-1* <*...entry-n*> </ ENTRYTYPE=*etype*>;

Required Arguments

entry-1 <...entry-n>
specifies the name of one or more SAS catalog entries.

Restriction: You must designate the type of the entry, either with the name (*ename.etype*) or with the ENTRYTYPE= option.

Options

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

See: "The ENTRYTYPE= Option" on page 144

See also: "Specifying an Entry Type" on page 143

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 switches.

Interaction: You can specify only the entry name without the entry type if you use the ENTRYTYPE= option on either the PROC CATALOG statement or the EXCHANGE statement.

See also: "Specifying an Entry Type" on page 143

Options

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

See: "The ENTRYTYPE= Option" on page 144

See also: "Specifying an Entry Type" on page 143

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 136 and "SELECT Statement" on page 142

Featured in: Example 1 on page 147

EXCLUDE *entry-1* <*...entry-n*> </ ENTRYTYPE=*etype*>;

Required Arguments

entry-1 <...entry-n>
specifies the name of one or more SAS catalog entries.

Restriction: You must designate the type of the entry, either when you specify the name (*ename.etype*) or with the ENTRYTYPE= option.

See also: "Specifying an Entry Type" on page 143

Options

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

See: "The ENTRYTYPE= Option" on page 144

See also: "Specifying an Entry Type" on page 143

MODIFY Statement

Changes the description of a catalog entry.

Featured in: Example 2 on page 151

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

Required Arguments

entry
specifies the name of one SAS catalog entry. You can specify the entry type with the name.

Restriction: You must designate the type of the entry, either when you specify the name (*ename.etype*) or with the ENTRYTYPE= option.

See also: "Specifying an Entry Type" on page 143

DESCRIPTION=<<'>*entry-description*<'>>
changes the description of a catalog entry by replacing it with a new description, up to 256 characters long, or by removing it altogether. You can enclose the description in single or double quotes.

Alias: DESC

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

Options

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

See: "The ENTRYTYPE= Option" on page 144

See also: "Specifying an Entry Type" on page 143

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 138

SAVE *entry-1* <*...entry-n*> </ ENTRYTYPE=*etype*>;

Required Arguments

entry <...entry-n>
specifies the name of one or more SAS catalog entries.

Restriction: You must designate the type of the entry, either with the name (*ename.etype*) or with the ENTRYTYPE= option.

Options

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

See: "The ENTRYTYPE= Option" on page 144

See also: "Specifying an Entry Type" on page 143

SELECT Statement

Specifies entries that the COPY statement copies.

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 136 and "EXCLUDE Statement" on page 140

Featured in: Example 1 on page 147

SELECT *entry-1 <...entry-n.* </ ENTRYTYPE=*etype*>;

Required Arguments

entry-1 <...entry-n>
specifies the name of one or more SAS catalog entries.

Restriction: You must designate the type of the entry, either when you specify the name (*ename.etype*) or with the ENTRYTYPE= option.

Options

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

See: "The ENTRYTYPE= Option" on page 144.

See also: "Specifying an Entry Type" on page 143.

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. The following list contains ways to terminate a PROC CATALOG step:

- submit a QUIT statement
- submit a RUN statement with the CANCEL option
- submit another DATA or PROC statement
- end your SAS session

Note: When you enter a QUIT, DATA, or PROC statement, any statements following the last RUN group execute before the CATALOG procedure terminates. If you enter a RUN statement with the CANCEL option, however, the remaining statements *do not execute* before the procedure ends. △

See Example 2 on page 151.

Error Handling and RUN Groups

Error handling is based in part on the division of statements into RUN groups. If a syntax error is encountered, *none* of the statements in the current RUN group execute, and execution proceeds to the next RUN group.

For example, the following statements contain a misspelled DELETE statement:

```
proc catalog catalog=misc entrytype=help;
   copy out=drink;
      select coffee tea;
   del juices;        /* INCORRECT!!! */
   exchange glass=plastic;
run;
   change calstats=nutri;
run;
```

Because the DELETE statement is incorrectly specified as DEL, no statements in that RUN group execute, *except* the PROC CATALOG statement itself. The CHANGE statement does execute, however, because it is in a different RUN group.

CAUTION:
Be careful when setting up batch jobs in which one RUN group's statements depend on the effects of a previous RUN group, especially when deleting and renaming entries. △

Specifying an Entry Type

Four Ways to Supply an Entry Type

There is no default entry type, so if you do not supply one, PROC CATALOG generates an error. You can supply an entry type in one of four ways, as shown in the following table:

Table 8.1 Supplying an Entry Type

Entry Type	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.

You can have many entries of one type and a few of other types. You can use ENTRYTYPE= to specify a default and then override that for individual entries with (ENTRYTYPE=) *in parentheses* after those entries.

Avoid a Common Error

You cannot specify the ENTRYTYPE= option in both the PROC CATALOG statement and a subordinate statement. For example, these statements generate an error and do not delete any entries because the ENTRYTYPE= specifications contradict each other:

```
/* THIS IS INCORRECT CODE. */
proc catalog cat=sample et=help;
   delete a b c / et=program;
run;
```

The ENTRYTYPE= Option

The ENTRYTYPE= option is available in every statement in the CATALOG procedure except CONTENTS.

ENTRYTYPE=*etype*
not in parentheses, sets a default entry type for the entire PROC step when used in the PROC CATALOG statement. In all other statements, this option sets a default entry type for the *current* statement.

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 143.

Featured in: Example 1 on page 147

(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 as shown in the following example:

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

See also: "Specifying an Entry Type" on page 143

Featured in: Example 1 on page 147 and Example 2 on page 151

Catalog Concatenation

There are two types of CATALOG concatenation. The first is specified by the LIBNAME statement and the second is specified by the global CATNAME statement. All statements and options that can be used on single (unconcatenated) catalogs can be used on catalog concatenations.

Restrictions

When you use the CATALOG procedure to copy concatenated catalogs and you use the NEW option, the following rules apply:

1 If the input catalog is a concatenation and if the output catalog exists in any level of the input concatenation, the copy is not allowed.
2 If the output catalog is a concatenation and if the input catalog exists in the first level of the output concatenation, the copy is not allowed.

For example, the following code demonstrates these two rules, and the copy fails:

```
LIBNAME first 'path-name1';
LIBNAME second 'path-name2';
/* create concat.x */
LIBNAME concat (first second);

/* fails rule #1 */
proc catalog c=concat.x;
   copy out=first.x new;
run;
quit;

/* fails rule #2 */
proc catalog c=first.x;
   copy out=concat.x new;
run;
quit;
```

In summary, the following table shows when copies are allowed. In the table, A and B are libraries, and each contains catalog X. Catalog C is an automatic concatenation of A and B, and catalog D is an automatic concatenation of B and A.

Input catalog	Output catalog	Copy allowed?
C.X	B.X	No
C.X	D.X	No
D.X	C.X	No
A.X	A.X	No
A.X	B.X	Yes
B.X	A.X	Yes
C.X	A.X	No
B.X	C.X	Yes
A.X	C.X	No

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 8.2 ODS Tables Produced by the CATALOG Procedure

Table Name	Type of Library
Catalog_Random	when the catalog is in a random-access library.
Catalog_Sequential	when the catalog is in a sequential library.

Examples: CATALOG Procedure

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

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 demonstrates all the following actions:

- copies entries by excluding a few entries
- copies entries by specifying a few entries
- protects entries from being edited
- moves entries
- deletes entries
- processes entries from multiple catalogs
- processes entries in multiple run groups

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

```
LIBNAME perm 'SAS-library';
```

Delete two entries from the PERM.SAMPLE catalog.

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

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

```
   copy out=tcatall;
run;
```

Copy everything except three LOG entries and PASSIST.SLIST from PERM.SAMPLE to WORK.TESTCAT. The EXCLUDE statement specifies which entries not to copy. ET= specifies a default type. (ET=) specifies an exception to the default type.

```
   copy out=testcat;
      exclude test1 test2 test3  passist (et=slist) / et=log;
run;
```

Move three LOG entries from PERM.SAMPLE to WORK.LOGCAT. The SELECT statement specifies which entries to move. ET= restricts processing to LOG entries.

```
   copy out=logcat move;
      select test1 test2 test3 / et=log;
run;
```

Copy five SAS/AF software entries from PERM.SAMPLE to PERM.FINANCE. The NOEDIT option protects these entries in PERM.FINANCE from further editing with PROC BUILD.

```
   copy out=perm.finance noedit;
      select loan.frame loan.help loan.keys loan.pmenu;
run;
```

Copy two formats from PERM.FORMATS to PERM.FINANCE. The IN= option enables you to copy from a different catalog than the one specified in the PROC CATALOG statement. Note the entry types for numeric and character formats: REVENUE.FORMAT is a numeric format and DEPT.FORMATC is a character format. The COPY and SELECT statements execute before the QUIT statement ends the PROC CATALOG step.

```
   copy in=perm.formats out=perm.finance;
      select revenue.format dept.formatc;
quit;
```

SAS Log

```
1    LIBNAME perm 'SAS-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-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 demonstrates the following actions:

- lists the entries in a catalog and routes the output to a file

- changes entry names
- changes entry descriptions
- processes entries in multiple run groups

Program

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 library that contains a permanent SAS catalog.

```
LIBNAME perm 'SAS-library';
```

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

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

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

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

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

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

Output

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

                        Contents of Catalog PERM.FINANCE

# Name    Type            Create Date       Modified Date Description

1 REVENUE FORMAT   16OCT2007:13:48:11  16OCT2007:13:48:11 FORMAT:MAXLEN=16,16,12
2 DEPT    FORMATC  30OCT2007:13:40:42  30OCT2007:13:40:42 FORMAT:MAXLEN=1,1,14
3 LOAN    FRAME    30OCT2007:13:40:43  30OCT2007:13:40:43 Loan analysis
                                                          application
4 LOAN    HELP     16OCT2007:13:48:10  16OCT2007:13:48:10 Information about
                                                          the application
5 LOAN    KEYS     16OCT2007:13:48:10  16OCT2007:13:48:10 Custom key definitions
                                                          for application
6 LOAN    PMENU    16OCT2007:13:48:10  16OCT2007:13:48:10 Custom menu
                                                          definitions for
                                                          application
7 LOAN    SCL      16OCT2007:13:48:10  16OCT2007:13:48:10 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   16OCT2007:13:48:11  16OCT2007:13:48:11 FORMAT:MAXLEN=
                                                          16,16,12
2 DEPTCODE FORMATC  30OCT2006:13:40:42  30OCT2007:13:40:42 FORMAT:MAXLEN=1,1,14
3 LOAN     FRAME    30OCT2006:13:40:43  11FEB2007:13:20:50 Loan analysis
                                                          app. - ver1
4 LOAN     HELP     16OCT2007:13:48:10  16OCT2007:13:48:10 Information about
                                                          the application
5 LOAN     KEYS     16OCT2007:13:48:10  16OCT2007:13:48:10 Custom key
                                                          definitions for
                                                          application
6 LOAN     PMENU    16OCT2007:13:48:10  16OCT2007:13:48:10 Custom menu
                                                          definitions for
                                                          application
7 LOAN     SCL      16OCT2007:13:48:10  16OCT2007:13:48:10 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 does 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;
```

SAS Log

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

CHAPTER

9

The CHART Procedure

Overview: CHART Procedure **155**
What Does the CHART Procedure Do? **155**
What Types of Charts Can PROC CHART Create? **156**
Syntax: CHART Procedure **160**
PROC CHART Statement **161**
BLOCK Statement **163**
BY Statement **164**
HBAR Statement **165**
PIE Statement **165**
STAR Statement **166**
VBAR Statement **167**
Customizing All Types of Charts **168**
Concepts: CHART Procedure **173**
Results: CHART Procedure **174**
Missing Values **174**
ODS Table Names **174**
Portability of ODS Output with PROC CHART **174**
Examples: CHART Procedure **175**
Example 1: Producing a Simple Frequency Count **175**
Example 2: Producing a Percentage Bar Chart **177**
Example 3: Subdividing the Bars into Categories **179**
Example 4: Producing Side-by-Side Bar Charts **182**
Example 5: Producing a Horizontal Bar Chart for a Subset of the Data **185**
Example 6: Producing Block Charts for BY Groups **186**
References **189**

Overview: CHART Procedure

What Does the CHART Procedure Do?

The CHART procedure produces vertical and horizontal bar charts, block charts, pie charts, and star charts. These types of charts graphically display values of a variable or a statistic associated with those values. The charted variable can be numeric or character.

PROC CHART is a useful tool that lets you visualize data quickly, but if you need to produce presentation-quality graphics that include color and various fonts, then use SAS/GRAPH software. The GCHART procedure in SAS/GRAPH software produces the

same types of charts as PROC CHART does. In addition, PROC GCHART can produce donut charts.

What Types of Charts Can PROC CHART Create?

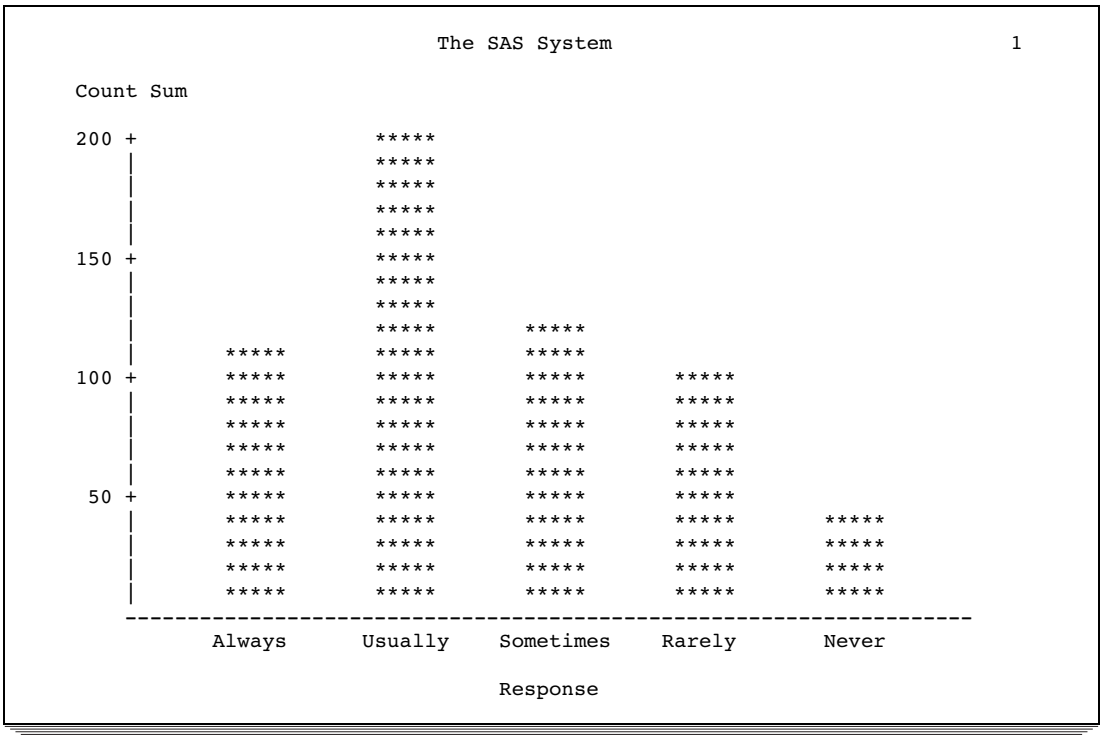
Bar Charts

Horizontal and vertical bar charts display the magnitude of data with bars, each of which represents a category of data. The length or height of the bars represents the value of the chart statistic for each category.

The following output shows a vertical bar chart that displays the number of responses for the five categories from the survey data. The following statements produce the output:

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

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

Output 9.1 Vertical Bar Chart



The following output shows the same data presented in a horizontal bar chart. The two types of bar charts have essentially the same characteristics, except that horizontal bar charts by default display a table of statistic values to the right of the bars. The following statements produce the output:

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

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

Output 9.2 Horizontal Bar Chart

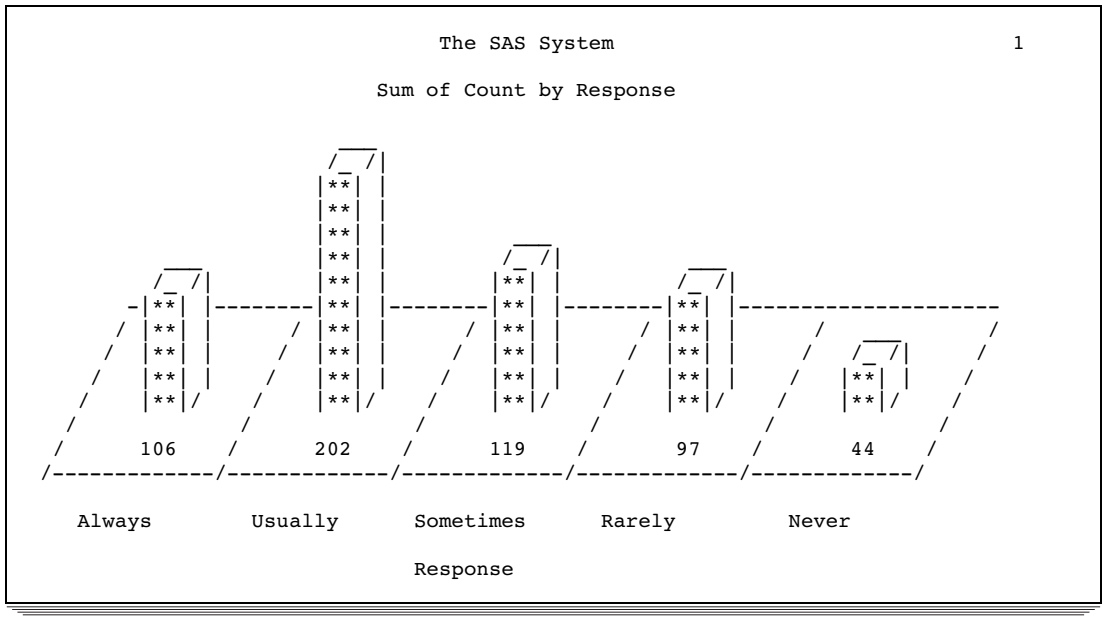
```
                              The SAS System                                1

      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. The following output 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 9.3 Block Chart

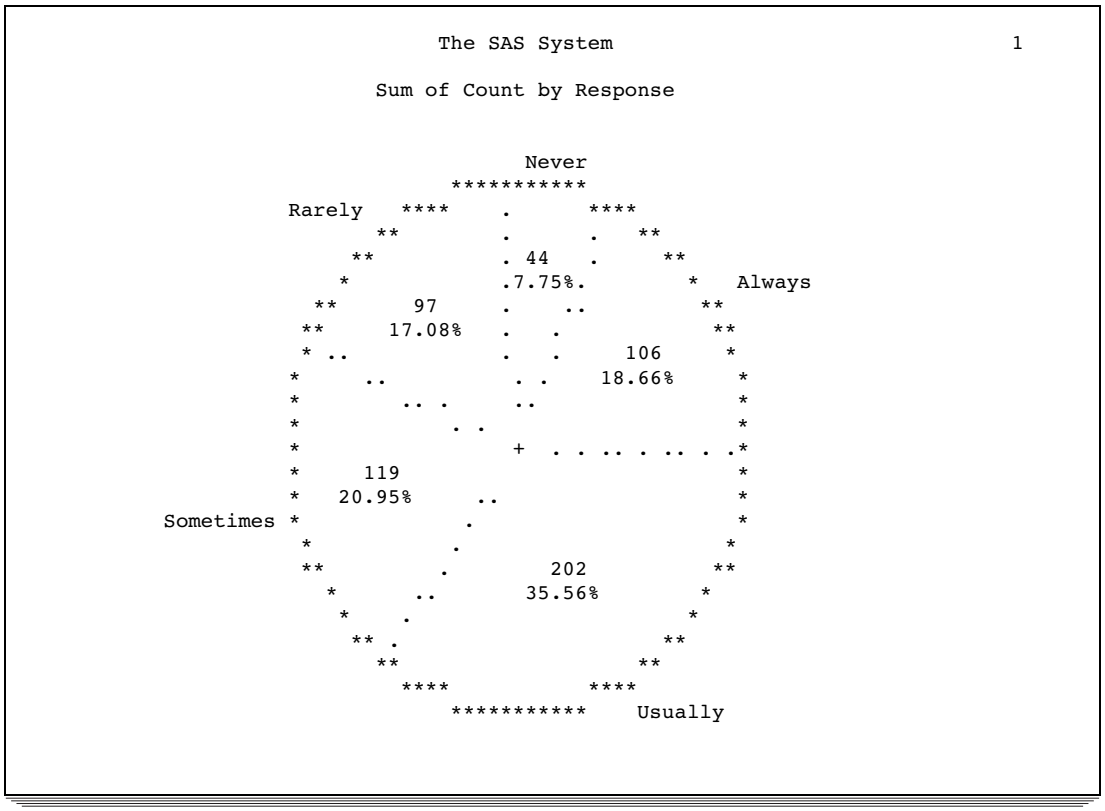


Pie Charts

Pie charts represent the relative contribution of parts to the whole by displaying data as wedge-shaped slices of a circle. Each slice represents a category of the data. The following output shows the survey results divided by response into five pie slices. The following statements produce the output:

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

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

Output 9.4 Pie Chart

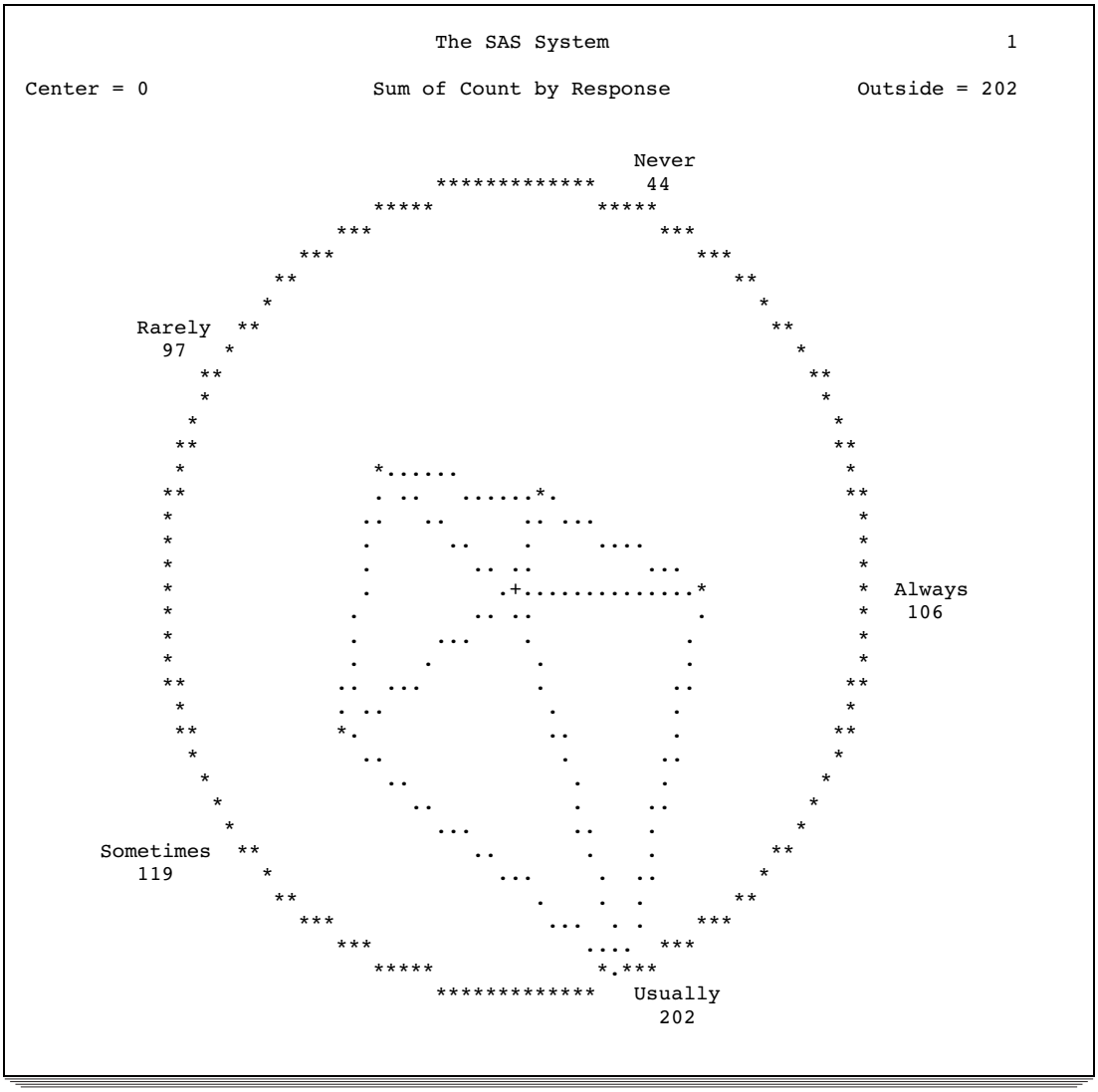


Star Charts

With PROC CHART, you can produce star charts that show group frequencies, totals, or mean values. A star chart is similar to a vertical bar chart, but the bars on a star chart radiate from a center point, like spokes in a wheel. Star charts are commonly used for cyclical data, such as measures taken every month or day or hour, or for data in which the categories have an inherent order ("always" meaning more frequent than "usually" which means more frequent than "sometimes"). The following output shows the survey data displayed in a star chart. The following statements produce the output:

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

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

Output 9.5 Star Chart



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: Basic Concepts in *SAS Output Delivery System: User's Guide* for details.

Tip: You can use the ATTRIB, FORMAT, LABEL, and WHERE statements. See Chapter 3, "Statements with the Same Function in Multiple Procedures," on page 35 for details. You can also use any global statements. See "Global Statements" on page 20 for a list.

PROC 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)*>;

Table 9.1

Task	Statement
Produce a chart	"PROC CHART Statement" on page 161
Produce a block chart	"BLOCK Statement" on page 163
Produce a separate chart for each BY group	"BY Statement" on page 164
Produce a horizontal bar chart	"HBAR Statement" on page 165
Produce a PIE chart	"PIE Statement" on page 165
Produce a STAR chart	"STAR Statement" on page 166
Produce a vertical bar chart	"VBAR Statement" on page 167

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 20

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 9.2 on page 162 shows the formatting characters that PROC CHART uses.

Figure 9.1 on page 163 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 example, the following option assigns the asterisk (*) to the second formatting character, the pound sign (#) to the seventh character, and does not alter the remaining characters:

```
formchar(2,7)='*#'
```

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

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

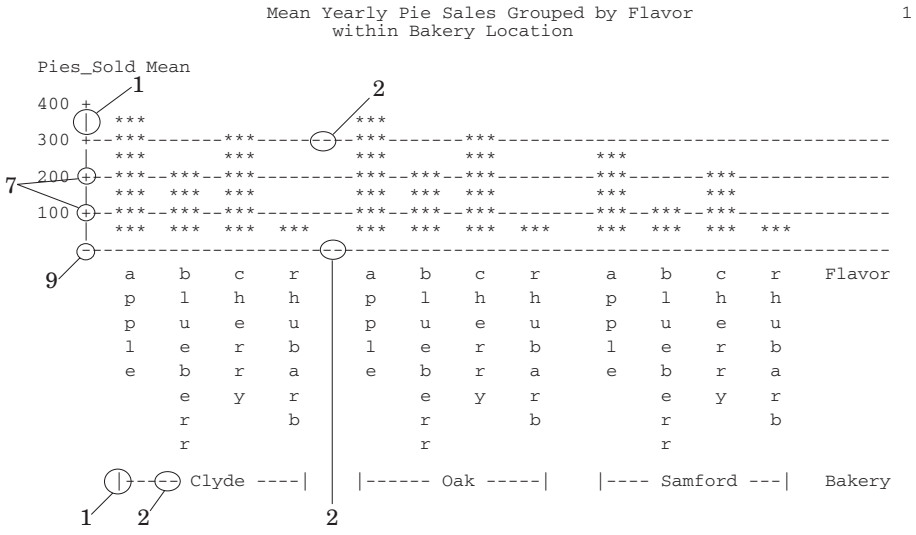
```
formchar(2,7)='2D7C'x
```

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

Table 9.2 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 9.1 Formatting Characters Commonly Used in PROC CHART Output



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 186

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 168.

Statement Results

Because each block chart must fit on one output page, you might have to adjust the SAS system options LINESIZE= and PAGESIZE= if you have a large number of charted values for the BLOCK variable and for the variable specified in the GROUP= option.

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

Table 9.3 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 36

Featured in: Example 6 on page 186

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 185

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 168.

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 168.

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 163 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 168.

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 169 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 163.

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 175
Example 2 on page 177
Example 3 on page 179
Example 4 on page 182

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 168.

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.

Task	Option
Specify that numeric variables are discrete	DISCRETE on page 170
Specify a frequency variable	FREQ= on page 170
Specify that missing values are valid levels	MISSING on page 171
Specify the variable for which values or means are displayed	SUMVAR= on page 172
Specify the statistic represented in the chart	TYPE= on page 173
Specify groupings	
Group the bars in side-by-side charts	GROUP= on page 170
Specify that group percentages sum to 100	G100 on page 170
Specify the number of bars for continuous variables	LEVELS= on page 170
Define ranges for continuous variables	MIDPOINTS= on page 171
Divide the bars into categories	SUBGROUP= on page 172
Compute statistics	
Compute the cumulative frequency for each bar	CFREQ on page 169
Compute the cumulative percentage for each bar	CPERCENT on page 169
Compute the frequency for each bar	FREQ on page 170
Compute the mean of the observations for each bar	MEAN on page 170
Compute the percentage of total observations for each bar	PERCENT on page 171
Compute the total number of observations for each bar	SUM on page 172
Control output format	
Print the bars in ascending order of size	ASCENDING on page 169
Specify the values for the response axis	AXIS= on page 169
Print the bars in descending order of size	DESCENDING on page 169
Specify extra space between groups of bars	GSPACE= on page 170
Suppress the default header line	NOHEADER on page 171
Allow no space between vertical bars	NOSPACE
Suppress the statistics	NOSTATS on page 171
Suppress the subgroup legend or symbol table	NOSYMBOL on page 171
Suppress the bars with zero frequency	NOZEROS on page 171
Draw reference lines	REF= on page 172
Specify the spaces between bars	SPACE= on page 172

Task	Option
Specify the symbols within bars or blocks	SYMBOL= on page 172
Specify the width of bars	WIDTH= on page 173

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 182, Example 5 on page 185, Example 6 on page 186

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 186

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 182

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 can 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 179

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 179, Example 4 on page 182, Example 5 on page 185, Example 6 on page 186

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 186

TYPE=*statistic*

specifies what the bars or sections in the chart represent. The *statistic* is one of the following:

CFREQ
specifies that each bar, block, or section represent the cumulative frequency.

CPERCENT
specifies that each bar, block, or section represent the cumulative percentage.

Alias: CPCT

FREQ
specifies that each bar, block, or section represent the frequency with which a value or range occurs for the chart variable in the data.

MEAN
specifies that each bar, block, or section represent the mean of the SUMVAR= variable across all observations that belong to that bar, block, or section.

Interaction: With TYPE=MEAN, you can compute only MEAN and FREQ statistics.

Featured in: Example 4 on page 182

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 177

SUM
specifies that each bar, block, or section represent the sum of the SUMVAR= variable for the observations that correspond to each bar, block, or section.

Default: FREQ (unless you use SUMVAR=, which causes a default of SUM)

Interaction: With TYPE=SUM, you can compute only SUM and FREQ statistics.

WIDTH=*n*

specifies the width of the bars on bar charts.

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 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 9.4 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.

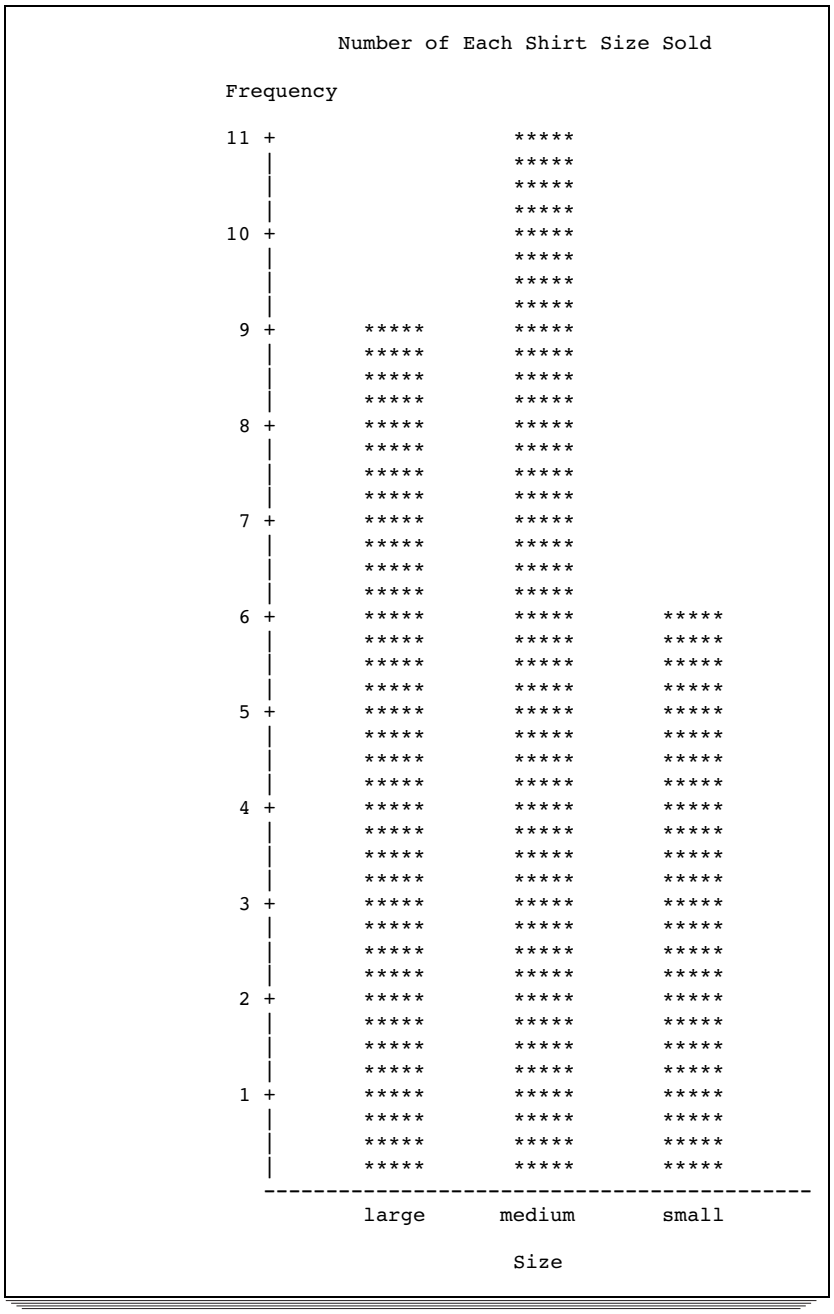
```
proc chart data=shirts;
   vbar size;
```

Specify the title.

```
    title 'Number of Each Shirt Size Sold';
run;
```

Output: Listing

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



Example 2: Producing a Percentage Bar Chart

Procedure features:

VBAR statement option:

TYPE=

Data set: SHIRTS on page 175

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.

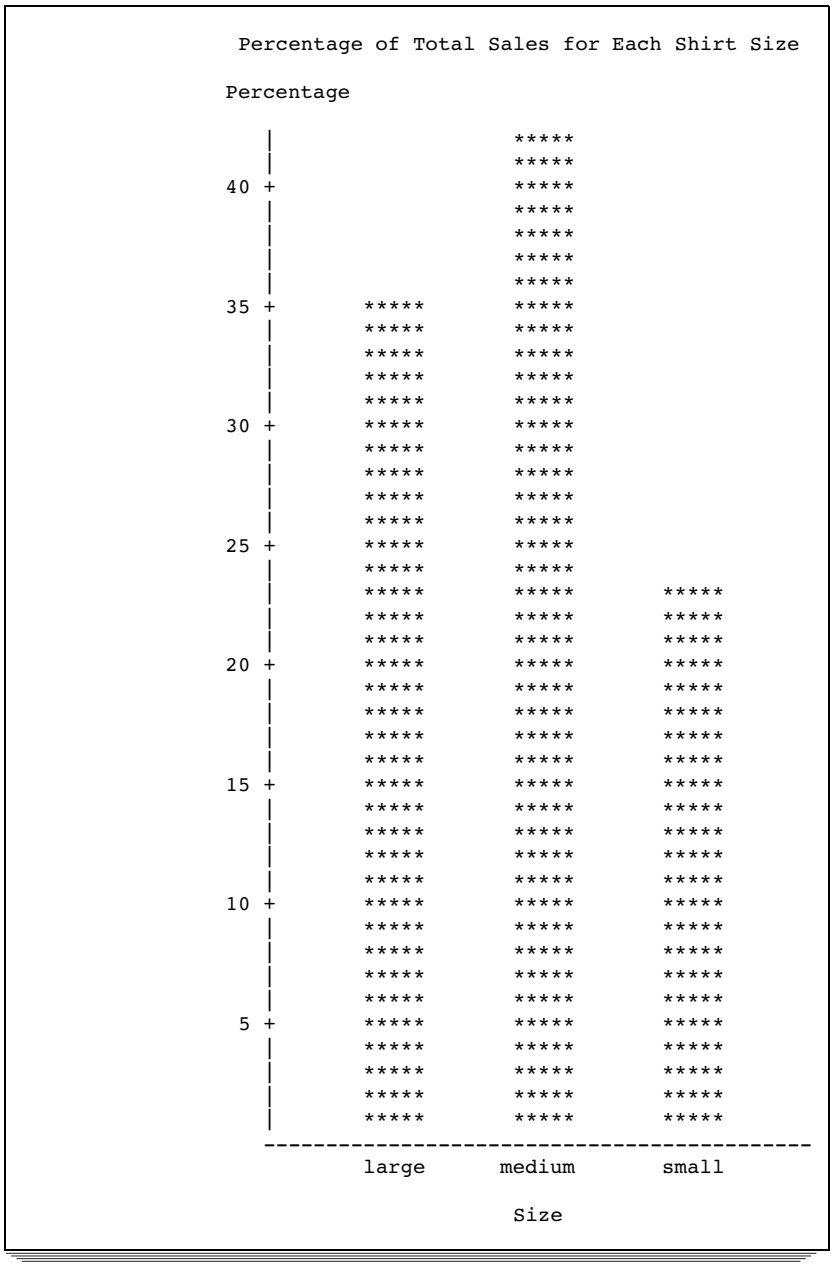
```
proc chart data=shirts;
   vbar size / type=percent;
```

Specify the title.

```
   title 'Percentage of Total Sales for Each Shirt Size';
run;
```

Output: Listing

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.



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.

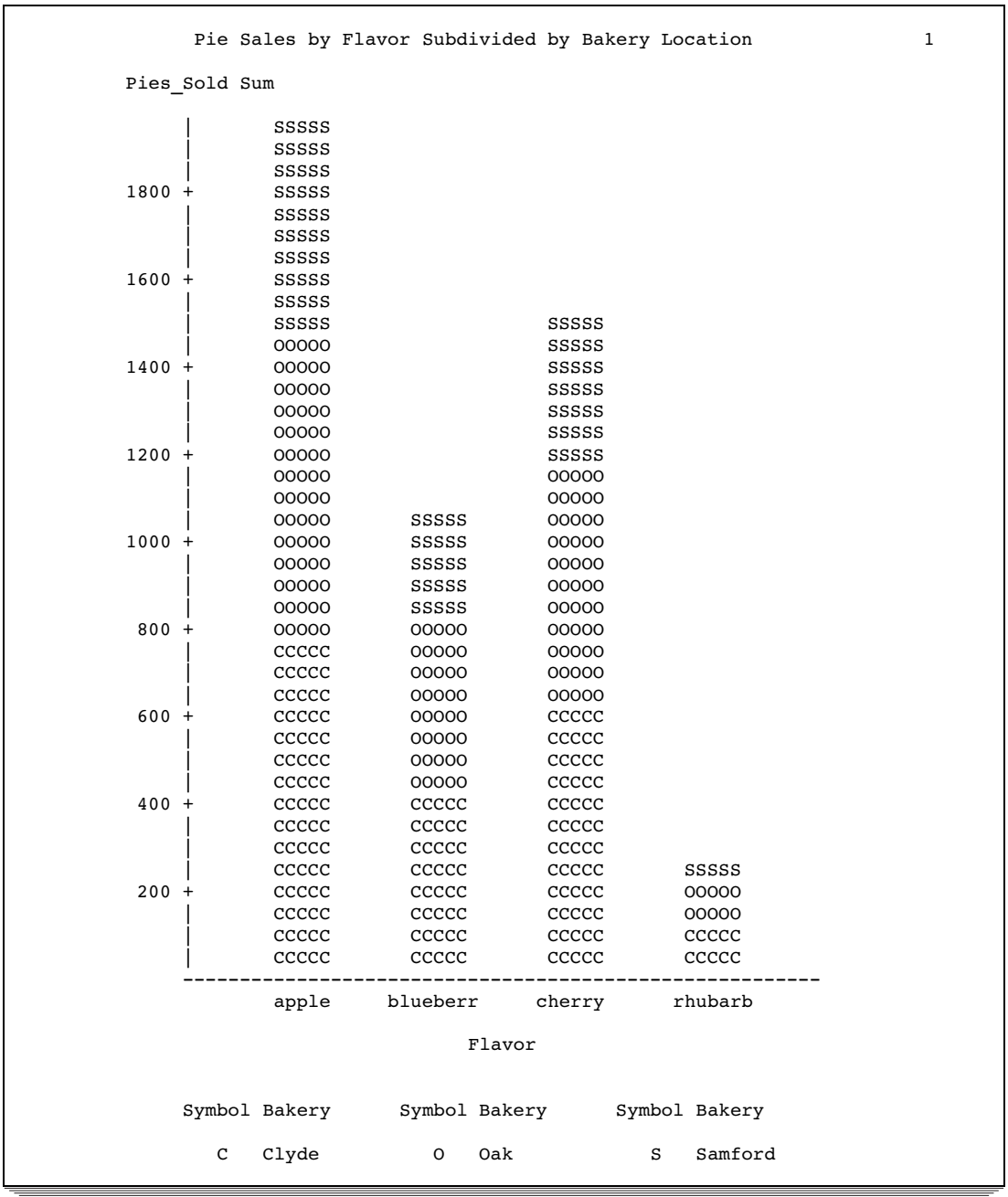
```
                 sumvar=pies_sold;
```

Specify the title.

```
   title 'Pie Sales by Flavor Subdivided by Bakery Location';
run;
```

Output: Listing

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.



Example 4: Producing Side-by-Side Bar Charts

Procedure features:

VBAR statement options:

GROUP=
REF=
SUMVAR=
TYPE=

Data set: PIESALES"Program" on page 180

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;
```

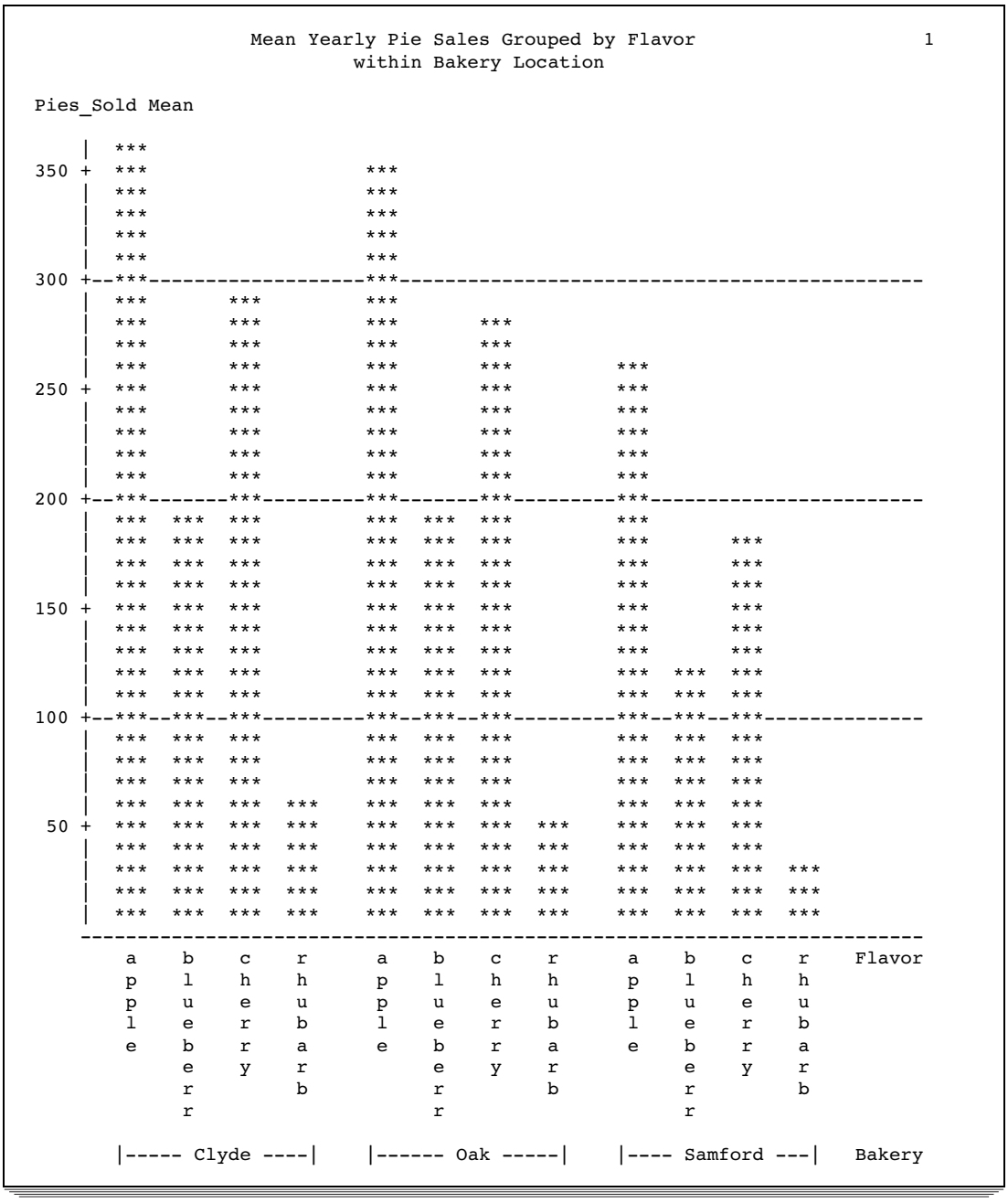
Specify the titles.

```
    title  'Mean Yearly Pie Sales Grouped by Flavor';
    title2 'within Bakery Location';
```

```
run;
```

Output: Listing

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.



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 180

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: Listing

```
              2007 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 180

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';
```

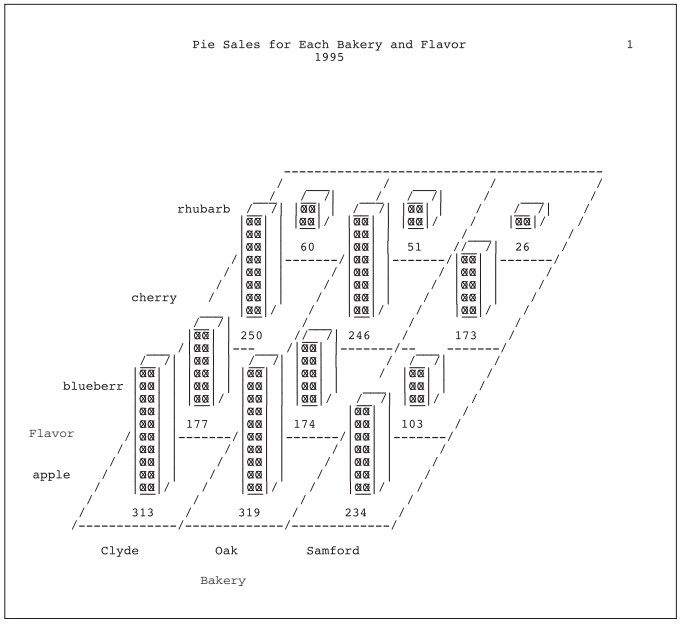
Specify the titles. The #BYVAL specification inserts the year into the second line of the title.

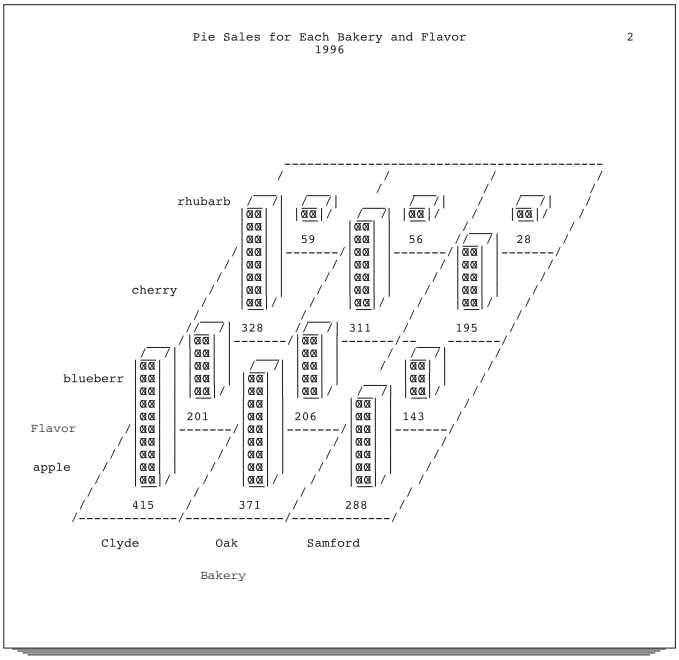
```
   title  'Pie Sales for Each Bakery and Flavor';
   title2 '#byval(year)';
run;
```

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

```
options byline;
```

Output: Listing





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 10

The CIMPORT Procedure

Overview: CIMPORT Procedure **191**
Purpose of the CIMPORT Procedure **191**
Process for Creating and Reading a Transport File **192**
Syntax: CIMPORT Procedure **192**
PROC CIMPORT Statement **193**
EXCLUDE Statement **196**
SELECT Statement **197**
CIMPORT Problems: Importing Transport Files **198**
About Transport Files and Encodings **198**
Problems with Transport Files Created Using a SAS Release Before 9.2 **199**
Overview: SAS Releases Before 9.2 **199**
Error: Transport File Encoding Is Unknown: Use the ISFILEUTF8= Option **199**
Warning: Transport File Encoding Is Unknown **200**
Problems with Transport Files Created Using SAS 9.2 **200**
Overview of SAS 9.2 **200**
Error: Target Session Uses UTF-8: Transport File Is not UTF-8 **201**
Error: Target Session Does Not Use UTF-8: Transport File Is UTF-8 **202**
Warning: Target Session Does Not Use UTF-8: Transport File Is Not UTF-8 **202**
Examples: CIMPORT Procedure **203**
Example 1: Importing an Entire Library **203**
Example 2: Importing Individual Catalog Entries **204**
Example 3: Importing a Single Indexed SAS Data Set **205**

Overview: CIMPORT Procedure

Purpose of the CIMPORT Procedure

The CIMPORT procedure *imports* a transport file that was created (*exported*) by the CPORT procedure. PROC CIMPORT restores the transport file to its original form as a SAS catalog, SAS data set, or SAS library. *Transport files* are sequential files that each contain a SAS library, a SAS catalog, or a SAS data set in transport format. The transport format that PROC CPORT writes is the same for all environments and for many releases of SAS.

PROC CIMPORT also *converts* SAS files, which means that it changes the format of a SAS file from the SAS format appropriate for one version of SAS to the SAS format appropriate for another version. For example, you can use PROC CPORT and PROC CIMPORT to move files from earlier releases of SAS to more recent releases (for example, from SAS 6 to SAS 9) or between the same versions (for example, from one

SAS 9 operating environment to another SAS 9 operating environment). PROC CIMPORT automatically converts the transport file as it imports it.

However, PROC CPORT and PROC CIMPORT do not allow file transport from a later version to an earlier version, which is known as regressing (for example, from SAS 9 to SAS 6).

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

Process for Creating and Reading a Transport File

Here is the process to create a transport file at the source computer and to read it at a target computer:

1. A transport file is created at the source computer using PROC CPORT.
2. The transport file is transferred from the source computer to the target computer via communications software or a magnetic medium.
3. The transport file is read at the target computer using PROC CIMPORT.

Note: Transport files that are created using PROC CPORT are not interchangeable with transport files that are created using the XPORT engine. △

For complete details about the steps to create a transport file (PROC CPORT), to transfer the transport file, and to restore the transport file (PROC CIMPORT), see *Moving and Accessing SAS Files*.

Syntax: CIMPORT Procedure

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*>;

Task	Statement
Imports a transport file	"PROC CIMPORT Statement" on page 193
Excludes one or more specified files from the import process.	"EXCLUDE Statement" on page 196
Specifies one or more files or entries to import process.	"SELECT Statement" on page 197

PROC CIMPORT Statement

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

Task	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	
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=
Specify whether the file is encoded in UTF-8 format	ISFILEUTF8=
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 catalog, SAS data set, or SAS library to import.

destination
identifies the file or files in the transport file as a single catalog, as a single SAS data set, or as the members of a SAS library. The *destination* argument can be one of the following:

CATALOG | CAT | C
DATA | DS | D
LIBRARY | LIB | L

libref | <libref. > member-name

specifies the specific catalog, SAS data set, or SAS library as the destination of the transport file. If the *destination* argument is CATALOG or DATA, you can specify both a *libref* and a member name. If the *libref* is omitted, PROC CIMPORT uses the default library as the *libref*, which is usually the WORK library. If the *destination* argument is LIBRARY, specify only a *libref*.

Options

EET=(*etype(s)*)
excludes specified entry types from the import process. If the *etype* is a single entry type, then you can omit the parentheses. Separate multiple values with spaces.

Interaction: You cannot specify both the EET= option and the ET= option in the same PROC CIMPORT step.

ET=(*etype(s)*)
specifies the entry types to import. If the *etype* is a single entry type, then you can omit the parentheses. Separate multiple values with spaces.

Interaction: You cannot specify both the EET= option and the ET= option in the same PROC CIMPORT step.

EXTENDSN=YES | NO
specifies whether to extend by 1 byte the length of short numerics (fewer than 8 bytes) when you import them. You can avoid a loss of precision when you transport a short numeric in IBM format to IEEE format if you extend its length. You cannot extend the length of an 8-byte short numeric.

Default: YES

Restriction: This option applies only to data sets.

Tip: Do not store fractions as short numerics.

ISFILEUTF8=YES | NO
ISFILEUTF8=TRUE | FALSE
explicitly designates the encoding of a data set that is contained in a transport file as UTF-8. Although data set encodings are recorded (or stamped) in SAS 9.2 transport files, encodings are not stamped in transport files created using SAS releases before 9.2. Therefore, designating the UTF-8 encoding is useful under these conditions:

- The data set in the transport file was created using a SAS release before 9.2.
- The data set is known to be encoded as UTF-8.

The person who restores the transport file in the target environment should have a description of the transport file in advance of the restore operation.

YES | Y | yes | y | TRUE | true | T | t
specifies that the data set in the transport file is encoded as UTF-8.

NO | N | no | n | FALSE | false | F | f
specifies that the data set in the transport file is not encoded as UTF-8. NO is the default.

Default: NO

Restriction: PROC CIMPORT uses this option only if the transport file is not stamped with the encoding of the data set. Encodings were not recorded in SAS releases before 9.2. If an encoding is recorded in the transport file and the ISFILEUTF8= option is specified in PROC CIMPORT, ISFILEUTF8= is ignored.

Tips: In order to successfully import a transport file in the target SAS session, you should have this information about the transport file:

- Source operating environment; for example, **Windows**
- SAS release; for example **SAS 9.2**
- Name of the transport file; for example, **tport.dat**
- Encoding of the character data; for example **wlatin1**
- National language of the character data; for example, American English (or **en_US**)

See Also: "CIMPORT Problems: Importing Transport Files" on page 198

See Also: For more information about creating and restoring transport files, see *Moving and Accessing SAS Files*.

FORCE
enables access to a locked catalog. By default, PROC CIMPORT locks the catalog that it is updating to prevent other users from accessing the catalog while it is being updated. The FORCE option overrides this lock, which allows other users to access the catalog while it is being imported, or 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 203.

MEMTYPE=*mtype*
specifies that only data sets, only catalogs, or both, be imported from the transport file. Values for *mtype* can be as follows:

ALL
both catalogs and data sets

CATALOG | CAT
catalogs

DATA | DS
SAS data sets

NEW
creates a new catalog to contain the contents of the imported transport file when the destination 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.

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

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

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

Required Arguments

SAS file(s) | catalog entry(s)
specifies one or more SAS files or one or more catalog entries to be excluded from the import process. Specify SAS filenames if you import a library; specify catalog entry names if you import an individual SAS catalog. Separate multiple filenames or entry names with a space. You can use shortcuts to list many like-named files in the EXCLUDE statement. For more information, see “Shortcuts for Specifying Lists of Variable Names” on page 25.

Options

ENTRYTYPE=*entry-type*
specifies a single entry type for one or more catalog entries that are listed in the EXCLUDE statement. See *SAS Language Reference: Concepts* for a complete list of catalog entry types.

Alias: ETYPE=, ET=

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

MEMTYPE=*mtype*
specifies a single member type for one or more SAS files listed in the EXCLUDE statement. Values for *mtype* can be

ALL
both catalogs and data sets

CATALOG
catalogs

DATA
SAS data sets.

You can also specify the MEMTYPE= option, enclosed in parentheses, immediately after the name of a file. In parentheses, MEMTYPE= identifies the type of the filename that just precedes it. When you use this form of the option, it overrides the MEMTYPE= option that follows the slash in the EXCLUDE statement, but it must match the MEMTYPE= option in the PROC CIMPORT statement.

Alias: MTYPE=, MT=

Default: ALL

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

SELECT Statement

Specifies individual files or entries to import.

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

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

Featured in: Example 2 on page 204

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

Required Arguments

SAS file(s) | catalog entry(s)
specifies one or more SAS files or one or more catalog entries to import. Specify SAS filenames if you import a library; specify catalog entry names if you import an individual SAS catalog. Separate multiple filenames or entry names with a space. You can use shortcuts to list many like-named files in the SELECT statement. For more information, see “Shortcuts for Specifying Lists of Variable Names” on page 25.

Options

ENTRYTYPE=*entry-type*

specifies a single entry type for one or more catalog entries that are listed in the SELECT statement. See *SAS Language Reference: Concepts* for a complete list of catalog entry types.

Alias: ETYPE=, ET=

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

MEMTYPE=*mtype*
specifies a single member type for one or more SAS files listed in the SELECT statement. Valid values are CATALOG or CAT, DATA, or ALL.

You can also specify the MEMTYPE= option, enclosed in parentheses, immediately after the name of a file. In parentheses, MEMTYPE= identifies the type of the filename that just precedes it. When you use this form of the option, it overrides the MEMTYPE= option that follows the slash in the SELECT statement, but it must match the MEMTYPE= option in the PROC CIMPORT statement.

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

Alias: MTYPE=, MT=

Default: ALL

CIMPORT Problems: Importing Transport Files

About Transport Files and Encodings

The character data in a transport file is created in either of two types of encodings:

- the UTF-8 encoding of the SAS session in which the transport file is created
- the Windows encoding that is associated with the locale of the SAS session in which the transport file is created

These examples show how SAS applies an encoding to a transport file:

Table 10.1 Assignment of Encodings to Transport Files

Encoding Value of the Transport File	Example of Applying an Encoding in a SAS Invocation	Explanation
UTF-8	sas9 -encoding utf8;	A SAS session is invoked using the UTF-8 encoding. The session encoding is applied to the transport file .
wlatin2	sas9 -locale pl_PL;	A SAS session is invoked using the default UNIX encoding, latin2, which is associated with the Polish Poland locale.

For a complete list of encodings that are associated with each locale, see the Locale Table in *SAS National Language Support (NLS): Reference Guide*.

The encodings of the source and target SAS sessions must be compatible in order for a transport file to be imported successfully.. Here is an example of compatible source and target SAS sessions:

Table 10.2 Compatible Encodings

Source SAS Session			Target SAS Session	
Locale	UNIX SAS Session Encoding	Transport File Encoding	Locale	Windows SAS Session Encoding
es_MX (Spanish Mexico)	latin1	wlatin1	it_IT (Italian Italy)	wlatin1

The encodings of the source and target SAS sessions are compatible because the Windows default encoding for the es_MX locale is wlatin1 and the encoding of the target SAS session is wlatin1.

However, if the encodings of the source and target SAS sessions are incompatible, a transport file cannot be successfully imported. Here is an example of incompatible encodings:

Table 10.3 Incompatible Encodings

Source SAS Session			Target SAS Session	
Locale	UNIX SAS Session Encoding	Transport File Encoding	Locale	z/OS Encoding
cs_CZ (Czech Czechoslovakia)	latin2	wlatin2	de_DE (German Germany)	open_ed-1141

The encodings of the source and target SAS sessions are incompatible because the Windows default encoding for the cs_CZ locale is wlatin2 and the encoding of the target SAS session is open_ed-1141. A transport file cannot be imported between these locales.

When importing transport files, you will be alerted to compatibility problems via warnings and error messages.

Problems with Transport Files Created Using a SAS Release Before 9.2

Overview: SAS Releases Before 9.2

Transport files that were created by SAS releases before 9.2 are not stamped with encoding values. Therefore, the CIMPORT procedure does not know the identity of the transport file's encoding and cannot report specific warning and error detail. The encoding of the transport file must be inferred when performing recovery actions.

However, using your knowledge about the transport file, you should be able to recover from transport problems. For information that is useful for importing the transport file in the target SAS session, see Transport File Tips on page 194. For complete details about creating and restoring transport files, see *Moving and Accessing SAS Files*.

Here are the warning and error messages with recovery actions:

- "Error: Transport File Encoding Is Unknown: Use the ISFILEUTF8= Option" on page 199
- "Warning: Transport File Encoding Is Unknown" on page 200

Error: Transport File Encoding Is Unknown: Use the ISFILEUTF8= Option

The error message provides this information:

- The transport file was created using a SAS release before 9.2.
- Because the encoding is not stamped in the transport file, the encoding is unknown.
- The target SAS session uses the UTF-8 encoding.

Note: In order to perform recovery steps, you must know the encoding of the transport file. △

If you know that the transport file is encoded as UTF-8, you can import the file again, and use the ISFILEUTF8=YES option in PROC CIMPORT.

Example: UTF-8 transport file, which was created using a SAS release before 9.2, and UTF-8 target SAS session

```
filename importin 'transport-file';
libname target 'SAS-data-library';
proc cimport isfileutf8=yes infile=importin library=target memtype=data;
run;
```

For syntax details, see the ISFILEUTF8= Option in PROC CIMPORT on page 194.

PROC CIMPORT should succeed.

Warning: Transport File Encoding Is Unknown

The warning message provides this information:

- The transport file was created using a SAS release before 9.2.
- Because the encoding is not stamped in the transport file, the encoding is unknown.

Try to read the character data from the imported data set. If you cannot read the data, you can infer that the locale of the target SAS session is incompatible with the encoding of the transport file.

Note: In order to perform recovery steps, you must know the encoding of the transport file. △

Example: The transport file, which was created using a Polish Poland locale, was created in a source SAS session using a SAS release before 9.2. The target SAS session uses a German locale.

1. In the target SAS session, start another SAS session and change the locale to the locale of the source SAS session that created the transport file.

   In this example, you start a new SAS session in the Polish Poland locale.

   ```
   sas9 -locale pl_PL;
   ```

2. Import the file again. Here is an example:

   ```
   filename importin 'transport-file';
   libname target 'SAS-data-library';
   proc cimport infile=importin library=target memtype=data;
   run;
   ```

PROC CIMPORT should succeed and the data should be readable in the SAS session that uses a Polish_Poland locale.

Problems with Transport Files Created Using SAS 9.2

Overview of SAS 9.2

The encoding of the character data is stamped in transport files that are created using SAS 9.2. Therefore, the CIMPORT procedure can detect error conditions such as

UTF8–encoded transport files cannot be imported into SAS sessions that do not use the UTF–8 encoding. For example, a UTF-8 transport file cannot be imported into a SAS session that uses the Wlatin2 encoding.

Also, SAS 9.2 can detect the condition of incompatibility between the encoding of the transport file and the locale of the target SAS session. Because some customers' SAS applications ran successfully using a release before SAS 9.2, PROC CIMPORT will report a warning only, but will allow the import procedure to continue.

Here are the warning and error messages with recovery actions:

- "Error: Target Session Uses UTF-8: Transport File Is not UTF-8" on page 201
- "Error: Target Session Does Not Use UTF-8: Transport File Is UTF-8" on page 202
- "Warning: Target Session Does Not Use UTF-8: Transport File Is Not UTF-8" on page 202

Error: Target Session Uses UTF-8: Transport File Is not UTF-8

The error message provides this information:

- The target SAS session uses the UTF-8 encoding.
- The transport file has an identified encoding that is not UTF-8. The encodings of the transport file and the target SAS session are incompatible.

The encoding of the target SAS session cannot be UTF–8. Also, the locales of the source and target SAS sessions must be identical.

Example: SAS 9.2 Wlatin2 transport file and UTF-8 target SAS session:

1 To recover, in the target SAS session, start another SAS session and change the locale to the locale that was used in the source SAS session that created the transport file.

The LOCALE= value is preferred over the ENCODING= value because it sets automatically the default values for the ENCODING=, DFLANG=, DATESTYLE= and PAPERSIZE= options.

If you do not know the locale of the source session (or the transport file), you can infer it from the national language that is used by the character data in the transport file.

For example, if you know that Polish is the national language, specify the **pl_PL** (Polish Poland) locale in a new target SAS session. Here are the encoding values that are associated with the **pl_PL** locale:

Table 10.4 LOCALE= Value for the Polish Language

Posix Locale	Windows Encoding	UNIX Encoding	z/OS Encoding
pl_PL (Polish Poland)	wlatin2	latin2	open_ed-870

Here is an example of specifying the **pl_PL** locale in a new SAS session:

```
sas9 -locale pl_PL;
```

For complete details, see the Locale Table in *SAS National Language Support (NLS): Reference Guide*.

Note: Verify that you do not have a SAS invocation command that already contains the specification of the UTF-8 encoding; for example, **sas9 -encoding utf8;**. If it exists, the UTF-8 encoding would persist regardless of a new locale specification. △

2 Import the file again. Here is an example:

```
filename importin 'transport-file';
libname target 'SAS-data-library';
proc cimport infile=importin library=target memtype=data;
run;
```

PROC CIMPORT should succeed.

Error: Target Session Does Not Use UTF-8: Transport File Is UTF-8

The error message provides this information:

- The target session uses an identified encoding.
- The transport file is encoded as UTF-8. The encodings of the transport file and the target SAS session are incompatible.

The encoding of the target SAS session must be changed to UTF-8.
Example: SAS 9.2 UTF-8 transport file and Wlatin1 target SAS session:

1 To recover, in the target SAS session, start a new SAS session and change the session encoding to UTF-8. Here is an example:

```
sas9 -encoding utf8;
```

2 Import the file again. Here is an example:

```
filename importin 'transport-file';
libname target 'SAS-data-library';
proc cimport infile=importin library=target memtype=data;
run;
```

PROC CIMPORT should succeed.

Warning: Target Session Does Not Use UTF-8: Transport File Is Not UTF-8

The warning message provides this information:

- The target SAS session uses an identified encoding.
- The encoding of the transport file is identified. The encodings of the transport file and the target SAS session are incompatible.

Example: Wlatin2 transport file and open_ed-1141 target SAS session
This table shows the locale and encoding values of incompatible source and target SAS sessions. Although the wlatin2 Windows encoding that is assigned to the transport file in the source SAS session is incompatible with the open_ed-1141 encoding of the target SAS session, a warning is displayed and the import will continue.

Table 10.5 Encoding Values for the Czech and German Locales

SAS Session	Posix Locale	Windows Encoding	UNIX Encoding	z/OS Encoding
Source SAS Session	cs_CZ (Czech Czechoslovakia)	wlatin2	latin2	open_ed-870
Target SAS Session	de_DE (German Germany)	wlatin1	latin9	open_ed-1141

The transport file is imported, but the contents of the file are questionable. The message identifies the incompatible encoding formats. To recover, try to read the contents of the imported file. If the file is unreadable, perform these steps:

1 In the target SAS session, start a new SAS session and change the locale (rather than the encoding) to the locale that is used in the source SAS session.

   The LOCALE= value is preferred over the ENCODING= value because it automatically sets the default values for the ENCODING=, DFLANG=, DATESTYLE= and PAPERSIZE= options.

   If you do not know the locale of the source session (or the transport file), you can infer it from the national language of the transport file.

   For example, if you know that Czech is the national language, specify the **cs_CZ** locale in a new target SAS session.

   Here is an example of specifying the **cs_CZ** locale in a new SAS session:

   ```
   sas9 -locale cs_CZ;
   ```

   The target SAS session and the transport file use compatible encodings. They both use wlatin2.

   For complete details, see the Locale Table in *SAS National Language Support (NLS): Reference Guide*.

2 Import the file again. Here is an example:

   ```
   filename importin 'transport-file';
   libname target 'SAS-data-library';
   proc cimport infile=importin library=target memtype=data;
   run;
   ```

PROC CIMPORT should succeed.

Examples: CIMPORT Procedure

Example 1: Importing an Entire 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 library in another operating environment. The transport file was moved to the new operating environment by means of communications software or magnetic medium. PROC CIMPORT imports the transport file to a SAS library, called NEWLIB, in the new operating environment.

Program

Specify the library name and filename. The LIBNAME statement specifies a libname for the new SAS library. The FILENAME statement specifies the filename of the transport file that PROC CPORT created and enables you to specify any operating environment options for file characteristics.

```
libname newlib 'SAS-data-library';
filename tranfile 'transport-file'
```

```
                    host-option(s)-for-file-characteristics;
```

Import the SAS library in the NEWLIB library. PROC CIMPORT imports the SAS library into the library named NEWLIB.

```
proc cimport library=newlib infile=tranfile;
run;
```

SAS Log

```
 NOTE: Proc CIMPORT begins to create/update catalog NEWLIB.FINANCE
NOTE: Entry LOAN.FRAME has been imported.
NOTE: Entry LOAN.HELP has been imported.
NOTE: Entry LOAN.KEYS has been imported.
NOTE: Entry LOAN.PMENU has been imported.
NOTE: Entry LOAN.SCL has been imported.
NOTE: Total number of entries processed in catalog NEWLIB.FINANCE: 5

NOTE: Proc CIMPORT begins to create/update catalog NEWLIB.FORMATS
NOTE: Entry REVENUE.FORMAT has been imported.
NOTE: Entry DEPT.FORMATC has been imported.
NOTE: Total number of entries processed in catalog NEWLIB.FORMATS: 2
```

Example 2: Importing Individual Catalog Entries

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 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 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

11

The COMPARE Procedure

Overview: COMPARE Procedure **208**
What Does the COMPARE Procedure Do? **208**
What Information Does PROC COMPARE Provide? **208**
How Can PROC COMPARE Output Be Customized? **209**
Syntax: COMPARE Procedure **211**
PROC COMPARE Statement **211**
BY Statement **218**
ID Statement **219**
VAR Statement **221**
WITH Statement **222**
Concepts: COMPARE Procedure **222**
Comparisons Using PROC COMPARE **222**
A Comparison by Position of Observations **223**
A Comparison with an ID Variable **224**
The Equality Criterion **224**
Using the CRITERION= Option **224**
Definition of Difference and Percent Difference **226**
How PROC COMPARE Handles Variable Formats **226**
Results: COMPARE Procedure **226**
Results Reporting **226**
SAS Log **226**
Macro Return Codes (SYSINFO) **227**
Procedure Output **228**
Procedure Output Overview **228**
Data Set Summary **228**
Variables Summary **229**
Observation Summary **230**
Values Comparison Summary **231**
Value Comparison Results **232**
Table of Summary Statistics **233**
Comparison Results for Observations (Using the TRANSPOSE Option) **235**
ODS Table Names **236**
Output Data Set (OUT=) **237**
Output Statistics Data Set (OUTSTATS=) **238**
Examples: COMPARE Procedure **239**
Example 1: Producing a Complete Report of the Differences **239**
Example 2: Comparing Variables in Different Data Sets **244**
Example 3: Comparing a Variable Multiple Times **245**
Example 4: Comparing Variables That Are in the Same Data Set **247**
Example 5: Comparing Observations with an ID Variable **249**
Example 6: Comparing Values of Observations Using an Output Data Set (OUT=) **253**

Overview: COMPARE Procedure

What Does the COMPARE Procedure Do?

The COMPARE procedure compares the contents of two SAS data sets, selected variables in different data sets, or variables within the same data set.

PROC COMPARE compares two data sets: the *base data set* and the *comparison data set*. The procedure determines matching variables and matching observations. *Matching variables* are variables with the same name or variables that you pair by using the VAR and WITH statements. Matching variables must be of the same type. *Matching observations* are observations that have the same values for all ID variables that you specify or, if you do not use the ID statement, that occur in the same position in the data sets. If you match observations by ID variables, then both data sets must be sorted by all ID variables.

What Information Does PROC COMPARE Provide?

PROC COMPARE generates the following information about the two data sets that are being compared:

- whether matching variables have different values
- whether one data set has more observations than the other
- what variables the two data sets have in common
- how many variables are in one data set but not in the other
- whether matching variables have different formats, labels, or types.
- a comparison of the values of matching observations.

Further, PROC COMPARE creates two kinds of output data sets that give detailed information about the differences between observations of variables 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 11.1 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 228 shows the default output for these two data sets. Example 1 on page 239 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: Basic Concepts in *SAS Output Delivery System: User's Guide* for details.

Tip: You can use the LABEL, ATTRIB, FORMAT, and WHERE statements. See Chapter 3, "Statements with the Same Function in Multiple Procedures," on page 35 for details. You can also use any global statements. See "Global Statements" on page 20 for a list.

PROC 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)*;

Task	Statement
Compare the contents of SAS data sets, or compare two variables	"PROC COMPARE Statement" on page 211
Produce a separate comparison for each BY group	"BY Statement" on page 218
Identify variables to use to match observations	"ID Statement" on page 219
Restrict the comparison to values of specific variables	"VAR Statement" on page 221
Compare variables of different names	"WITH Statement" on page 222 and "VAR Statement" on page 221
Compare two variables in the same data set	"WITH Statement" on page 222 and "VAR Statement" on page 221

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)*>;

Task	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	
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

Task	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 233 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 247

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 that case, the number itself becomes the equality criterion. If you use a negative value for γ, then PROC COMPARE uses an equality criterion proportional to the precision of the computer on which SAS is running.

Default: 0.00001

See also: "The Equality Criterion" on page 224

ERROR
displays an error message in the SAS log when differences are found.

Interaction: This option overrides the WARNING option.

FUZZ=*number*
alters the values comparison results for numbers less than *number*. PROC COMPARE prints

- 0 for any variable value that is less than *number*
- a blank for difference or percent difference if it is less than *number*
- 0 for any summary statistic that is less than *number*.

Default 0

Range: 0 - 1

Tip: A report that contains many trivial differences is easier to read in this form.

LISTALL
lists all variables and observations that are found in only one data set.

Alias LIST

Interaction: using LISTALL is equivalent to using the following four options: LISTBASEOBS, LISTCOMPOBS, LISTBASEVAR, and LISTCOMPVAR.

LISTBASE
lists all observations and variables that are found in the base data set but not in the comparison data set.

Interaction: Using LISTBASE is equivalent to using the LISTBASEOBS and LISTBASEVAR options.

LISTBASEOBS
lists all observations that are found in the base data set but not in the comparison data set.

LISTBASEVAR
lists all variables that are found in the base data set but not in the comparison data set.

LISTCOMP
lists all observations and variables that are found in the comparison data set but not in the base data set.

Interaction: Using LISTCOMP is equivalent to using the LISTCOMPOBS and LISTCOMPVAR options.

LISTCOMPOBS
lists all observations that are found in the comparison data set but not in the base data set.

LISTCOMPVAR
lists all variables that are found in the comparison data set but not in the base data set.

LISTEQUALVAR
prints a list of variables whose values are judged equal at all observations in addition to the default list of variables whose values are judged unequal.

LISTOBS
lists all observations that are found in only one data set.

Interaction: Using LISTOBS is equivalent to using the LISTBASEOBS and LISTCOMPOBS options.

LISTVAR
lists all variables that are found in only one data set.

Interaction: Using LISTVAR is equivalent to using both the LISTBASEVAR and LISTCOMPVAR options.

MAXPRINT=*total* | (*per-variable, total*)
specifies the maximum number of differences to print, where

total
is the maximum total number of differences to print. The default value is 500 unless you use the ALLOBS option (or both the ALLVAR and TRANSPOSE options). In that case, the default is 32000.

per-variable
is the maximum number of differences to print for each variable within a BY group. The default value is 50 unless you use the ALLOBS option (or both the ALLVAR and TRANSPOSE options). In that case, the default is 1000.

The MAXPRINT= option prevents the output from becoming extremely large when data sets differ greatly.

METHOD=ABSOLUTE | EXACT | PERCENT | RELATIVE<(δ)>
specifies the method for judging the equality of numeric values. The constant δ (delta) is a number between 0 and 1 that specifies a value to add to the denominator when calculating the equality measure. By default, δ is 0.

Unless you use the CRITERION= option, the default method is EXACT. If you use the CRITERION= option, then the default method is RELATIVE(ϕ), where ϕ (phi) is a small number that depends on the numerical precision of the computer on which SAS is running and on the value of CRITERION=.

See also: "The Equality Criterion" on page 224

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 equal only to a missing value of the same kind, that is .=., .^=.A, .A=.A, .A^=.B, and so on.

Alias: NOMISS

Interaction: Using NOMISSING is equivalent to using both NOMISSBASE and NOMISSCOMP.

NOPRINT
suppresses all printed output.

Tip: You may want to use this option when you are creating one or more output data sets.

Featured in: Example 6 on page 253

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 244

NOTE
displays notes in the SAS log that describe the results of the comparison, whether differences were found.

NOVALUES
suppresses the report of the value comparison results.

Featured in: "Overview: COMPARE Procedure" on page 208

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 237

Featured in: Example 6 on page 253

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 237

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 237

Featured in: Example 6 on page 253

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 237

Featured in: Example 6 on page 253

OUTDIF
writes an observation to the output data set for each pair of matching observations. The values in the observation include values for the differences between the values in the pair of observations. The value of _TYPE_ in each observation is DIF.

Default: The OUTDIF option is the default unless you specify the OUTBASE, OUTCOMP, or OUTPERCENT option. If you use any of these options, then you must specify the OUTDIF option to create _TYPE_=DIF observations in the output data set.

See also: "Output Data Set (OUT=)" on page 237

Featured in: Example 6 on page 253

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 237

Featured in: Example 6 on page 253

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 237

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 238

"Table of Summary Statistics" on page 233

Featured in: Example 7 on page 255

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

Featured in: Example 1 on page 239

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 233 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 235.

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 36

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 the ones of the BY variables in the base data set. The following table shows how PROC COMPARE behaves under different circumstances:

Condition	Behavior of PROC COMPARE
All BY variables are in the comparison data set and all attributes match exactly	Compares corresponding BY groups
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 224
Featured in: Example 5 on page 249

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 220

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 duplicate observations in the base data set and the comparison data set to compare the observations on a one-to-one basis.

When the data sets are not sorted, PROC COMPARE detects only those duplicate observations that occur in succession.

VAR Statement

Restricts the comparison of the values of variables to the ones named in the VAR statement.

Featured in:

Example 2 on page 244

Example 3 on page 245

Example 4 on page 247

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 the ones that appear in BY and ID statements.
- If a variable in the VAR statement does not exist in the COMPARE= data set, then PROC COMPARE writes a warning message to the SAS log and ignores the variable.
- If a variable in the VAR statement does not exist in the BASE= data set, then PROC COMPARE stops processing and writes an error message to the SAS log.
- The VAR statement restricts only the comparison of values of matching variables. PROC COMPARE still reports on the total number of matching variables and compares their attributes. However, it produces neither error nor warning messages about these variables.

WITH Statement

Compares variables in the base data set with variables that have different names in the comparison data set, and compares different variables that are in the same data set.

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

Featured in:

Example 2 on page 244
Example 3 on page 245
Example 4 on page 247

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

The following figure shows two data sets. The data inside the shaded boxes shows the part of the data sets that the procedure compares. Assume that variables with the same names have the same type.

Figure 11.1 Comparison by the Positions of Observations

Data Set ONE

IDNUM	NAME	GENDER	GPA
2998	Bagwell	f	3.722
9866	Metcalf	m	3.342
2118	Gray	f	3.177
3847	Baglione	f	4.000
2342	Hall	m	3.574

Data Set TWO

IDNUM	NAME	GENDER	GPA	YEAR
2998	Bagwell	f	3.722	2
9866	Metcalf	m	3.342	2
2118	Gray	f	3.177	3
3847	Baglione	f	4.000	4
2342	Hall	m	3.574	4
7565	Gold	f	3.609	2
1755	Syme	f	3.883	3

When you use PROC COMPARE to compare data set TWO with data set ONE, the procedure compares the first observation in data set ONE with the first observation in data set TWO, and it compares the second observation in the first data set with the second observation in the second data set, and so on. In each observation that it compares, the procedure compares the values of the IDNUM, NAME, GENDER, and GPA.

The procedure does not report on the values of the last two observations or the variable YEAR in data set TWO because there is nothing to compare them with in data set ONE.

A Comparison with an ID Variable

In a simple comparison, PROC COMPARE uses the observation number to determine which observations to compare. When you use an ID variable, PROC COMPARE uses the values of the ID variable to determine which observations to compare. ID variables should have unique values and must have the same type.

For the two data sets shown in the following figure, assume that IDNUM is an ID variable and that IDNUM has the same type in both data sets. The procedure compares the observations that have the same value for IDNUM. The data inside the shaded boxes shows the part of the data sets that the procedure compares.

Figure 11.2 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 249 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, γ, is equal to the absolute value of the specified criterion multiplied by a very small number, ε (epsilon), that depends on the numerical precision of the computer. This number ε is defined as the smallest positive floating-point value such that, using machine arithmetic, $1-\varepsilon<1<1+\varepsilon$. Round-off or truncation error in floating-point computations is typically a few orders of magnitude larger than ε. CRITERION=–1000 often provides a reasonable test of the equality of computed results at the machine level of precision.

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

Specifying a value for δ avoids this extreme sensitivity of the RELATIVE method for small values. If you specify METHOD=RELATIVE(δ) CRITERION=γ when both x and y are much smaller than δ in absolute value, then the comparison is as if you had specified METHOD=ABSOLUTE CRITERION=$\delta\gamma$. 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=δ γ and METHOD=RELATIVE CRITERION=γ.

For character variables, if one value is longer than the other, then the shorter value is padded with blanks for the comparison. Nonblank character values are judged equal only if they agree at each character. If the NOMISSING option is in effect, then blank character values are judged equal to anything.

Definition of Difference and Percent Difference

In the reports of value comparisons and in the OUT= data set, PROC COMPARE displays difference and percent difference values for the numbers compared. These quantities are defined using the value from the base data set as the reference value. For a numeric variable compared, let x be its value in the base data set and let y be its value in the comparison data set. If x and y are both nonmissing, then the difference and percent difference are defined as follows:

Difference = $y - x$

Percent Difference = $(y - x)/x * 100 \quad \text{for} \quad x \neq 0$

Percent Difference = missing for $x = 0$.

How PROC COMPARE Handles Variable Formats

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

Results: COMPARE Procedure

Results Reporting

PROC COMPARE reports the results of its comparisons in the following ways:

- the SAS log
- return codes stored in the automatic macro SYSINFO
- procedure output
- output data sets.

SAS Log

When you use the WARNING, PRINTALL, or ERROR option, PROC COMPARE writes a description of the differences to the SAS log.

Macro Return Codes (SYSINFO)

PROC COMPARE stores a return code in the automatic macro variable SYSINFO. The value of the return code provides information about the result of the comparison. By checking the value of SYSINFO after PROC COMPARE has run and before any other step begins, SAS macros can use the results of a PROC COMPARE step to determine what action to take or what parts of a SAS program to execute.

The following table is a key for interpreting the SYSINFO return code from PROC COMPARE. For each of the conditions listed, the associated value is added to the return code if the condition is true. Thus, the SYSINFO return code is the sum of the codes listed in the following table for the applicable conditions:

Table 11.1 Macro Return Codes

Bit	Condition	Code	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 on, then use the following statements:

```
proc compare base=SAS-data-set
             compare=SAS-data-set;
run;
```

```
%if &sysinfo >= 64 %then
   %do;
      handle error;
   %end;
```

You can examine individual bits in the SYSINFO value by using DATA step bit-testing features to check for specific conditions. For example, to check for the presence of observations in the base data set that are not in the comparison data set, use the following statements:

```
proc compare base=SAS-data-set
             compare=SAS-data-set;
run;

%let rc=&sysinfo;
data _null_;
   if &rc='1......'b then
      put 'Observations in Base but not
           in Comparison Data Set';
run;
```

PROC COMPARE must run before you check SYSINFO and you must obtain the SYSINFO value before another SAS step starts because every SAS step resets SYSINFO.

Procedure Output

Procedure Output Overview

The following sections show and describe the default output of the two data sets shown in "Overview: COMPARE Procedure" on page 208. 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.

The following output shows the Data Set Summary.

Output 11.2 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.)

The following output shows the Variables Summary.

Output 11.3 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 different observations. Then, the report lists the following:

- the number of observations that the data sets have in common
- the number of observations in the base data set that are not in the comparison data set and vice versa
- the total number of observations in each data set
- the number of matching observations for which PROC COMPARE judged some variables unequal
- the number of matching observations for which PROC COMPARE judged all variables equal.

Output 11.4 shows the Observation Summary.

Output 11.4 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.

The following output shows the Values Comparison Summary.

Output 11.5 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).

The following output shows the Value Comparison Results for Variables.

Output 11.6 Partial Output

Value Comparison Results for Variables

Obs	Home State Base Value state	Compare Value state
2	MD	MA
4	MA	MD

Obs	Base gr1	Compare gr1	Diff.	% Diff
1	85.0	84.00	-1.0000	-1.1765
3	78.0	79.00	1.0000	1.2821

Obs	Base gr2	Compare gr2	Diff.	% Diff
3	72.0000	73.0000	1.0000	1.3889
4	94.0000	74.0000	-20.0000	-21.2766

You can suppress the value comparison results with the NOVALUES option. If you use both the NOVALUES and TRANSPOSE options, then PROC COMPARE lists for each observation the names of the variables with values judged unequal but does not display the values and differences.

Table of Summary Statistics

If you use the STATS, ALLSTATS, or PRINTALL option, then the Value Comparison Results for Variables section contains summary statistics for the numeric variables that are being compared. The STATS option generates these statistics for only the numeric variables whose values are judged unequal. The ALLSTATS and PRINTALL options generate these statistics for all numeric variables, even if all values are judged equal.

Note: In all cases PROC COMPARE calculates the summary statistics based on all matching observations that do not contain missing values, not just on those containing unequal values. △

The following output shows the following summary statistics for base data set values, comparison data set values, differences, and percent differences:

N
the number of nonmissing values

MEAN
the mean, or average, of the values

STD
the standard deviation

MAX

the maximum value

MIN
the minimum value

MISSDIFF
the number of missing values in either a base or compare data set

STDERR
the standard error of the mean

T
the T ratio (MEAN/STDERR)

PROB> | T |
the probability of a greater absolute T value if the true population mean is 0.

NDIF
the number of matching observations judged unequal, and the percent of the matching observations that were judged unequal.

DIFMEANS
the difference between the mean of the base values and the mean of the comparison values. This line contains three numbers. The first is the mean expressed as a percentage of the base values mean. The second is the mean expressed as a percentage of the comparison values mean. The third is the difference in the two means (the comparison mean minus the base mean).

R
the correlation of the base and comparison values for matching observations that are nonmissing in both data sets.

RSQ
the square of the correlation of the base and comparison values for matching observations that are nonmissing in both data sets.

The following output is from the ALLSTATS option using the two data sets shown in "Overview":

Output 11.7 Partial Output

Value Comparison Results for Variables

Obs	Base gr1	Compare gr1	Diff.	% Diff
1	85.0	84.00	-1.0000	-1.1765
3	78.0	79.00	1.0000	1.2821
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		

Obs	Base gr2	Compare gr2	Diff.	% Diff
3	72.0000	73.0000	1.0000	1.3889
4	94.0000	74.0000	-20.0000	-21.2766
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.

The following output shows the differences in PROCLIB.ONE and PROCLIB.TWO by observation instead of by variable.

Output 11.8 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 11.2 ODS Tables Produced by the COMPARE Procedure

Table Name	Description	Generated...
CompareData sets	Information about the data set or data sets	by default, unless NOSUMMARY or NOVALUES option is specified
CompareDetails (Comparison Results for Observations)	A listing of observations that the base data set and the compare data set do not have in common	if PRINTALL option is specified

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 253.

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 233). 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 255 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 208

This example shows the most complete report that PROC COMPARE produces as procedure output.

Program

Declare the PROCLIB SAS library.

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

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=40;
```

Create a complete report of the differences between two data sets. BASE= and COMPARE= specify the data sets to compare. PRINTALL prints a full report of the differences.

```
proc compare base=proclib.one compare=proclib.two printall;
   title 'Comparing Two Data Sets: Full Report';
run;
```

Output: Listing

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 233.

```
                    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

Obs	Year of Birth Base Value year	Compare Value year
1	1970	1970
2	1971	1971
3	1969	1969
4	1970	1970

Obs	Home State Base Value state	Compare Value state
1	NC	NC
2	MD	MA
3	PA	PA
4	MA	MD

Comparing Two Data Sets: Full Report 5

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

Value Comparison Results for Variables

Obs	Base gr1	Compare gr1	Diff.	% Diff
1	85.0	84.00	-1.0000	-1.1765
2	92.0	92.00	0	0
3	78.0	79.00	1.0000	1.2821
4	87.0	87.00	0	0
>				
N	4	4	4	4
Mean	85.5000	85.5000	0	0.0264
Std	5.8023	5.4467	0.8165	1.0042
Max	92.0000	92.0000	1.0000	1.2821
Min	78.0000	79.0000	-1.0000	-1.1765
StdErr	2.9011	2.7234	0.4082	0.5021
t	29.4711	31.3951	0.0000	0.0526
Prob>|t|	<.0001	<.0001	1.0000	0.9614
Ndif	2	50.000%		
DifMeans	0.000%	0.000%	0	
r, rsq	0.991	0.983		

```
                   Comparing Two Data Sets: Full Report                    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 208.

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 library.

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

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=40;
```

Suppress all summary reports of the differences between two data sets. BASE= specifies the base data set and COMPARE= specifies the comparison data set. NOSUMMARY suppresses all summary reports.

```
proc compare base=proclib.one compare=proclib.two nosummary;
```

Specify one variable from the base data set to compare with one variable from the comparison data set. The VAR and WITH statements specify the variables to compare. This example compares GR1 from the base data set with GR2 from the comparison data set.

```
   var gr1;
   with gr2;
   title 'Comparison of Variables in Different Data Sets';
run;
```

Output: Listing

```
                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 208.

This example compares one variable from the base data set with two variables in the comparison data set.

Program

Declare the PROCLIB SAS library.

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

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=40;
```

Suppress all summary reports of the differences between two data sets. BASE= specifies the base data set and COMPARE= specifies the comparison data set. NOSUMMARY suppresses all summary reports.

```
proc compare base=proclib.one compare=proclib.two nosummary;
```

Specify one variable from the base data set to compare with two variables from the comparison data set. The VAR and WITH statements specify the variables to compare. This example compares GR1 from the base data set with GR1 and GR2 from the comparison data set.

```
   var gr1 gr1;
   with gr1 gr2;
   title 'Comparison of One Variable with Two Variables';
run;
```

Output: Listing

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 208.

This example shows that PROC COMPARE can compare two variables that are in the same data set.

Program

Declare the PROCLIB SAS library.

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

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=40;
```

Create a short summary report of the differences within one data set. ALLSTATS prints summary statistics. BRIEFSUMMARY prints only a short comparison summary.

```
proc compare base=proclib.one allstats briefsummary;
```

Specify two variables from the base data set to compare. The VAR and WITH statements specify the variables in the base data set to compare. This example compares GR1 with GR2. Because there is no comparison data set, the variables GR1 and GR2 must be in the base data set.

```
   var gr1;
   with gr2;
   title 'Comparison of Variables in the Same Data Set';
run;
```

Output: Listing

```
                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 library.

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

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=40;
```

Create the PROCLIB.EMP95 and PROCLIB.EMP96 data sets. PROCLIB.EMP95 and PROCLIB.EMP96 contain employee data. IDNUM works well as an ID variable because it has unique values. A DATA step on page 1614 creates PROCLIB.EMP95. A DATA step on page 1615 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: Listing

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 253.

```
               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 250

This example creates and prints an output data set that shows the differences between matching observations.

In Example 5 on page 249, 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 library.

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

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=120 pagesize=40;
```

Sort the data sets by the ID variable. Both data sets must be sorted by the variable that will be used as the ID variable in the PROC COMPARE step. OUT= specifies the location of the sorted data.

```
proc sort data=proclib.emp95 out=emp95_byidnum;

 by idnum;
run;

proc sort data=proclib.emp96 out=emp96_byidnum;
   by idnum;
run;
```

Specify the data sets to compare. BASE= and COMPARE= specify the data sets to compare.

```
proc compare base=emp95_byidnum compare=emp96_byidnum
```

Create the output data set RESULT and include all unequal observations and their differences. OUT= names and creates the output data set. NOPRINT suppresses the printing of the procedure output. OUTNOEQUAL includes only observations that are judged unequal. OUTBASE writes an observation to the output data set for each observation in the base data set. OUTCOMP writes an observation to the output data set for each observation in the comparison data set. OUTDIF writes an observation to the output data set that contains the differences between the two observations.

```
              out=result outnoequal outbase outcomp outdif
     noprint;
```

Specify the ID variable. The ID statement specifies IDNUM as the ID variable.

```
   id idnum;
run;
```

Print the output data set RESULT and use the BY and ID statements with the ID variable. PROC PRINT prints the output data set. Using the BY and ID statements with the same variable makes the output easy to read. See Chapter 43, "The PRINT Procedure," on page 815 for more information on this technique.

```
proc print data=result noobs;
   by idnum;
   id idnum;
   title 'The Output Data Set RESULT';
run;
```

Output: Listing

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	.....	...XXXXXXXXXXXXX.XXXXX.XXXXXXXXXX.....	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 250

This example creates an output data set that contains summary statistics for the numeric variables that are compared.

Program

Declare the PROCLIB SAS library.

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

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=40;
```

Sort the data sets by the ID variable. Both data sets must be sorted by the variable that will be used as the ID variable in the PROC COMPARE step. OUT= specifies the location of the sorted data.

```
proc sort data=proclib.emp95 out=emp95_byidnum;
   by idnum;
run;

proc sort data=proclib.emp96 out=emp96_byidnum;
   by idnum;
run;
```

Create the output data set of statistics and compare observations that have matching values for the ID variable. BASE= and COMPARE= specify the data sets to compare. OUTSTATS= creates the output data set DIFFSTAT. NOPRINT suppresses the procedure output. The ID statement specifies IDNUM as the ID variable. PROC COMPARE uses the values of IDNUM to match observations.

```
proc compare base=emp95_byidnum compare=emp96_byidnum
             outstats=diffstat noprint;
   id idnum;
run;
```

Print the output data set DIFFSTAT. PROC PRINT prints the output data set DIFFSTAT.

```
proc print data=diffstat noobs;
   title 'The DIFFSTAT Data Set';
```

```
run;
```

Output: Listing

The variables are described in "Output Statistics Data Set (OUTSTATS=)" on page 238.

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
12

The CONTENTS Procedure

Overview: CONTENTS Procedure **259**
Syntax: CONTENTS Procedure **259**

Overview: CONTENTS Procedure

The CONTENTS procedure shows the contents of a SAS data set and prints the directory of the SAS library.

Generally, the CONTENTS procedure functions the same as the CONTENTS statement in the DATASETS procedure. The differences between the CONTENTS procedure and the CONTENTS statement in PROC DATASETS are as follows:

- The default for *libref* in the DATA= option in PROC CONTENTS is WORK. For the CONTENTS statement, the default is the libref of the procedure input library.
- PROC CONTENTS can read sequential files. The CONTENTS statement cannot.

Syntax: CONTENTS Procedure

Restriction: You cannot use the WHERE option to affect the output because PROC CONTENTS does not process any observations.

Tip:

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

Tip: You can use data set options with the DATA=, OUT=, and OUT2= options.

Tip: Complete documentation for the CONTENTS statement and the CONTENTS procedure is in “CONTENTS Statement” on page 314.

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

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

Note: The links in the following table are to the DATASETS procedure documentation, which explains these options.
△

Task	Option
List the contents of one or more SAS data sets and print the directory of the SAS library	"CONTENTS Statement" on page 314
Print centiles information for indexed variables	CENTILES on page 315
Specify the input data set	DATA= on page 315
Include information in the output about the number of observations, number of variables, number of indexes, and data set labels	DETAILS|NODETAILS on page 316
Print a list of the SAS files in the SAS library	DIRECTORY on page 316
Print the length of a variable's informat or format	FMTLEN on page 316
Restrict processing to one or more types of SAS files	MEMTYPE= on page 316
Suppress the printing of individual files	NODS on page 317
Suppress the printing of the output	NOPRINT on page 317
Print a list of variables in various order	ORDER= on page 317
Specify the name for an output data set	OUT= on page 317
Specify the name of an output data set to contain information about indexes and integrity constraints	OUT2= on page 317
Print abbreviated output	SHORT on page 317
Print a list of the variables by their position in the data set. By default, the CONTENTS statement lists the variables alphabetically.	VARNUM on page 318

Note: The ORDER= option does not affect the order of the OUT= and OUT2= data sets. △

CHAPTER
13

The COPY Procedure

Overview: COPY Procedure **261**
Syntax: COPY Procedure **261**
Concepts: COPY Procedure **262**
Transporting SAS Data Sets between Hosts **262**
Example: COPY Procedure **263**
Example 1: Copying SAS Data Sets between Hosts **263**

Overview: COPY Procedure

The COPY procedure copies one or more SAS files from a SAS library.

Generally, the COPY procedure functions the same as the COPY statement in the DATASETS procedure. The two differences are as follows:

- The IN= argument is required with PROC COPY. In the COPY statement, IN= is optional. If IN= is omitted, the default value is the libref of the procedure input library.
- PROC DATASETS cannot work with libraries that allow only sequential data access.

Note: The MIGRATE procedure is available specifically for migrating a SAS library from a previous release to the most recent release. For migration, PROC MIGRATE offers benefits that PROC COPY does not. For more information on PROC MIGRATE, see Chapter 37, "The MIGRATE Procedure," on page 683 . △

Syntax: COPY Procedure

Restriction: PROC COPY ignores concatenations with catalogs. Use PROC CATALOG COPY to copy concatenated catalogs.

Restriction: PROC COPY does not support data set options.

Tip:

Tip: Complete documentation for the COPY statement and the COPY procedure is in "COPY Statement" on page 318.

PROC COPY OUT=*libref-1* IN=*libref-2*
<CLONE|NOCLONE>
<CONSTRAINT=YES|NO>

<DATECOPY>
<INDEX=YES|NO>
<MEMTYPE=(*mtype-1* <...*mtype-n*>)>
<MOVE <ALTER=*alter-password*>>;

EXCLUDE *SAS-file-1* <...*SAS-file-n*> </ MEMTYPE=*mtype*>;

SELECT *SAS-file-1* <...*SAS-file-n*> </ <MEMTYPE=*mtype*>
<ALTER=*alter-password*>>;

Note: The links in the following table are to the DATASETS procedure documentation, which explains these options.
△

Task	Option
Copy one or more files	"COPY Statement" on page 318
Exclude files or memtypes	EXCLUDE"EXCLUDE Statement" on page 333
Name of source library	IN= on page 319(required)
Name of destination library	OUT= on page 319 (required)
Select files or memtypes	SELECT"SELECT Statement" on page 352

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 library.

For an example, see Example 1 on page 263.

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 10, "The CIMPORT Procedure," on page 191 and Chapter 15, "The CPORT Procedure," on page 269.

If you need to migrate a SAS library from a previous release of SAS, see the Migration Community at http://support.sas.com/migration.

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 library that contains the SAS data set that you want to transport. Also, assign a libref to the transport file and use the XPORT keyword to specify the XPORT engine.

```
LIBNAME source 'SAS-library-on-sending-host';
LIBNAME xptout xport 'filename-on-sending-host';
```

Copy the SAS data sets to the transport file. 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-library-on-sending-host ';
NOTE: Libref SOURCE was successfully assigned as follows:
      Engine:        V9
      Physical Name: SAS-library-on-sending-host
2    LIBNAME xptout xport 'filename-on-sending-host';
NOTE: Libref XPTOUT was successfully assigned as follows:
      Engine:        XPORT
      Physical Name: filename-on-sending-host
3    proc copy in=source out=xptout memtype=data;
4    select bonus budget salary;
5    run;

NOTE: Copying SOURCE.BONUS to XPTOUT.BONUS (memtype=DATA).
NOTE: The data set XPTOUT.BONUS has 1 observations and 3 variables.
NOTE: Copying SOURCE.BUDGET to XPTOUT.BUDGET (memtype=DATA).
NOTE: The data set XPTOUT.BUDGET has 1 observations and 3 variables.
NOTE: Copying SOURCE.SALARY to XPTOUT.SALARY (memtype=DATA).
NOTE: The data set XPTOUT.SALARY has 1 observations 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 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
14

The CORR Procedure

Information about the CORR Procedure **267**

Information about the CORR Procedure

See: The documentation for the CORR procedure has moved to the *Base SAS Procedures Guide: Statistical Procedures*.

CHAPTER

15

The CPORT Procedure

Overview: CPORT Procedure **269**
Purpose of the CPORT Procedure **269**
Process for Creating and Reading a Transport File **270**
Syntax: CPORT Procedure **270**
PROC CPORT Statement **271**
EXCLUDE Statement **276**
SELECT Statement **277**
TRANTAB Statement **278**
READ= Data Set Option in the PROC CPORT Statement **278**
Results: CPORT Procedure **279**
Examples: CPORT Procedure **279**
Example 1: Exporting Multiple Catalogs **279**
Example 2: Exporting Individual Catalog Entries **280**
Example 3: Exporting a Single SAS Data Set **281**
Example 4: Applying a Translation Table **282**
Example 5: Exporting Entries Based on Modification Date **283**

Overview: CPORT Procedure

Purpose of the CPORT Procedure

The CPORT procedure writes SAS data sets, SAS catalogs, or SAS libraries to sequential file formats (transport files). Use PROC CPORT with the CIMPORT procedure to move files from one environment to another. *Transport files* are sequential files that each contain a SAS library, a SAS catalog, or a SAS data set in transport format. The transport format that PROC CPORT writes is the same for all environments and for many releases of SAS. In PROC CPORT, *export* means to put a SAS library, a SAS catalog, or a SAS data set into transport format. PROC CPORT exports catalogs and data sets, either singly or as a SAS library. PROC CIMPORT restores (*imports*) the transport file to its original form as a SAS catalog, SAS data set, or SAS library.

PROC CPORT also *converts* SAS files, which means that it changes the format of a SAS file from the format appropriate for one version of SAS to the format appropriate for another version. For example, you can use PROC CPORT and PROC CIMPORT to move files from earlier releases of SAS to more recent releases. PROC CIMPORT automatically converts the transport file as it imports it.

PROC CPORT produces no output (other than the transport files), but it does write notes to the SAS log.

Process for Creating and Reading a Transport File

Here is the process to create a transport file at the source computer and to read it at a target computer:

1. A transport file is created at the source computer using PROC CPORT.
2. The transport file is transferred from the source computer to the target computer via communications software or a magnetic medium.
3. The transport file is read at the target computer using PROC CIMPORT.

Note: Transport files that are created using PROC CPORT are not interchangeable with transport files that are created using the XPORT engine. △

For complete details about the steps to create a transport file (PROC CPORT), to transfer the transport file, and to restore the transport file (PROC CIMPORT), see *Moving and Accessing SAS Files*.

Syntax: CPORT Procedure

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)*>;

Task	Statement
Creates a transport file	"PROC CPORT Statement" on page 271
Excludes one or more specified files from the transport file.	"EXCLUDE Statement" on page 276
Specifies one or more files or entries to include in the transport file.	"SELECT Statement" on page 277
Specifies one or more translation tables for characters in catalog entries to be exported.	"TRANTAB Statement" on page 278

PROC CPORT Statement

PROC CPORT *source-type=libref | <libref.>member-name<option(s)>*;

Task	Option
Identify the transport file	
Specify the transport file to write to	FILE=
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

Task	Option
Specify that exported catalog entries contain compiled SCL code, but not the source code	NOSRC
Specify a libref associated with a SAS 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 library to export.

source-type
identifies one or more files to export as a single catalog, as a single SAS data set, or as the members of a SAS library. The *source-type* argument can be one of the following:

CATALOG | CAT | C

DATA | DS | D

LIBRARY | LIB | L

Note: If you specify a password-protected data set as the source type, you must also include the password when creating its transport file. For details, see "READ= Data Set Option in the PROC CPORT Statement" on page 278. △

libref | <libref.>member-name
specifies the specific catalog, SAS data set, or SAS library to export. If *source-type* is CATALOG or DATA, you can specify both a libref and a member name. If the *libref* is omitted, PROC CPORT uses the default library as the *libref*, which is usually the WORK library. If the *source-type* argument is LIBRARY, specify only a *libref*. If you specify a library, PROC CPORT exports only data sets and catalogs from that library. You cannot export other types of files.

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 283.

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 CONSTRAINT=NO, indexes created without integrity constraints are ported, but neither integrity constraints nor any indexes created with integrity constraints are ported. For more information 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 342 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

Default: If the INTYPE= option is not used, the DBCS type defaults to the value of the SAS system option DBCSTYPE=.

Restriction The INTYPE= option is allowed only if SAS is built with Double-Byte Character Set (DBCS) extensions. Because these extensions require significant computing resources, there is a special distribution for those sites that require it. An error is reported if this option is used at a site for which DBCS extensions are not enabled.

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 279.

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 library. If you specify the OUTLIB= option, PROC CIMPORT is invoked automatically to re-create the input library, data set, or catalog in the specified library.

Alias: OUT=

Tip: Use the OUTLIB= option when you change SAS files from one DBCS type to another within the same operating environment if you want to keep the original data intact.

OUTTYPE=UPCASE
writes all displayed characters to the transport file and to the OUTLIB= file in uppercase.

Interaction: The OUTTYPE= option invokes the NOCOMPRESS option.

TAPE
directs the output from PROC CPORT to a tape.

Default: The output from PROC CPORT is sent to disk.

TRANSLATE=(*translation-list*)
translates specified characters from one ASCII or EBCDIC value to another. Each element of *translation-list* has the form

ASCII-value-1 TO *ASCII-value-2*

EBCDIC-value-1 TO *EBCDIC-value-2*

You can use hexadecimal or decimal representation for ASCII values. If you use the hexadecimal representation, values must begin with a digit and end with an x. Use a leading zero if the hexadecimal value begins with an alphabetic character.

For example, to translate all left brackets to left braces, specify the TRANSLATE= option as follows (for ASCII characters):

```
translate=(5bx to 7bx)
```

The following example translates all left brackets to left braces and all right brackets to right braces:

```
translate=(5bx to 7bx 5dx to 7dx)
```

EXCLUDE Statement

Excludes specified files or entries from the transport file.

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

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

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

Required Arguments

SAS file(s) | catalog entry(s)
specifies one or more SAS files or one or more catalog entries to be excluded from the transport file. Specify SAS filenames when you export a SAS library; specify catalog entry names when you export an individual SAS catalog. Separate multiple filenames or entry names with a space. You can use shortcuts to list many like-named files in the EXCLUDE statement. For more information, see "Shortcuts for Specifying Lists of Variable Names" on page 25.

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.

Alias: ETYPE=, ET=

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

MEMTYPE=*mtype*
specifies a single member type for one or more SAS files listed in the EXCLUDE statement. Valid values are CATALOG or CAT, DATA, or ALL. If you do not specify the MEMTYPE= option in the EXCLUDE statement, then processing is restricted to those member types specified in the MEMTYPE= option in the PROC CPORT statement.

You can also specify the MEMTYPE= option, enclosed in parentheses, immediately after the name of a file. In parentheses, MEMTYPE= identifies the type of the

filename that just precedes it. When you use this form of the option, it overrides the MEMTYPE= option that follows the slash in the EXCLUDE statement, but it must match the MEMTYPE= option in the PROC CPORT statement:

Alias: MTYPE=, MT=

Default: If you do not specify MEMTYPE= in the PROC CPORT statement or in the EXCLUDE statement, the default is MEMTYPE=ALL.

Restriction: MEMTYPE= is valid only when you export a SAS 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.

SELECT Statement

Includes specified files or entries in the transport file.

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

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

Featured in: Example 2 on page 280

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

Required Arguments

SAS file(s) | catalog entry(s)
specifies one or more SAS files or one or more catalog entries to be included in the transport file. Specify SAS filenames when you export a SAS library; specify catalog entry names when you export an individual SAS catalog. Separate multiple filenames or entry names with a space. You can use shortcuts to list many like-named files in the SELECT statement. For more information, see "Shortcuts for Specifying Lists of Variable Names" on page 25.

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.

Alias: ETYPE=, ET=

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

MEMTYPE=*mtype*
specifies a single member type for one or more SAS files listed in the SELECT statement. Valid values are CATALOG or CAT, DATA, or ALL. If you do not specify

the MEMTYPE= option in the SELECT statement, then processing is restricted to those member types specified in the MEMTYPE= option in the PROC CPORT statement.

You can also specify the MEMTYPE= option, enclosed in parentheses, immediately after the name of a member. In parentheses, MEMTYPE= identifies the type of the member name that just precedes it. When you use this form of the option, it overrides the MEMTYPE= option that follows the slash in the SELECT statement, but it must match the MEMTYPE= option in the PROC CPORT statement.

Alias: MTYPE=, MT=

Default: If you do not specify MEMTYPE= in the PROC CPORT statement or in the SELECT statement, the default is MEMTYPE=ALL.

Restriction: MEMTYPE= is valid only when you export a SAS 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.

TRANTAB Statement

Specifies translation tables for characters in catalog entries that you export.

Tip: You can specify only one translation table for each TRANTAB statement. However, you can use more than one translation table in a single invocation of PROC CPORT.

See: The TRANTAB Statement for the CPORT Procedure and the UPLOAD and DOWNLOAD Procedures in *SAS National Language Support (NLS): Reference Guide*

Featured in: Example 4 on page 282.

TRANTAB NAME=*translation-table-name*
<*option(s)*>;

READ= Data Set Option in the PROC CPORT Statement

To be authorized to create the transport file for a read-protected data set, you must include the password (clear-text or encoded). If the password is not included, the transport file cannot be created.

You use the READ= data set option to include the appropriate password for the read-protected data set when creating a transport file.

Example 1: Clear-Text Password:

```
proc cport data=source.grades(read=admin) file=gradesout;
```

PROC CPORT copies the input file that is named SOURCE.GRADES, includes the password ADMIN with the data set, and creates the transport file named GRADESOUT as the output.

Example 2: Encoded Password

```
proc cport data=source.grades(read={sas003}6EDB396015B96DBD9E80F0913A543819A8E5)
   file=gradesout;
```

An encoded password is generated via the PWENCODE procedure. For details, see Chapter 48, "The PWENCODE Procedure," on page 937.

If the password is omitted from PROC CPORT, SAS prompts for the password. If an invalid password is specified, an error message is sent to the log. Here is an example:

```
ERROR: Invalid or missing READ password on member WORK.XYZ.DATA
```

If the data set is transported as part of a library, a password is not required. If the target SAS engine does not support passwords, the transport file cannot be imported.

For details about the READ= data set option, see *SAS Language Reference: Dictionary*, and for details about password-protected data sets, see *SAS Language Reference: Concepts*.

Note: PROC CIMPORT does not require a password in order to restore the transport file in the target environment. However, other SAS procedures that use the password-protected data set must include the password. △

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 library you specify.

Program

Specify the library reference for the SAS library that contains the source files to be exported and the file reference to which the output transport file is written. The LIBNAME statement assigns a libref for the SAS library. The FILENAME statement assigns a fileref and any operating environment options for file characteristics for the transport file that PROC CPORT creates.

```
libname source 'SAS-data-library';
filename tranfile 'transport-file'
                  host-option(s)-for-file-characteristics;
```

Create the transport file. The PROC CPORT step executes on the operating environment where the source library is located. MEMTYPE=CATALOG writes all SAS catalogs in the source library to the transport file.

```
proc cport library=source file=tranfile memtype=catalog;
run;
```

SAS Log

```
NOTE: Proc CPORT begins to transport catalog SOURCE.FINANCE
NOTE: The catalog has 5 entries and its maximum logical record length is 866.
NOTE: Entry LOAN.FRAME has been transported.
NOTE: Entry LOAN.HELP has been transported.
NOTE: Entry LOAN.KEYS has been transported.
NOTE: Entry LOAN.PMENU has been transported.
NOTE: Entry LOAN.SCL has been transported.

NOTE: Proc CPORT begins to transport catalog SOURCE.FORMATS
NOTE: The catalog has 2 entries and its maximum logical record length is 104.
NOTE: Entry REVENUE.FORMAT has been transported.
NOTE: Entry DEPT.FORMATC has been transported.
```

Example 2: Exporting Individual Catalog Entries

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
16

The CV2VIEW Procedure

Information about the CV2VIEW Procedure **285**

Information about the CV2VIEW Procedure

See: For complete documentation of the CV2VIEW procedure, see *SAS/ACCESS for Relational Databases: Reference*.

CHAPTER

17

The DATASETS Procedure

Overview: DATASETS Procedure **288**
What Does the DATASETS Procedure Do? **288**
Sample PROC DATASETS Output **289**
Notes **290**
Syntax: DATASETS Procedure **291**
PROC DATASETS Statement **296**
AGE Statement **300**
APPEND Statement **302**
ATTRIB Statement **309**
AUDIT Statement **310**
CHANGE Statement **312**
CONTENTS Statement **314**
COPY Statement **318**
DELETE Statement **328**
EXCHANGE Statement **332**
EXCLUDE Statement **333**
FORMAT Statement **333**
IC CREATE Statement **334**
IC DELETE Statement **337**
IC REACTIVATE Statement **337**
INDEX CENTILES Statement **338**
INDEX CREATE Statement **339**
INDEX DELETE Statement **340**
INFORMAT Statement **341**
LABEL Statement **341**
MODIFY Statement **342**
REBUILD Statement **347**
RENAME Statement **348**
REPAIR Statement **349**
SAVE Statement **351**
SELECT Statement **352**
Concepts: DATASETS Procedure **353**
Procedure Execution **353**
Execution of Statements **353**
RUN-Group Processing **353**
Error Handling **355**
Password Errors **355**
Forcing a RUN Group with Errors to Execute **355**
Ending the Procedure **355**
Using Passwords with the DATASETS Procedure **355**
Restricting Member Types for Processing **356**

In the PROC DATASETS Statement **356**
In Subordinate Statements **356**
Member Types **357**
Restricting Processing for Generation Data Sets **358**
Results: DATASETS Procedure **359**
Directory Listing to the SAS Log **359**
Directory Listing as SAS Output **360**
Procedure Output **360**
The CONTENTS Statement **360**
Data Set Attributes **360**
Engine and Operating Environment-Dependent Information **361**
Alphabetic List of Variables and Attributes **362**
Alphabetic List of Indexes and Attributes **363**
Sort Information **364**
PROC DATASETS and the Output Delivery System (ODS) **364**
ODS Table Names **365**
Output Data Sets **366**
The CONTENTS Statement **366**
The OUT= Data Set **366**
The OUT2= Data Set **371**
Examples: DATASETS Procedure **372**
Example 1: Removing All Labels and Formats in a Data Set **372**
Example 2: Manipulating SAS Files **374**
Example 3: Saving SAS Files from Deletion **380**
Example 4: Modifying SAS Data Sets **382**
Example 5: Describing a SAS Data Set **384**
Example 6: Concatenating Two SAS Data Sets **386**
Example 7: Aging SAS Data Sets **388**
Example 8: ODS Output **389**
Example 9: Getting Sort Indicator Information **392**
Example 10: Using the ORDER= Option with the CONTENTS Statement **394**

Overview: DATASETS Procedure

What Does the DATASETS Procedure Do?

The DATASETS procedure is a utility procedure that manages your SAS files. With PROC DATASETS, you can do the following:

- copy SAS files from one SAS library to another
- rename SAS files
- repair SAS files
- delete SAS files
- list the SAS files that are contained in a SAS library
- list the attributes of a SAS data set, such as:
  - the date when the data was last modified
  - whether the data is compressed
  - whether the data is indexed

- manipulate passwords on SAS files
- append SAS data sets
- modify attributes of SAS data sets and variables within the data sets
- create and delete indexes on SAS data sets
- create and manage audit files for SAS data sets
- create and delete integrity constraints on SAS data sets

Sample PROC DATASETS Output

The following DATASETS procedure includes the following:

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 the following output.

```
LIBNAME control 'SAS-library-1';
LIBNAME health 'SAS-library-2';


proc datasets memtype=data;
   copy in=control out=health;
run;

proc datasets library=health memtype=data details;
   delete syndrome;
   change prenat=infant;
run;
quit;
```

Output 17.1 Log from PROC DATASETS

```
148  proc datasets library=health memtype=data details;

                               Directory

    Libref         HEALTH
    Engine         V9
    Physical Name  SAS library 2
    File Name      SAS library 2

                          Member  Obs, Entries                                 File
               #  Name     Type     or Indexes   Vars  Label                   Size  Last Modified

               1  ALL      DATA          23        17                        13312  12Sep07:13:57:48
               2  BODYFAT  DATA           1         2                         5120  12Sep07:13:57:48
               3  CONFOUND DATA           8         4                         5120  12Sep07:13:57:48
               4  CORONARY DATA          39         4                         5120  12Sep07:13:57:48
               5  DRUG1    DATA           6         2  JAN2005 DATA           5120  12Sep07:13:57:49
               6  DRUG2    DATA          13         2  MAY2005 DATA           5120  12Sep07:13:57:49
               7  DRUG3    DATA          11         2  JUL2005 DATA           5120  12Sep07:13:57:49
               8  DRUG4    DATA           7         2  JAN2002 DATA           5120  12Sep07:13:57:49
               9  DRUG5    DATA           1         2  JUL2002 DATA           5120  12Sep07:13:57:49
              10  GROUP    DATA         148        11                        25600  12Sep07:13:57:50
              11  GRPOUT   DATA          11        40                        17408  24Mar05:15:33:31
              12  MLSCL    DATA          32         4  Multiple Sclerosis Data 5120  12Sep07:13:57:50
              13  NAMES    DATA           7         4                         5120  12Sep07:13:57:50
              14  OXYGEN   DATA          31         7                         9216  12Sep07:13:57:50
              15  PERSONL  DATA         148        11                        25600  12Sep07:13:57:51
              16  PHARM    DATA           6         3  Sugar Study            5120  12Sep07:13:57:51
              17  POINTS   DATA           6         6                         5120  12Sep07:13:57:51
              18  PRENAT   DATA         149         6                        17408  12Sep07:13:57:51
              19  RESULTS  DATA          10         5                         5120  12Sep07:13:57:52
              20  SLEEP    DATA         108         6                         9216  12Sep07:13:57:52
              21  SYNDROME DATA          46         8                         9216  12Sep07:13:57:52
              22  TENSION  DATA           4         3                         5120  12Sep07:13:57:52
              23  TEST2    DATA          15         5                         5120  12Sep07:13:57:52
              24  TRAIN    DATA           7         2                         5120  12Sep07:13:57:53
              25  VISION   DATA          16         3                         5120  12Sep07:13:57:53
              26  WEIGHT   DATA          83        13  California Results     13312  12Sep07:13:57:53
              27  WGHT     DATA          83        13                        13312  12Sep07:13:57:53
6  !                                              delete syndrome;    change
7    prenat=infant; run;

NOTE: Deleting HEALTH.SYNDROME (memtype=DATA).
NOTE: Changing the name HEALTH.PRENAT to HEALTH.INFANT (memtype=DATA).
7  !                     quit;

NOTE: PROCEDURE DATASETS used (Total process time):
      real time           17.10 seconds
      cpu time            0.15 seconds
```

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 131.
- 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.

- You cannot change the length of a variable using the LENGTH statement or the LENGTH= option on an ATTRIB statement.
- There can be a discrepancy between the modified date in PROC DATASETS, PROC CONTENTS, and other components of SAS, such as SAS Explorer. The two modified dates and times are distinctly different:
  - Operating-environment modified date and time is reported by the SAS Explorer and the PROC DATASETS LIST option.
  - The modified date and time reported by the CONTENTS statement is the date and time that the data within the data set was actually modified.
- If you have a library containing a large number of members, the DATASETS procedure might show an increase in process time. You might want to reorganize your library into smaller libraries for better performance.

Syntax: DATASETS Procedure

Tip: Supports RUN-group processing.

Tip:

Tip:

See: DATASETS Procedure OpenVMS in the documentation for your operating environment.

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

AGE *current-name related-SAS-file-1* <*...current-name related-SAS-file-n*>
</ <ALTER=*alter-password*>
<MEMTYPE=*mtype*>>;

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

AUDIT *SAS-file* <(*SAS-password*)>;
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-1* <*...option-n*>>;

COPY OUT=*libref-1*

<CLONE|NOCLONE>
<CONSTRAINT=YES|NO>
<DATECOPY>
<FORCE>
<IN=*libref-2*>
<INDEX=YES|NO>
<MEMTYPE=(*mtype-1 <...mtype-n>*)>
<MOVE <ALTER=*alter-password*>>;
EXCLUDE *SAS-file-1 <...SAS-file-n>* < / MEMTYPE=*mtype*>;
SELECT *SAS-file-1 <...SAS-file-n>*
</ <ALTER=*alter-password*>
<MEMTYPE= *mtype*>>;

DELETE *SAS-file-1 <...SAS-file-n>*
</ <ALTER=*alter-password*>
<GENNUM=ALL|HIST|REVERT|*integer*>
<MEMTYPE=*mtype*>>;

EXCHANGE *name-1=other-name-1*
<*…name-n=other-name-n*>
</ <ALTER=*alter-password*>
<MEMTYPE=*mtype*> >;

MODIFY *SAS-file* <(*option-1 <...option-n>*)>
</ <CORRECTENCODING=*encoding-value*>
<DTC=*SAS-date-time*>
<GENNUM=*integer*>
<MEMTYPE=*mtype*>>;
ATTRIB *variable list(s) attribute list(s)*;
FORMAT *variable-1 <format-1>*
<*…variable-n <format-n>*>;
IC CREATE <*constraint-name=*> *constraint*
<MESSAGE='*message-string*' <MSGTYPE=USER>>;
IC DELETE *constraint-name-1 <...constraint-name-n>*| _ALL_;
IC REACTIVATE *foreign-key-name* REFERENCES *libref*;
INDEX CENTILES *index-1 <...index-n>*
</ <REFRESH>
<UPDATECENTILES= ALWAYS|NEVER|*integer*>>;
INDEX CREATE *index-specification-1 <...index-specification-n>*
</ <NOMISS>
<UNIQUE>
<UPDATECENTILES=ALWAYS|NEVER|*integer*>>;
INDEX DELETE *index-1 <...index-n>* | _ALL_;
INFORMAT *variable-1 <informat-1>*
<*…variable-n <informat-n>*>;
LABEL *variable-1=<'label-1'|' '>*
<*…variable-n=<'label-n'|' '* >>;
RENAME *old-name-1=new-name-1*
<*…old-name-n=new-name-n*>;

REBUILD *SAS-file* </ ALTER=*password* GENNUM=*n* MEMTYPE=*mtype*
NOINDEX>;

REPAIR *SAS-file-1 <...SAS-file-n>*
</ <ALTER=*alter-password*>
<GENNUM=*integer*>
<MEMTYPE=*mtype*>>;

SAVE *SAS-file-1 <...SAS-file-n>* </ MEMTYPE=*mtype*>;

Task	Statement
Manage SAS files	"PROC DATASETS Statement" on page 296
Rename a group of related SAS files	"AGE Statement" on page 300
Add observations from one SAS data set to the end of another SAS data set	"APPEND Statement" on page 302
associates a format, informat, or label with variables in the SAS data set specified in the MODIFY statement	"ATTRIB Statement" on page 309
Initiate, control, suspend, resume, or terminate event logging to an audit file	"AUDIT Statement" on page 310
Rename one or more SAS files	"CHANGE Statement" on page 312
Describe the contents of one or more SAS data sets and prints a directory of the SAS library	"CONTENTS Statement" on page 314
Copy all or some of the SAS files	"COPY Statement" on page 318
Delete SAS files	"DELETE Statement" on page 328
Exchange the names of two SAS files	"EXCHANGE Statement" on page 332
Exclude SAS files from copying	"EXCLUDE Statement" on page 333
Permanently assign, change, and remove variable formats	"FORMAT Statement" on page 333
Create an integrity constraint	"IC CREATE Statement" on page 334
Delete an integrity constraint	"IC DELETE Statement" on page 337
Reactivate a foreign key integrity constraint	"IC REACTIVATE Statement" on page 337
Update centiles statistics for indexed variables	"INDEX CENTILES Statement" on page 338
Create simple or composite indexes	"INDEX CREATE Statement" on page 339
Delete one or more indexes	"INDEX DELETE Statement" on page 340
Permanently assign, change, and remove variable informats	"INFORMAT Statement" on page 341
Assign, change, and remove variable labels	"LABEL Statement" on page 341
Change the attributes of a SAS file and the attributes of variables	"MODIFY Statement" on page 342
Specifies whether to restore or delete the disabled indexes and integrity constraints	"REBUILD Statement" on page 347
Rename variables in the SAS data set	"RENAME Statement" on page 348
Attempt to restore damaged SAS data sets or catalogs	"REPAIR Statement" on page 349

Task	Statement
Delete all the SAS files except the ones listed in the SAVE statement	"SAVE Statement" on page 351
Select SAS files for copying	"SELECT Statement" on page 352

The following table lists the statements and the options for the DATASETS procedure. Several of the statements can be used only in a MODIFY run group.

Statement	Options
"PROC DATASETS Statement" on page 296	
"AGE Statement" on page 300	ALTAR
	MEMTYPE
"APPEND Statement" on page 302	APPENDVER
	DATA
	FORCE
	GETSORT
	NOWARN
"ATTRIB Statement" on page 309 (must be used in a MODIFY RUN group)	FORMAT
	INFORMAT
	LABEL
"AUDIT Statement" on page 310	GENNUM
	AUDIT_ALL
	LOG
	USER_VAR
	SUSPEND
	RESUME
	TERMINATE
"CHANGE Statement" on page 312	ALTER
	GENNUM
	MEMTYPE

Statement	Options
"CONTENTS Statement" on page 314	CENTILES
	DATA
	DETAILS
	DIRECTORY
	FMTLEN
	MEMTYPE
	NODS
	ORDER
	OUT
	OUT2
	SHORT
	VARNUM
"COPY Statement" on page 318	ALTER
	CLONE
	CONSTRAINT
	DATECOPY
	FORCE
	INDEX
	MEMTYPE
	MOVE
	NOWARN
"DELETE Statement" on page 328	ALTER
	GENNUM
	MEMTYPE
"EXCHANGE Statement" on page 332	ALTER
	MEMTYPE
"EXCLUDE Statement" on page 333	MEMTYPE
"FORMAT Statement" on page 333 (must be used in a MODIFY RUN group)	
"IC CREATE Statement" on page 334 (must be used in a MODIFY RUN group)	MESSAGE
"IC DELETE Statement" on page 337 (must be used in a MODIFY RUN group)	_ALL_
"IC REACTIVATE Statement" on page 337 (must be used in a MODIFY RUN group)	
"INDEX CENTILES Statement" on page 338 (must be used in a MODIFY RUN group)	REFRESH
	UPDATECENTILES
"INDEX CREATE Statement" on page 339 (must be used in a MODIFY RUN group)	NOMISS
	UNIQUE
	UPDATECENTILES
"INDEX DELETE Statement" on page 340 (must be used in a MODIFY RUN group)	_ALL_

Statement	Options
"INFORMAT Statement" on page 341 (must be used in a MODIFY RUN group)	
"LABEL Statement" on page 341 (must be used in a MODIFY RUN group)	
"MODIFY Statement" on page 342	ALTER
	CORRECTENCODING
	DTC
	GENMAX
	GENNUM
	LABEL
	MEMTYPE
	PW
	READ
	SORTEDBY
	TYPE
	WRITE
"REBUILD Statement" on page 347	ALTER
	GENNUM
	MEMTYPE
	NOINDEX
"RENAME Statement" on page 348 (must be used in a MODIFY RUN group)	
"REPAIR Statement" on page 349	ALTER
	GENNUM
	MEMTYPE
"SAVE Statement" on page 351	MEMTYPE
"SELECT Statement" on page 352	ALTER
	MEMTYPE

PROC DATASETS Statement

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

Task	Option
Provide alter access to any alter-protected SAS file in the SAS library	ALTER=
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
Specify the procedure input/output library	LIBRARY=
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 library.

See also: "Using Passwords with the DATASETS Procedure" on page 355

DETAILS | NODETAILS
determines whether the following columns are written to the log:

Obs, Entries, or Indexes
gives the number of observations for SAS files of type AUDIT, DATA, and VIEW; the number of entries for type CATALOG; and the number of files of type INDEX that are associated with a data file, if any. If SAS cannot determine the number of observations in a SAS data set, the value in this column is set to missing. For example, in a very large data set, if the number of observations or deleted observations exceeds the number that can be stored in a double-precision integer, the count shows as missing. The value for type CATALOG is the total number of entries. For other types, this column is blank.

Tip: The value for files of type INDEX includes both user-defined indexes and indexes created by integrity constraints. To view index ownership and attribute information, use PROC DATASETS with the CONTENTS statement and the OUT2 option.

Vars
gives the number of variables for types AUDIT, DATA, and VIEW. If SAS cannot determine the number of variables in the SAS data set, the value in this column is set to missing. For other types, this column is blank.

Label

contains the label associated with the SAS data set. This column prints a label only for the type DATA.

The DETAILS option affects output only when a directory is specified and requires read access to all read-protected SAS files in the SAS library. If you do not supply the read password, the directory listing contains missing values for the columns produced by the DETAILS option.

Default: If neither DETAILS or NODETAILS is specified, the default is the system option setting. The default system option setting is NODETAILS.

Tip: If you are using the SAS windowing environment and specify the DETAILS option for a library that contains read-protected SAS files, a dialog box prompts you for each read password that you do not specify in the PROC DATASETS statement. Therefore, you might want to assign the same read password to all SAS files in the same SAS library.

Featured in: Example 2 on page 374

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 353 for a discussion of RUN-group processing and error handling.
- forces all APPEND statements to concatenate two data sets even when the variables in the data sets are not exactly the same. The APPEND statement drops the extra variables and issues a warning message to the SAS log unless the NOWARN option is specified (either with the APPEND statement or PROC DATASETS). Refer to "APPEND Statement" on page 302 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 358

"Understanding Generation Data Sets" in *SAS Language Reference: Concepts*

KILL

deletes *all* SAS files in the SAS library that are available for processing. The MEMTYPE= option subsets the member types that the statement deletes. The following example deletes all the data files in the WORK library:

```
proc datasets lib=work kill memtype=data;
run;
quit;
```

CAUTION:
The KILL option deletes the SAS files immediately after you submit the statement. △

LIBRARY=*libref*
names the library that the procedure processes. This library is the *procedure input/output library*.

Aliases: DDNAME=, DD=, LIB=

Default: WORK or USER. See "Temporary and Permanent SAS Data Sets" on page 18 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 2 on page 374

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 356

NODETAILS
See the description of DETAILS on page 297.

NOLIST
suppresses the printing of the directory of the SAS files in the SAS log.

Featured in: Example 4 on page 382

Note: If you specify the ODS RTF destination, PROC DATASETS output goes to both the SAS log and the ODS output area. The NOLIST option suppresses output to both. To see the output only in the SAS log, use the ODS EXCLUDE statement by specifying the member directory as the exclusion. △

NOWARN
suppresses the error processing that occurs when a SAS file that is specified in a SAVE, CHANGE, EXCHANGE, REPAIR, DELETE, or COPY statement or listed as the first SAS file in an AGE statement, is not in the procedure input library. When an error occurs and the NOWARN option is in effect, PROC DATASETS continues processing that RUN group. If NOWARN is not in effect, PROC DATASETS stops processing that RUN group and issues a warning for all operations except DELETE, for which it does not stop processing.

PW= *password*
provides the password for any protected SAS files in the SAS library. PW= can act as an alias for READ=, WRITE=, or ALTER=.

See also: "Using Passwords with the DATASETS Procedure" on page 355

READ=*read-password*
provides the read-password for any read-protected SAS files in the SAS library.

See also: "Using Passwords with the DATASETS Procedure" on page 355

AGE Statement

Renames a group of related SAS files in a library.

Featured in: Example 7 on page 388

AGE *current-name related-SAS-file-1 <...current-name related-SAS-file-n>*
</ <ALTER=*alter-password*>
<MEMTYPE=*mtype*>>;

Required Arguments

current-name
is a SAS file that the procedure renames. *current-name* receives the name of the first name in *related-SAS-file-1 <...related-SAS-file-n>*.

related-SAS-file-1 <...related-SAS-file-n>
is one or more SAS files in the SAS library.

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 355

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 356

Details

- The AGE statement renames the *current-name* to the name of the first name in the *related-SAS-files*, renames the first name in the *related-SAS-files* to the second name in the *related-SAS-files*, and so on, until it changes the name of the

next-to-last SAS file in the *related-SAS-files* to the last name in the *related-SAS-files*. The AGE statement then deletes the last file in the *related-SAS-files*..

- If the first SAS file named in the AGE statement does not exist in the SAS library, PROC DATASETS stops processing the RUN group containing the AGE statement and issues an error message. The AGE statement does not age any of the *related-SAS-files*. To override this behavior, use the NOWARN option in the PROC DATASETS statement.

  If one of the *related-SAS-files* does not exist, the procedure prints a warning message to the SAS log but continues to age the SAS files that it can.
- If you age a data set that has an index, the index continues to correspond to the data set.
- You can age only entire generation groups. For example, if data sets A and B have generation groups, then the following statement deletes generation group B and ages (renames) generation group A to the name B:

```
age a b;
```

For example, suppose the generation group for data set A has three historical versions and the generation group for data set B has tw historical versions. Then aging A to B has this effect:

Old Name	Version	New Name	Version
A	base	B	base
A	1	B	1
A	2	B	2
A	3	B	3
B	base	is deleted	
B	1	is deleted	
B	2	is deleted	

APPEND Statement

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

Default: If the BASE= data set is accessed through a SAS server and if no other user has the data set open at the time the APPEND statement begins processing, the BASE= data set defaults to CNTLLEV=MEMBER (member-level locking). When this behavior happens, no other user can update the file while the data set is processed.

Requirement: The BASE= data set must be a member of a SAS library that supports update processing.

Tip: You can specify most data set options for the BASE= argument and DATA= option. However, if you specify DROP=, KEEP=, or RENAME= data set option for the BASE= data set, the option is ignored. You can use any global statements as well. See "Global Statements" on page 20.

Tip: If a failure occurs during processing, the data set is marked as damaged and is reset to its preappend condition at the next REPAIR statement. If the data set has an index, the index is not updated with each observation but is updated once at the end. (This behavior is Version 7 and later, as long as APPENDVER=V6 is not set.)

Featured in: Example 6 on page 386

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

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.

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

Alias: OUT=

Featured in: Example 6 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 306

DATA=<*libref.*> *SAS-data-set*
names the SAS data set containing observations that you want to append to the end of the SAS data set specified in the BASE= argument.

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

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

Alias: NEW=

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

See also: "Appending with Generation Groups" on page 308

Featured in: Example 6 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 307

"Appending to Data Sets That Contain Variables with Different Attributes" on page 307

Featured in: Example 6 on page 386

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;
```

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

- The BASE= data set must:

- be SAS Version 7 or higher
- contain no observations
- accept sort indicators

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

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

Restrictions: The GETSORT option has no effect on the data sets if one of the following criteria applies:

- if the BASE= data set has an audit trail associated with it

  Note: This restriction causes a WARNING in the output while the APPEND process continues. △

- if there are dropped, kept, or renamed variables in the DATA= data file

Featured in: Example 9 on page 392

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

Appending Sorted Data Sets

You can append sorted data sets and maintain the sort using the following guidelines:

- The DATA= data set and the BASE= data set contain sort indicators from the SORT procedure.
- The DATA= data set and the BASE= data set are sorted using the same variables.
- The observations added from the DATA= data set do not violate the sort order of the BASE= data set.

The sort indicator from the BASE= data set is retained.

Using the Block I/O Method to Append

The block I/O method is used to append blocks of data instead of one observation at a time. This method increases performance when you are appending large data sets. SAS determines whether to use the block I/O method. Not all data sets can use the block I/O method. There are restrictions set by the APPEND statement and the Base SAS engine.

To display information in the SAS log about the append method that is being used, you can specify the MSGLEVEL= system option as follows:

```
options msglevel=i;
```

The following message is written to the SAS log, if the block I/O method is *not* used:

```
INFO: Data set block I/O cannot be used because:
```

If the APPEND statement determines that the block I/O will *not* be used, one of the following explanations is written to the SAS log:

```
INFO: - The data sets use different engines, have different variables
or have attributes that might differ.
INFO: - There is a WHERE clause present.
INFO: - There is no member level locking.
INFO: - The OBS option is active.
```

```
INFO: - The FIRSTOBS option is active.
```

If the Base SAS engine determines that the block I/O method will *not* be used, one of the following explanations is written to the SAS log:

```
INFO: - Referential Integrity Constraints exist.
INFO: - Cross Environment Data Access is being used.
INFO: - The file is compressed.
INFO: - The file has an audit file which is not suspended.
```

Restricting the Observations That Are Appended

You can use the WHERE= data set option with the DATA= data set in order to restrict the observations that are appended. Likewise, you can use the WHERE statement in order to restrict the observations from the DATA= data set. The WHERE statement has no effect on the BASE= data set. If you use the WHERE= data set option with the BASE= data set, WHERE= has no effect.

CAUTION:

For an existing BASE= data set: If there is a WHERE statement in the BASE= data set, it takes effect only if the WHEREUP= option is set to YES. △

CAUTION:

For the non-existent BASE= data set: If there is a WHERE statement in the non-existent BASE= data set, regardless of the WHEREUP option setting, you use the WHERE statement. △

Note: You cannot append a data set to itself by using the WHERE= data set option. △

Choosing between the SET Statement and the APPEND Statement

If you use the SET statement in a DATA step to concatenate two data sets, SAS must process all the observations in both data sets to create a new one. The APPEND statement bypasses the processing of data in the original data set and adds new observations directly to the end of the original data set. Using the APPEND statement can be more efficient than using a SET statement if any of the following list occurs:

- the BASE= data set is large.
- all variables in the BASE= data set have the same length and type as the variables in the DATA= data set and if all variables exist in both data sets.

  Note: You can use the CONTENTS statement to see the variable lengths and types. △

The APPEND statement is especially useful if you frequently add observations to a SAS data set (for example, in production programs that are constantly appending data to a journal-type data set).

Appending Password-Protected SAS Data Sets

In order to use the APPEND statement, you need read access to the DATA= data set and write access to the BASE= data set. To gain access, use the READ= and WRITE= data set options in the APPEND statement the way you would use them in any other SAS statement, which is in parentheses immediately after the data set name. When you are appending password-protected data sets, use the following guidelines:

- If you do not give the read password for the DATA= data set in the APPEND statement, by default the procedure looks for the read password for the DATA= data set in the PROC DATASETS statement. However, the procedure does not look for the write password for the BASE= data set in the PROC DATASETS

statement. Therefore, you must specify the write password for the BASE= data set in the APPEND statement.

- If the BASE= data set is read-protected only, you must specify its read password in the APPEND statement.

Appending to a Compressed Data Set

You can concatenate compressed SAS data sets. Either or both of the BASE= and DATA= data sets can be compressed. If the BASE= data set allows the reuse of space from deleted observations, the APPEND statement might insert the observations into the middle of the BASE= data set to make use of available space.

For information 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 sequentially. The behavior reduces most of the disk I/O and results in a faster append method.

The fast-append method is used by default when the following requirements are met; otherwise, the Version 6 method is used:

- The BASE= data set is open for member-level locking. If CNTLLEV= is set to record, then the fast-append method is not used.
- The BASE= data set does not contain referential integrity constraints.
- The BASE= data set is not accessed using the Cross Environment Data Access (CEDA) facility.
- The BASE= data set is not using a WHERE= data set option.

To display information in the SAS log about the append method that is being used, you can specify the MSGLEVEL= system option as follows:

```
options msglevel=i;
```

Either a message 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. After observations are appended, some of them might subsequently be deleted.

For a simple example, consider that the BASE= data set has ten observations numbered from 1 to 10 with a UNIQUE index for the variable ID. You append a data set that contains five observations numbered from 1 to 5, and observations 3 and 4 both contain the same value for ID. The following actions occur:

1. After the observations are appended, the BASE= data set contains 15 observations numbered from 1 to 15.
2. SAS updates the index for ID, validates the values, and determines that observations 13 and 14 contain the same value for ID.

3 SAS deletes one of the observations from the BASE= data set, resulting in 14 observations that are numbered from 1 to 15. For example, observation 13 is deleted. Note that you cannot predict which observation is deleted, because the internal sort might place either observation first. (In Version 6, you could predict that observation 13 would be added and observation 14 would be rejected.)

If you do not want the current behavior (which could result in deleted observations) or if you want to be able to predict which observations are appended, request the Version 6 append method by specifying the APPENDVER=V6 option:

```
proc datasets;
   append base=a data=b appendver=v6;
run;
```

Note: In Version 6, deleting the index and then recreating it after the append could improve performance. The current method might eliminate the need to do that. However, the performance depends on the nature of your data. △

Appending to Data Sets with Different Variables

If the DATA= data set contains variables that are not in the BASE= data set, use the FORCE option in the APPEND statement to force the concatenation of the two data sets. The APPEND statement drops the extra variables and issues a warning message. You can use the NOWARN option to suppress the warning message.

If the BASE= data set contains a variable that is not in the DATA= data set, the APPEND statement concatenates the data sets, but the observations from the DATA= data set have a missing value for the variable that was not present in the DATA= data set. The FORCE option is not necessary in this case.

If you use the DROP=, KEEP=, or RENAME= options on the BASE= data set, the options ONLY affect the APPEND processing and does *not* change the variables in the appended BASE= data set. Variables that are dropped or not kept using the DROP= and KEEP= options still exist in the appended BASE= data set. Variables that are renamed using the RENAME= option remain with their original name in the appended BASE= data set.

Appending to Data Sets That Contain Variables with Different Attributes

- If a variable has different attributes in the BASE= data set than it does in the DATA= data set, the attributes in the BASE= data set prevail.
- If the SAS formats in the DATA= data set are different from those in the BASE= data set, then the SAS formats in the BASE= data set are used. However, SAS does not convert the data from the DATA= data set in order to be consistent with the SAS formats in the BASE= data set. The result could be data that seems to be incorrect. A warning message is displayed in the SAS log. The following example illustrates appending data by using different SAS formats:

```
data format1;
   input Date date9.;
   format Date date9.;
datalines;
24sep1975
22may1952
;

data format2;
```

```
   input Date datetime20.;
   format Date datetime20.;
datalines;
25aug1952:11:23:07.4
;

proc append base=format1 data=format2;
run;
```

The following messages are displayed in the SAS log.

Output 17.2 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 the following consequences:
  - The length of the variables in the BASE= data set takes precedence. SAS truncates values from the DATA= data set to fit them into the length that is specified in the BASE= data set.
  - The type of the variables in the BASE= data set takes precedence. The APPEND statement replaces values of the wrong type (all values for the variable in the DATA= data set) with missing values.

Note: If a character variable's transcoding attribute is opposite in the BASE= and DATA= data sets (for example, one is YES and the other is NO), then a warning is issued. To determine the transcoding attributes, use the CONTENTS procedure for each data set. You set the transcoding attribute with the TRANSCODE= option in the ATTRIB statement or with the TRANSCODE= column modifier in PROC SQL. △

Appending Data Sets That Contain Integrity Constraints

If the DATA= data set contains integrity constraints and the BASE= data set does not exist, the APPEND statement copies the general constraints. Note that the referential constraints are not copied. If the BASE= data set exists, the APPEND action copies only observations.

Appending with Generation Groups

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

SAS Statements	Result

`proc datasets;`
`  append base=a`
`  data=b(gennum=2);`
	appends historical version B#002 to base A

`proc datasets;`
`  append base=a(gennum=2)`
`  data=b(gennum=2);`
	appends historical version B#002 to historical version A#002

Using the APPEND Procedure Instead of the APPEND Statement

The only difference between the APPEND procedure and the APPEND statement in PROC DATASETS, is the default for *libref* in the BASE= and DATA= arguments. For PROC APPEND, the default is either WORK or USER. For the APPEND statement, the default is the libref of the procedure input library.

System Failures

If a system failure or some other type of interruption occurs while the procedure is executing, the append operation might not be successful; it is possible that not all, perhaps none, of the observations are added to the BASE= data set. In addition, the BASE= data set might suffer damage. The APPEND operation performs an update in place, which means that it does not make a copy of the original data set before it begins to append observations. If you want to be able to restore the original observations, you can initiate an audit trail for the base data file and select to store a before-update image of the observations. Then you can write a DATA step to extract and reapply the original observations to the data file. For information about initiating an audit trail, see the PROC DATASETS AUDIT Statement.

ATTRIB Statement

Associates a format, informat, or label with variables in the SAS data set specified in the MODIFY statement.

Restriction: Must appear in a MODIFY RUN group

Featured in: Example 1 on page 372

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

Required Arguments

variable list
names the variables that you want to associate with the attributes. You can list the variables in any form that SAS allows.

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

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

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

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

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

LABEL='*label*'
associates a label with variables in the *variable-list.*

Details

Within the DATASETS procedure, the ATTRIB statement must be used in a MODIFY RUN group and can use only the FORMAT, INFORMAT, and LABEL options. The ATTRIB statement is the simplest way to remove or change all variable labels, formats, or informats in a data set using the keyword _ALL_. For an example, see Example 1 on page 372.

Note: For more information about shortcut methods for specifying variables, see "Shortcuts for Specifying Lists of Variable Names" on page 25. △

If you are not deleting or changing all attributes, it is easier to use the following statements, "LABEL Statement" on page 341, "FORMAT Statement" on page 333, and "INFORMAT Statement" on page 341.

AUDIT Statement

Initiates and controls event logging to an audit file as well as suspends, resumes, or terminates event logging in an audit file.

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.

See also: "Understanding an Audit Trail" in *SAS Language Reference: Concepts*

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
specifies whether the administrative events are logged to the audit file (that is, the SUSPEND and RESUME actions).

BEFORE_IMAGE=YES|NO
specifies whether the before-update record images are logged to the audit file.

DATA_IMAGE=YES|NO
specifies whether the added, deleted, and after-update record images are logged to the audit file.

ERROR_IMAGE=YES|NO
specifies whether the after-update record images are logged to the audit file.

Default: All images are logged by default; that is, all four are set to YES.

Tip: If you do not want to log a particular image, specify NO for the image type. For example, the following code turns off logging the error images, but the administrative, before, and data images continue to be logged:

```
log error_image=no;
```

USER_VAR *variable-1 < ... variable-n>*
defines optional variables to be logged in the audit file with each update to an observation. The following syntax defines variables:

USER_VAR *variable-name-1* <$> <*length*> <LABEL=’*variable-label*’ >
<… *variable-name-n* <$> <*length*> <LABEL=’*variable-label*’ > >

where

variable-name
is a name for the variable.

$
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 library.

Featured in: Example 2 on page 374

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 2 on page 374

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 355

GENNUM=ALL|*integer*
restricts processing for generation data sets. You can use the option either in parentheses after the name of each SAS file or after a forward slash. The following list shows valid values:

ALL | 0
refers to the base version and all historical versions of a generation group.

integer
refers to a specific version from a generation group. Specifying a positive number is an absolute reference to a specific generation number that is appended to a data set's name; that is, **gennum=2** specifies MYDATA#002. Specifying a negative number is a relative reference to a historical version in relation to the base version, from the youngest to the oldest; that is, **gennum=-1** refers to the youngest historical version.
For example, the following statements change the name of version A#003 to base B:

```
proc datasets;
   change A=B / gennum=3;

proc datasets;
   change A(gennum=3)=B;
```

The following CHANGE statement produces an error:

```
proc datasets;
   change A(gennum=3)=B(gennum=3);
```

See also:

"Restricting Processing for Generation Data Sets" on page 358

"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 356

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 library, PROC DATASETS stops processing the RUN group containing the CHANGE statement and issues an error message. To override this behavior, use the NOWARN option in the PROC DATASETS statement.
- If you change the name of a data set that has an index, the index continues to correspond to the data set.

CONTENTS Statement

Describes the contents of one or more SAS data sets and prints the directory of the SAS library.

Restriction: You cannot use the WHERE option to affect the output because PROC CONTENTS does not process any observations.

Tip: You can use data set options with the DATA=, OUT=, and OUT2= options. You can use any global statements as well. See "Global Statements" on page 20.

Featured in: Example 5 on page 384

CONTENTS <*option-1* <*...option-n*>>;

Task	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
Print a list of the SAS files in the SAS 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

Task	Option
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 and numeric order beginning with uppercase and then lowercase names	ORDER=COLLATE
Print a list of variables in alphabetical and numeric order even if they include mixed-case names	ORDER=CASECOLLATE
Print a list of variables in alphabetical order, ignoring the case of the letters	ORDER=IGNORECASE
Print a list of variables in the order of their logical position in the data set	ORDER=VARNUM
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	percentage of the data values that must be changed before the CENTILES for the indexed variables are automatically updated.
Current Update Percentage	percentage of index updated since CENTILES were refreshed.
# of Unique Values	number of unique indexed values.
Variables	names of the variables used to make up the index. Centile information is listed below the variables.

DATA=*SAS-file-specification*
specifies an entire library or a specific SAS data set within a library. *SAS-file-specification* can take one of the following forms:

<libref.>SAS-data-set
names one SAS data set to process. The default for *libref* is the libref of the procedure input library. For example, to obtain the contents of the SAS data set HTWT from the procedure input library, use the following CONTENTS statement:

```
contents data=HtWt;
```

To obtain the contents of a specific version from a generation group, use the GENNUM= data set option as shown in the following CONTENTS statement:

```
contents data=HtWt(gennum=3);
```

*<libref.>*_ALL_
gives you information about all SAS data sets that have the type or types specified by the MEMTYPE= option. *libref* refers to the SAS library. The default for *libref* is the libref of the procedure input library.

- If you are using the _ALL_ keyword, you need read access to all read-protected SAS data sets in the SAS library.
- DATA=_ALL_ automatically prints a listing of the SAS files that are contained in the SAS library. Note that for SAS views, all librefs that are associated with the views must be assigned in the current session in order for them to be processed for the listing.

Default: most recently created data set in your job or session, from any SAS library.

Tip: If you specify a read-protected data set in the DATA= option but do not give the read password, by default the procedure looks in the PROC DATASETS statement for the read password. However, if you do not specify the DATA= option and the default data set (last one created in the session) is read protected, the procedure does not look in the PROC DATASETS statement for the read password.

Featured in: Example 5 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 nor NODETAILS is specified, the defaults are as follows: for the CONTENTS procedure, the default is the system option setting, which is NODETAILS; for the CONTENTS statement, the default is whatever is specified in the PROC DATASETS statement, which also defaults to the system option setting.

See also: description of the additional columns in "Options" in "PROC DATASETS Statement" on page 296

DIRECTORY
prints a list of all SAS files in the specified SAS library. If DETAILS is also specified, using DIRECTORY causes the additional columns described in DETAILS|NODETAILS on page 297 to be printed.

FMTLEN
prints the length of the informat or format. If you do not specify a length for the informat or format when you associate it with a variable, the length does not appear in the output of the CONTENTS statement unless you use the FMTLEN option. The length also appears in the FORMATL or INFORML variable in the output data set.

MEMTYPE=(*mtype-1 <...mtype-n>*)
restricts processing to one or more member types. The CONTENTS statement produces output only for member types DATA, VIEW, and ALL, which includes DATA and VIEW.

MEMTYPE= in the CONTENTS statement differs from MEMTYPE= in most of the other statements in the DATASETS procedure in the following ways:

- A slash does not precede the option.
- You cannot enclose the MEMTYPE= option in parentheses to limit its effect to only the SAS file immediately preceding it.

MEMTYPE= results in a directory of the library in which the DATA= member is located. However, MEMTYPE= does not limit the types of members whose contents

are displayed unless the _ALL_ keyword is used in the DATA= option. For example, the following statements produce the contents of only the SAS data sets with the member type DATA:

```
proc datasets memtype=data;
   contents data=_all_;
run;
```

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 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 316.

NOPRINT
suppresses printing the output of the CONTENTS statement.

ORDER= COLLATE | CASECOLLATE | IGNORECASE | VARNUM

COLLATE	prints a list of variables in alphabetical order beginning with uppercase and then lowercase names.
CASECOLLATE	prints a list of variables in alphabetical order even if they include mixed-case names and numerics.
IGNORECASE	prints a list of variables in alphabetical order ignoring the case of the letters.
VARNUM	is the same as the VARNUM option. See VARNUM on page 318.

Note: The ORDER= option does not affect the order of the OUT= and OUT2= data sets. △
See Example 10 on page 394 to compare the default and the four options for ORDER=.

OUT=*SAS-data-set*
names an output SAS data set.

Tip: OUT= does not suppress the printed output from the statement. If you want to suppress the printed output, you must use the NOPRINT option.

See: "The OUT= Data Set" on page 366 for a description of the variables in the OUT= data set.

See also: Example 8 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 371 for a description of the variables in the OUT2= data set.

SHORT

prints only the list of variable names, the index information, and the sort information for the SAS data set.

Restriction: If the list of variables is more than 32,767 characters, the list is truncated and a WARNING is written to the SAS log. To get a complete list of the variables, request an alphabetical listing of the variables.

VARNUM
prints a list of the variable names in the order of their logical position in the data set. By default, the CONTENTS statement lists the variables alphabetically. The physical position of the variable in the data set is engine-dependent.

Details

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 362.

Note: If a label is changed after a view is created from a data set with variable labels, the CONTENTS or DATASETS procedure output shows the original labels. The view must be recompiled in order for the CONTENTS or DATASETS procedure output to reflect the new variable labels. △

Using the CONTENTS Procedure instead of the CONTENTS Statement

The only difference between the CONTENTS procedure and the CONTENTS statement in PROC DATASETS is the default for *libref* in the DATA= option. For PROC CONTENTS, the default is WORK. For the CONTENTS statement, the default is the libref of the procedure input library.

COPY Statement

Copies all or some of the SAS files in a SAS library.

Restriction: The COPY statement does not support data set options.

Tip: The COPY statement defaults to the encoding and data representation of the output library when you use Remote Library Services (RLS) such as SAS/SHARE or SAS/CONNECT. If you are not using RLS, you must use the PROC COPY option NOCLONE for the output files to take on the encoding and data representation of the output library. Using the NOCLONE option results in a copy with the data representation of the data library (if specified in the OUTREP= LIBNAME option) or the native data representation of the operating environment.

Featured in: Example 2 on page 374

COPY OUT=*libref-1*
<CLONE|NOCLONE>
<CONSTRAINT=YES|NO>
<DATECOPY>
<FORCE>
<IN=*libref-2*>
<INDEX=YES|NO>

<MEMTYPE=(*mtype-1* <...*mtype-n*>)>
<MOVE <ALTER=*alter-password*>> ;

Required Arguments

OUT=*libref-1*
names the SAS library to copy SAS files to.

Aliases: OUTLIB= and OUTDD=

Featured in: Example 2 on page 374

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.

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 355

CLONE|NOCLONE
specifies whether to copy the following data set attributes:

- size of input/output buffers
- whether the data set is compressed
- whether free space is reused
- data representation of input data set, library, or operating environment
- encoding value
- whether a compressed data set can be randomly accessed by an observation number

These attributes are specified with data set options, SAS system options, and LIBNAME statement options:

- BUFSIZE= value for the size of the input/output buffers
- COMPRESS= value for whether the data set is compressed
- REUSE= value for whether free space is reused
- OUTREP= value for data representation
- ENCODING= or INENCODING= for encoding value
- POINTOBS= value for whether a compressed data set can be randomly accessed by an observation number

For the BUFSIZE= attribute, the following table summarizes how the COPY statement works:

Table 17.1 CLONE and the Buffer Page Size Attribute

Option	Copy Statement
CLONE	uses the BUFSIZE= value from the input data set for the output data set.
NOCLONE	uses the current setting of the SAS system option BUFSIZE= for the output data set.
neither	determines the type of access method, sequential or random, used by the engine for the input data set and the engine for the output data set. If both engines use the same type of access, the COPY statement uses the BUFSIZE= value from the input data set for the output data set. If the engines do not use the same type of access, the COPY statement uses the setting of SAS system option BUFSIZE= for the output data set.

For the COMPRESS= attribute, the following table summarizes how the COPY statement works:

Table 17.2 CLONE and the Compression Attribute

Option	Copy Statement
CLONE	uses the values from the input data set for the output data set.
NOCLONE	results in a copy with the compression of the operating environment or, if specified, the value of the COMPRESS= option in the LIBNAME statement for the library.
neither	defaults to CLONE.

For the REUSE= attribute, the following table summarizes how the COPY statement works:

Table 17.3 CLONE and the Reuse Space Attribute

Option	Copy Statement
CLONE	uses the values from the input data set for the output data set. If the engine for the input data set does not support the reuse space attribute, then the COPY statement uses the current setting of the corresponding SAS system option.
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 17.4 CLONE and the Data Representation Attribute

Option	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 OUT= library.
neither	defaults to CLONE.

Data representation is the form in which data is stored in a particular operating environment. Different operating environments use the following different standards or conventions:

- for storing floating-point numbers (for example, IEEE or IBM 390)
- for character encoding (ASCII or EBCDIC)
- for the ordering of bytes in memory (big Endian or little Endian)
- for word alignment (4-byte boundaries or 8-byte boundaries)
- for data-type length (16-bit, 32-bit, or 64-bit)

Native data representation is when the data representation of a file is the same as the CPU operating environment. For example, a file in Windows data representation is native to the Windows operating environment.

For the ENCODING= attribute, the following table summarizes how the COPY statement works.

Table 17.5 CLONE and the Encoding Attribute

Option	COPY Statement
CLONE	results in a copy that uses the encoding of the input data set or, if specified, the value of the INENCODING= option in the LIBNAME statement for the input library.
NOCLONE	results in a copy that uses the encoding of the current session encoding or, if specified, the value of the OUTENCODING= option in the LIBNAME statement for the output library.
neither	defaults to CLONE.

All data that is stored, transmitted, or processed by a computer is in an encoding. An encoding maps each character to a unique numeric representation. An encoding is a combination of a character set with an encoding method. A character set is the repertoire of characters and symbols that are used by a language or group of languages. An encoding method is the set of rules that are used to assign the numbers to the set of characters that are used in an encoding.

For the POINTOBS= attribute, the following table summarizes how the COPY statement works. To use POINTOBS=, the output data set must be compressed.

Table 17.6 CLONE and the POINTOBS= Attribute

Option	Copy Statement
CLONE	uses the POINTOBS= value from the input data set for the output data set.
NOCLONE	uses the LIBNAME statement if the output data set is compressed and the POINTOBS= option is specified and supported by the output engine. If the LIBNAME statement is not specified and the data set is compressed, the default is POINTOBS=YES when supported by the output engine.
neither	defaults to CLONE.

CONSTRAINT=YES|NO
specifies whether to copy all integrity constraints when copying a data set.

Default: NO

Tip: For data sets with integrity constraints that have a foreign key, the COPY statement copies the general and referential constraints if CONSTRAINT=YES is specified and the entire library is copied. If you use the SELECT or EXCLUDE statement to copy the data sets, then the referential integrity constraints are not copied. For more information, see "Understanding Integrity Constraints" in *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 in the MODIFY statement. See "MODIFY Statement" on page 342.

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 the last modified date changes to the current date. Other attributes that require additional processing and that could affect the last modified date include integrity constraints and a sort indicator.

FORCE
allows you to use the MOVE option for a SAS data set on which an audit trail exists.

Note: The AUDIT file is not moved with the audited data set. △

INDEX=YES|NO
specifies whether to copy all indexes for a data set when copying the data set to another SAS library.

Default: YES

MEMTYPE=(*mtype-1 <...mtype-n>*)
restricts processing to one or more member types.

Aliases: MT=, MTYPE=

Default: If you omit MEMTYPE= in the PROC DATASETS statement, the default is MEMTYPE=ALL.

Note: When PROC COPY processes a SAS library on tape and the MEMTYPE= option is not specified, it scans the entire sequential library for entries until it reaches the end-of-file. If the sequential library is a multivolume tape, all tape volumes are mounted. This behavior is also true for single-volume tape libraries. △

See also:

"Specifying Member Types When Copying or Moving SAS Files" on page 324

"Member Types" on page 357

Featured in: Example 2 on page 374

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 2 on page 374

NOCLONE

See the description of CLONE.

Using the Block I/O Method to Copy

The block I/O method is used to copy blocks of data instead of one observation at a time. This method can increase performance when you are copying large data sets. SAS determines whether to use this method. Not all data sets can use the block I/O method. There are restrictions set by the COPY statement and the Base SAS engine.

To display information in the SAS log about the copy method that is being used, you can specify the MSGLEVEL= system option as follows:

```
options msglevel=i;
```

The following message is written to the SAS log, if the block I/O method is *not* used:

```
INFO: Data set block I/O cannot be used because:
```

If the COPY statement determines that the block I/O will *not*be used, one of the following explanations is written to the SAS log:

```
INFO: - The data sets use different engines, have different variables
or have attributes that might differ.
INFO: - There is no member level locking.
INFO: - The OBS option is active.
INFO: - The FIRSTOBS option is active.
```

If the Base SAS engine determines that the block I/O method will *not* be used, one of the following explanations is written to the SAS log:

```
INFO: - Referential Integrity Constraints exist.
INFO: - Cross Environment Data Access is being used.
INFO: - The file is compressed.
INFO: - The file has an audit file which is not suspended.
```

If you are having performance issues and want to create a subset of a large data set for testing, you can use the OBS=0 option. In this case, you want to reduce the use of system resources by disabling the block I/O method.

The following example uses the OBS=0 option to reduce the use of system resources:

```
options obs=0 msglevel=i;
proc copy in=old out=lib;
select a;
```

```
run;
```

You get the same results when you use the SET statement:

```
data lib.new;
    if 0 then set old.a;
    stop;
run;
```

Copying an Entire Library

To copy an entire SAS library, simply specify an input data library and an output data library following the COPY statement. For example, the following statements copy all the SAS files in the SOURCE data library into the DEST data library:

```
proc datasets library=source;
   copy out=dest;
run;
```

Copying Selected SAS Files

To copy selected SAS files, use a SELECT or EXCLUDE statement. For more discussion of using the COPY statement with a SELECT or an EXCLUDE statement, see “Specifying Member Types When Copying or Moving SAS Files” on page 324 and see Example 2 on page 374 for an example. Also, see “EXCLUDE Statement” on page 333 and “SELECT Statement” on page 352.

You can also select or exclude an abbreviated list of members. For example, the following statement selects members TABS, TEST1, TEST2, and TEST3:

```
select tabs test1-test3;
```

Also, you can select a group of members whose names begin with the same letter or letters by entering the common letters followed by a colon (:). For example, you can select the four members in the previous example and all other members having names that begin with the letter T by specifying the following statement:

```
select t:;
```

You specify members to exclude in the same way that you specify those to select. That is, you can list individual member names, use an abbreviated list, or specify a common letter or letters followed by a colon (:). For example, the following statement excludes the members STATS, TEAMS1, TEAMS2, TEAMS3, TEAMS4 and all the members that begin with the letters RBI from the copy operation:

```
exclude stats teams1-teams4 rbi:;
```

Note that the MEMTYPE= option affects which types of members are available to be selected or excluded.

When a SELECT or EXCLUDE statement is used with CONSTRAINT=YES, only the general integrity constraints on the data sets are copied. Any referential integrity constraints are not copied. For more information, see “Understanding Integrity Constraints” in *SAS Language Reference: Concepts*.

Specifying Member Types When Copying or Moving SAS Files

The MEMTYPE= option in the COPY statement differs from the MEMTYPE= option in other statements in the procedure in several ways:

- A slash does not precede the option.
- You cannot limit its effect to the member immediately preceding it by enclosing the MEMTYPE= option in parentheses.

- The SELECT and EXCLUDE statements and the IN= option (in the COPY statement) affect the behavior of the MEMTYPE= option in the COPY statement according to the following rules:

  1. MEMTYPE= in a SELECT or EXCLUDE statement takes precedence over the MEMTYPE= option in the COPY statement. The following statements copy only VISION.CATALOG and NUTR.DATA from the default data library to the DEST data library; the MEMTYPE= value in the first SELECT statement overrides the MEMTYPE= value in the COPY statement.

     ```
     proc datasets;
        copy out=dest memtype=data;
           select vision(memtype=catalog) nutr;
     run;
     ```

  2. If you do not use the IN= option, or you use it to specify the library that happens to be the procedure input library, the value of the MEMTYPE= option in the PROC DATASETS statement limits the types of SAS files that are available for processing. The procedure uses the order of precedence described in rule 1 to further subset the types available for copying. The following statements do not copy any members from the default data library to the DEST data library; instead, the procedure issues an error message because the MEMTYPE= value specified in the SELECT statement is not one of the values of the MEMTYPE= option in the PROC DATASETS statement.

     ```
           /* This step fails! */
     proc datasets memtype=(data program);
        copy out=dest;
           select apples / memtype=catalog;
     run;
     ```

  3. If you specify an input data library in the IN= option other than the procedure input library, the MEMTYPE= option in the PROC DATASETS statement has no effect on the copy operation. Because no subsetting has yet occurred, the procedure uses the order of precedence described in rule 1 to subset the types available for copying. The following statements successfully copy BODYFAT.DATA to the DEST data library because the SOURCE library specified in the IN= option in the COPY statement is not affected by the MEMTYPE= option in the PROC DATASETS statement.

     ```
     proc datasets library=work memtype=catalog;
        copy in=source out=dest;
           select bodyfat / memtype=data;
     run;
     ```

Copying Views

The COPY statement with NOCLONE specified supports the OUTREP= and ENCODING= LIBNAME options for SQL views, DATA step views, and some SAS/ACCESS views (Oracle and Sybase). When you use the COPY statement with Remote Library Services (RLS) such as SAS/SHARE or SAS/CONNECT, the COPY statement defaults to the encoding and data representation of the output library.

CAUTION:

If you use the DATA statement's SOURCE=NOSAVE option when creating a DATA step view, the view cannot be copied from one version of SAS to another version. △

Copying Password-Protected SAS Files

You can copy a password-protected SAS file without specifying the password. In addition, because the password continues to correspond to the SAS file, you must know the password in order to access and manipulate the SAS file after you copy it.

Copying Data Sets with Long Variable Names

If the VALIDVARNAME=V6 system option is set and the data set has long variable names, the long variable names are truncated, unique variables names are generated, and the copy succeeds. The same is true for index names. If VALIDVARNAME=ANY, the copy fails with an error if the OUT= engine does not support long variable names.

When a variable name is truncated, the variable name is shortened to eight bytes. If this name has already been defined in the data set, the name is shortened and a digit is added, starting with the number 2. The process of truncation and adding a digit continues until the variable name is unique. For example, a variable named LONGVARNAME becomes LONGVARN, provided that a variable with that name does not already exist in the data set. In that case, the variable name becomes LONGVAR2.

CAUTION:

Truncated variable names can collide with names already defined in the input data set. This behavior is possible when the variable name that is already defined is exactly eight bytes long and ends in a digit. In the following example, the truncated name is defined in the output data set and the name from the input data set is changed:

```
options validvarname=any;
data test;
   longvar10='aLongVariableName';
   retain longvar1-longvar5 0;
run;

options validvarname=v6;
proc copy in=work out=sasuser;
   select test;
run;
```

In this example, LONGVAR10 is truncated to LONGVAR1 and placed in the output data set. Next, the original LONGVAR1 is copied. Its name is no longer unique and so it is renamed LONGVAR2. The other variables in the input data set are also renamed according to the renaming algorithm. The following example is from the SAS log:

```
1    options validvarname=any;
2    data test;
3       longvar10='aLongVariableName';
4       retain longvar1-longvar5 0;
5    run;

NOTE: The data set WORK.TEST has 1 observations and 6 variables.
NOTE: DATA statement used (Total process time):
      real time           2.60 seconds
      cpu time            0.07 seconds


6
7    options validvarname=v6;
8    proc copy in=work out=sasuser;
9       select test;
10   run;

NOTE: Copying WORK.TEST to SASUSER.TEST (memtype=DATA).
NOTE: The variable name longvar10 has been truncated to longvar1.
NOTE: The variable longvar1 now has a label set to longvar10.
NOTE: Variable LONGVAR1 already exists on file SASUSER.TEST, using LONGVAR2 instead.
NOTE: The variable LONGVAR2 now has a label set to LONGVAR1.
NOTE: Variable LONGVAR2 already exists on file SASUSER.TEST, using LONGVAR3 instead.
NOTE: The variable LONGVAR3 now has a label set to LONGVAR2.
NOTE: Variable LONGVAR3 already exists on file SASUSER.TEST, using LONGVAR4 instead.
NOTE: The variable LONGVAR4 now has a label set to LONGVAR3.
NOTE: Variable LONGVAR4 already exists on file SASUSER.TEST, using LONGVAR5 instead.
NOTE: The variable LONGVAR5 now has a label set to LONGVAR4.
NOTE: Variable LONGVAR5 already exists on file SASUSER.TEST, using LONGVAR6 instead.
NOTE: The variable LONGVAR6 now has a label set to LONGVAR5.
NOTE: There were 1 observations read from the data set WORK.TEST.
NOTE: The data set SASUSER.TEST has 1 observations and 6 variables.
NOTE: PROCEDURE COPY used (Total process time):
      real time           13.18 seconds
      cpu time            0.31 seconds


11
12   proc print data=test;
13   run;

ERROR: The value LONGVAR10 is not a valid SAS name.
NOTE: The SAS System stopped processing this step because of errors.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.15 seconds
      cpu time            0.01 seconds
```

△

Using the COPY Procedure instead of the COPY Statement

Generally, the COPY procedure functions the same as the COPY statement in the DATASETS procedure. The following is a list of differences:

- The IN= argument is required with PROC COPY. In the COPY statement, IN= is optional. If omitted, the default value is the libref of the procedure input library.
- PROC DATASETS cannot work with libraries that allow only sequential data access.
- The COPY statement honors the NOWARN option but PROC COPY does not.

Copying Generation Groups

You can use the COPY statement to copy an entire generation group. However, you cannot copy a specific version in a generation group.

Transporting SAS Data Sets between Hosts

You use the COPY procedure, along with the XPORT engine or a REMOTE engine, to transport SAS data sets between hosts. See “Strategies for Moving and Accessing SAS Files” in *Moving and Accessing SAS Files* for more information.

DELETE Statement

Deletes SAS files from a SAS library.

Featured in: Example 2 on page 374

DELETE *SAS-file-1 <...SAS-file-n>*
</ <ALTER=*alter-password*>
<GENNUM=ALL|HIST|REVERT|*integer*>
<MEMTYPE=*mtype*>>;

Required Arguments

SAS-file-1 <...SAS-file-n>
specifies one or more SAS files that you want to delete. You can also use a numbered range list or colon list. For more information, see “Data Set Lists” in the *SAS Language Reference: Concepts*.

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 355

GENNUM=ALL|HIST|REVERT|*integer*
restricts processing for generation data sets. You can use the option either in parentheses after the name of each SAS file or after a forward slash. The following is a list of valid values:

ALL
refers to the base version and all historical versions in a generation group.

HIST
refers to all historical versions, but excludes the base version in a generation group.

REVERT|0
deletes the base version and changes the most current historical version, if it exists, to the base version.

integer
is a number that references a specific version from a generation group. Specifying a positive number is an absolute reference to a specific generation number that is

appended to a data set's name; that is, **gennum=2** specifies MYDATA#002. Specifying a negative number is a relative reference to a historical version in relation to the base version, from the youngest to the oldest; that is, **gennum=-1** refers to the youngest historical version.

See also:

"Restricting Processing for Generation Data Sets" on page 358

"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 356

Featured in: Example 2 on page 374

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 following versions:

- delete the base version and all historical versions
- delete the base version and rename the youngest historical version to the base version
- delete an absolute version
- delete a relative version
- delete all historical versions and leave the base version

Deleting the Base Version and All Historical Versions

The following statements delete the base version and all historical versions where the data set name is A:

```
proc datasets;
   delete A(gennum=all);
```

```
proc datasets;
   delete A / gennum=all;

proc datasets gennum=all;
   delete A;
```

The following statements delete the base version and all historical versions where the data set name begins with the letter A:

```
proc datasets;
   delete A:(gennum=all);

proc datasets;
   delete A: / gennum=all;

proc datasets gennum=all;
   delete A:;
```

Deleting the Base Version and Renaming the Youngest Historical Version to the Base Version

The following statements delete the base version and rename the youngest historical version to the base version, where the data set name is A:

```
proc datasets;
   delete A(gennum=revert);

proc datasets;
   delete A / gennum=revert;

proc datasets gennum=revert;
   delete A;
```

The following statements delete the base version and rename the youngest historical version to the base version, where the data set name begins with the letter A:

```
proc datasets;
   delete A:(gennum=revert);

proc datasets;
   delete A: / gennum=revert;

proc datasets gennum=revert;
   delete A:;
```

Deleting a Version with an Absolute Number

The following statements use an absolute number to delete the first historical version:

```
proc datasets;
   delete A(gennum=1);

proc datasets;
   delete A / gennum=1;

proc datasets gennum=1;
   delete A;
```

The following statements delete a specific historical version, where the data set name begins with the letter A:

```
proc datasets;
   delete A:(gennum=1);

proc datasets;
   delete A: / gennum=1;

proc datasets gennum=1;
   delete A:;
```

Deleting a Version with a Relative Number

The following statements use a relative number to delete the youngest historical version, where the data set name is A:

```
proc datasets;
   delete A(gennum=-1);

proc datasets;
   delete A / gennum=-1;

proc datasets gennum=-1;
   delete A;
```

The following statements use a relative number to delete the youngest historical version, where the data set name begins with the letter A:

```
proc datasets;
   delete A:(gennum=-1);

proc datasets;
   delete A: / gennum=-1;

proc datasets gennum=-1;
   delete A:;
```

Deleting All Historical Versions and Leaving the Base Version

The following statements delete all historical versions and leave the base version, where the data set name is A:

```
proc datasets;
   delete A(gennum=hist);

proc datasets;
   delete A / gennum=hist;

proc datasets gennum=hist;
   delete A;
```

The following statements delete all historical versions and leave the base version, where the data set name begins with the letter A:

```
proc datasets;
   delete A:(gennum=hist);

proc datasets;
   delete A: / gennum=hist;

proc datasets gennum=hist;
   delete A:;
```

EXCHANGE Statement

Exchanges the names of two SAS files in a SAS library.

Featured in: Example 2 on page 374

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 355

MEMTYPE=*mtype*
restricts processing to one member type. You can exchange only the names of SAS files of the same type. You can use the option either in parentheses after the name of each SAS file or after a forward slash.

Default: If you do not specify MEMTYPE= in the PROC DATASETS statement, the default is ALL.

See also: "Restricting Member Types for Processing" on page 356

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 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 2 on page 374

EXCLUDE *SAS-file-1* <*...SAS-file-n*> </ MEMTYPE=*mtype*>;

Required Arguments

SAS-file-1 <...SAS-file-n>
specifies one or more SAS files to exclude from the copy operation. All SAS files you name in the EXCLUDE statement must be in the library that is specified in the IN= option in the COPY statement. If the SAS files are generation groups, the EXCLUDE statement allows only selection of the base versions.

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 356

"Specifying Member Types When Copying or Moving SAS Files" on page 324

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 25.

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 4 on page 382

FORMAT *variable-1* <*format-1*>

<…*variable-n* <*format-n*>>;

Required Arguments

***variable-1* <*...variable-n*>**
specifies one or more variables whose format you want to assign, change, or remove. If you want to disassociate a format with a variable, list the variable last in the list with no format following:

```
format x1-x3 4.1 time hhmm2.2 age;
```

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*.

Tip: To remove all formats from a data set, use the "ATTRIB Statement" on page 309 and the _ALL_ keyword.

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. The following is a list of valid values:

NOT NULL (*variable*)
specifies that *variable* does not contain a SAS missing value, including special missing values.

UNIQUE (*variables*)
specifies that the values of *variables* must be unique. This constraint is identical to DISTINCT.

DISTINCT (*variables*)
specifies that the values of *variables* must be unique. This constraint is identical to UNIQUE.

CHECK (WHERE-*expression*)

limits the data values of variables to a specific set, range, or list of values. This behavior is accomplished with a WHERE expression.

PRIMARY KEY (*variables*)
specifies a primary key, that is, a set of variables that do not contain missing values and whose values are unique.

Requirement: When defining overlapping primary key and foreign key constraints, which means that variables in a data file are part of both a primary key and a foreign key definition, if you use exactly the same variables, then the variables must be defined in a different order.

Interaction: A primary key affects the values of an individual data file until it has a foreign key referencing it.

FOREIGN KEY (*variables*) REFERENCES *table-name*
<ON DELETE *referential-action*> <ON UPDATE *referential-action*>
specifies a foreign key, that is, a set of variables whose values are linked to the values of the primary key variables in another data file. The referential actions are enforced when updates are made to the values of a primary key variable that is referenced by a foreign key.

There are three types of referential actions: RESTRICT, SET NULL, and CASCADE:

The following operations can be done with the RESTRICT referential action:

a delete operation
deletes the primary key row, but only if no foreign key values match the deleted value.

an update operation
updates the primary key value, but only if no foreign key values match the current value to be updated.

The following operations can be done with the SET NULL referential action:

a delete operation
deletes the primary key row and sets the corresponding foreign key values to NULL.

an update operation
modifies the primary key value and sets all matching foreign key values to NULL.

The following operations can be done with the CASCADE referential action:

an update operation
modifies the primary key value, and additionally modifies any matching foreign key values to the same value. CASCADE is not supported for delete operations.

Default: RESTRICT is the default action if no referential action is specified.

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.

Interaction: Before it enforces a SET NULL or CASCADE referential action, SAS checks to see whether there are other foreign keys that reference the primary key and that specify RESTRICT for the intended operation. If RESTRICT is specified, or if the constraint reverts to the default values, then RESTRICT is enforced for all foreign keys, unless no foreign key values match the values to updated or deleted.

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:

```
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-1* <*...constraint-name-n*> | _ALL_;

Arguments

constraint-name-1 <...constraint-name-n>
names one or more constraints to delete. For example, to delete the constraints Unique_D and Unique_E, use the following statement:

```
ic delete Unique_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 Statement

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-1 <...index-n>*
</ <REFRESH>
<UPDATECENTILES= ALWAYS|NEVER|*integer*>>;

Required Arguments

index-1 <..index-n>
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 percentage of the data values that can be changed before centiles for the indexed variables are updated.

The following is a list of valid values:

ALWAYS | 0
updates centiles when the data set is closed if any changes have been made to the data set index. You can specify ALWAYS or 0 and produce the same results.

NEVER | 101
does not update centiles. You can specify NEVER or 101 and produce the same results.

integer
is the percentage of values for the indexed variable that can be updated before centiles are refreshed.

Alias: UPDCEN

Default 5 (percent)

INDEX CREATE Statement

Creates simple or composite indexes for the SAS data set specified in the MODIFY statement.

Restriction: Must be in a MODIFY RUN group

See also: "Understanding SAS Indexes" in *SAS Language Reference: Concepts*

Featured in: Example 4 on page 382

INDEX CREATE *index-specification-1 <...index-specification-n>*
</ <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 4 on page 382

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 4 on page 382

UPDATECENTILES=ALWAYS|NEVER|*integer*
specifies when centiles are to be updated. It is not practical to update centiles after every data set update. Therefore, you can specify the percentage of the data values that can be changed before centiles for the indexed variables are updated. The following is a list of valid values:

ALWAYS | 0
updates centiles when the data set is closed if any changes have been made to the data set index. You can specify ALWAYS or 0 and produce the same results.

NEVER | 101
does not update centiles. You can specify NEVER or 101 and produce the same results.

integer
specifies the percentage of values for the indexed variable that can be updated before centiles are refreshed.

Alias: UPDCEN

Default: 5 (percent)

INDEX DELETE Statement

Deletes one or more indexes associated with the SAS data set specified in the MODIFY statement.

Restriction: Must appear in a MODIFY RUN group

INDEX DELETE *index-1* <*...index-n*> | _ALL_;

Required Arguments

index-1 <...index-n>
names one or more indexes to delete. The index(es) must be for variables in the SAS data set that is named in the preceding MODIFY statement. You can delete both simple and composite indexes.

ALL
deletes all indexes, except for indexes that are owned by an integrity constraint. When an index is created, it is marked as owned by the user, by an integrity constraint, or by both. If an index is owned by both a user and an integrity constraint, the index is not deleted until both an IC DELETE statement and an INDEX DELETE statement are processed.

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 4 on page 382

INFORMAT *variable-1 <informat-1>*
<…variable-n <informat-n>>;

Required Arguments

variable
specifies one or more variables whose informats you want to assign, change, or remove. If you want to disassociate an informat with a variable, list the variable last in the list with no informat following:

```
informat a b 2. x1-x3 4.1 c;
```

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*.

Tip: To remove all informats from a data set, use the "ATTRIB Statement" on page 309 and the _ALL_ keyword.

LABEL Statement

Assigns, changes, and removes variable labels for the SAS data set specified in the MODIFY statement.

Restriction: Must appear in a MODIFY RUN group

Featured in: Example 4 on page 382

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

Required Arguments

variable=<'label'>

specifies a text string of up to 256 characters. If the label text contains single quotation marks, use double quotation marks around the label, or use two single quotation marks in the label text and surround the string with single quotation marks. To remove a label from a data set, assign a label that is equal to a blank that is enclosed in quotation marks.

Range: 1 - 256 characters

Tip: To remove all variable labels in a data set, use the "ATTRIB Statement" on page 309 and the _ALL_ keyword.

MODIFY Statement

Changes the attributes of a SAS file and, through the use of subordinate statements, the attributes of variables in the SAS file.

Restriction: You cannot change the length of a variable using the LENGTH= option on an ATTRIB statement.

Featured in: Example 4 on page 382

MODIFY *SAS-file* <(*option-1* <...*option-n*>)>
</ <CORRECTENCODING=*encoding-value*>
<DTC=*SAS-date-time*>
<GENNUM=*integer*>
<MEMTYPE=*mtype*>>;

Action	Option
Restrict processing to a certain type of SAS file	MEMTYPE=
Specify data set 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	

Action	Option
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 346

CORRECTENCODING=*encoding-value*
enables you to change the encoding indicator, which is recorded in the file's descriptor information, in order to match the actual encoding of the file's data.

See: CORRECTENCODING= Option in the MODIFY Statement of the DATASETS Procedure in *SAS National Language Support (NLS): Reference Guide*

DTC=*SAS-date-time*
specifies a date and time to substitute for the date and time stamp placed on a SAS file at the time of creation. You cannot use this option in parentheses after the name of each SAS file; you must specify DTC= after a forward slash:

```
modify mydata / dtc='03MAR00:12:01:00'dt;
```

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.

Tip: Use DTC= to alter a SAS file's creation date and time before using the DATECOPY option in the COPY procedure, CPORT procedure, SORT procedure, and the COPY statement in the DATASETS procedure.

GENMAX=*number-of-generations*
specifies the maximum number of versions. Use this option in parentheses after the name of SAS file.

Default: 0

Range: 0 to 1,000

GENNUM=*integer*
restricts processing for generation data sets. You can specify GENNUM= either in parentheses after the name of each SAS file or after a forward slash. Valid value is *integer*, which is a number that references a specific version from a generation group. Specifying a positive number is an absolute reference to a specific generation number that is appended to a data set's name; that is, **`gennum=2`** specifies MYDATA#002. Specifying a negative number is a relative reference to a historical version in relation to the base version, from the youngest to the oldest; that is, **`gennum=-1`** refers to the youngest historical version. Specifying 0, which is the default, refers to the base version.

See 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 - 256 characters

Featured in: Example 4 on page 382

Tip: To remove all variable labels in a data set, use the "ATTRIB Statement" on page 309.

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 346

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 346

Featured in: Example 4 on page 382

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 indicator.

Restriction: The data must be sorted in the order that you specify. If the data is not in the specified order, SAS does not sort it for you.

Tip: When using the MODIFY SORTEDBY option, you can also use a numbered range list or colon list. For more information, see "Data Set Lists" in the *SAS Language Reference: Concepts*.

Featured in: Example 4 on page 382

TYPE=*special-type*
assigns or changes the special data set type of a SAS data set. SAS does *not* verify the following:

- the SAS data set type 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 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 346

Changing Data Set Labels and Variable Labels

The LABEL option can change either the data set label or a variable label within the data set. To change a data set label, use the following syntax:

```
modify datasetname(label='Label for Data Set');
run;
```

You can change one or more variable labels within a data set. To change a variable label within the data set, use the following syntax:

```
modify datasetname;
    label  variablename='Label for Variable';
run;
```

For an example of changing both a data set label and a variable label in the same PROC DATASETS, see Example 4 on page 382.

Manipulating Passwords

In order to assign, change, or remove a password, you must specify the password for the highest level of protection that currently exists on that file.

Assigning Passwords

```
/* assigns a password to an unprotected file */
modify colors (pw=green);


/* assigns an alter password to an already read-protected SAS data set */
modify colors (read=green alter=red);
```

Changing Passwords

```
/* changes the write password from YELLOW to BROWN */
modify cars (write=yellow/brown);


/* uses alter access to change unknown read password to BLUE */
modify colors (read=/blue alter=red);
```

Removing Passwords

```
/* removes the alter password RED from STATES */
modify states (alter=red/);


/* uses alter access to remove the read password */
modify zoology (read=green/ alter=red);


/* uses PW= as an alias for either WRITE= or ALTER= to remove unknown
   read password */
modify biology (read=/ pw=red);
```

Working with Generation Groups

Changing the Number of Generations

```
/* changes the number of generations on data set A to 99 */
modify A (genmax=99);
```

Removing Passwords

```
/* removes the alter password RED from STATES#002 */
modify states (alter=red/) / gennum=2;
```

REBUILD Statement

Specifies whether to restore or delete the disabled indexes and integrity constraints.

Default: Rebuild indexes and integrity constraints

Restriction: Data sets created in Version 7 or later

REBUILD *SAS-file* </ ALTER=*password* GENNUM=*n* MEMTYPE=*mtype* NOINDEX>;

Required Argument

SAS-file
specifies a SAS data file that contains the disabled indexes and integrity constraints. You can also use a numbered range list or colon list. For more information, see "Data Set Lists" in the *SAS Language Reference: Concepts*.

Options

ALTER=*alter-password*
provides the alter password for any alter-protected SAS files that are named in the REBUILD statement. You can use the option either in parentheses after the name of each SAS file or after a forward slash.

GENNUM=*integer*
restricts processing for generation data sets. You can use the option either in parentheses after the name of each SAS file or after a forward slash. Valid value is *integer*, which is a number that references a specific version from a generation group. Specifying a positive number is an absolute reference to a specific generation number that is appended to a data set's name; that is, **`gennum=2`** specifies MYDATA#002. Specifying a negative number is a relative reference to a historical version in relation to the base version, from the youngest to the oldest; that is, **`gennum=-1`** refers to the youngest historical version. Specifying 0, which is the default, refers to the base version.

See 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 REBUILD statement, the default is MEMTYPE=ALL.

NOINDEX
specifies to delete the disabled indexes and integrity constraints.

Restriction: The NOINDEX option cannot be used for data files that contain one or more referential integrity constraints.

Details

When the DLDMGACTION=NOINDEX data set or system option is specified and SAS encounters a damaged data file, SAS does the following:

- repairs the data file without indexes and integrity constraints
- deletes the index file
- updates the data file to reflect the disabled indexes and integrity constraints
- limits the data file to be opened only in INPUT mode
- writes the following warning to the SAS log:

```
WARNING: SAS data file MYLIB.MYFILE.DATA was damaged and
         has been partially repaired. To complete the repair,
         execute the DATASETS procedure REBUILD statement.
```

The REBUILD statement completes the repair of a damaged SAS data file by rebuilding or deleting all of the data file's disabled indexes and integrity constraints. The REBUILD statement establishes and uses member-level locking in order to process the new index file and to restore the indexes and integrity constraints.

To rebuild the index file and restore the indexes and integrity constraints, use the following code:

```
proc datasets library=mylib
rebuild myfile
/alter=password
 gennum=n
 memtype=mytype;
```

To delete the disabled indexes and integrity constraints, use the following code:

```
proc datasets library=mylib
rebuild myfile /noindex;
```

After you execute the REBUILD statement, the data file is no longer restricted to INPUT mode.

The REBUILD statement default is to rebuild the indexes and integrity constraints and the index file.

If a data file contains one or more referential integrity constraints and you use the NOINDEX option with the REBUILD statement, the following error message is written to the SAS log:

```
Error: Unable to rebuild data file MYLIB.MYFILE.DATA using the
              NOINDEX option because the data file contains referential
              constraints. Resubmit the REBUILD statement without the
              NOINDEX option to restore the data file.
```

RENAME Statement

Renames variables in the SAS data set specified in the MODIFY statement.

Restriction: Must appear in a MODIFY RUN group

Featured in: Example 4 on page 382

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-1 <...SAS-file-n>*
</ <ALTER=*alter-password*>
<GENNUM=*integer*>
<MEMTYPE=*mtype*>>;

Required Arguments

SAS-file-1 <...SAS-file-n>
specifies one or more SAS data sets or catalogs in the procedure input library. You can also use a numbered range list or colon list. For more information, see "Data Set Lists" in the *SAS Language Reference: Concepts*.

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 355

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 358

"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 356

Details

The most common situations that require the REPAIR statement are as follows:

- A system failure occurs while you are updating a SAS data set or catalog.
- The device on which a SAS data set or an associated index resides is damaged. In this case, you can restore the damaged data set or index from a backup device, but the data set and index no longer match.
- The disk that stores the SAS data set or catalog becomes full before the file is completely written to disk. You might need to free some disk space. PROC DATASETS requires free space when repairing SAS data sets with indexes and when repairing SAS catalogs.
- An I/O error occurs while you are writing a SAS data set or catalog entry.

When you use the REPAIR statement for SAS data sets, it recreates all indexes for the data set. It also attempts to restore the data set to a usable condition, but the restored data set might not include the last several updates that occurred before the system failed. You cannot use the REPAIR statement to 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 might not include the last updates that occurred before a system crash or an I/O error. The REPAIR statement issues warning messages for entries that might have truncated data.

To repair a damaged catalog, the version of SAS that you use must be able to update the catalog. Whether a SAS version can update a catalog (or just read it) is determined by the SAS version that created the catalog:

- A damaged Version 6 catalog can be repaired with Version 6 only.
- A damaged Version 8 catalog can be repaired with either Version 8 or SAS 9, but not with Version 6.

□ A damaged SAS 9 catalog can be repaired with SAS 9 only.

If the REPAIR operation is not successful, try to restore the SAS data set or catalog from your system's backup files.

If you issue a REPAIR statement for a SAS file that does not exist in the specified library, PROC DATASETS stops processing the run group that contains the REPAIR statement, and issues an error message. To override this behavior and continue processing, use the NOWARN option in the PROC DATASETS statement.

If you are using Cross-Environment Data Access (CEDA) to process a damaged foreign SAS data set, CEDA cannot repair it. CEDA does not support update processing, which is required in order to repair a damaged data set. To repair the foreign file, you must move it back to its native environment. Note that observations might be lost during the repair process. For more information about CEDA, refer to "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 3 on page 380

SAVE *SAS-file-1* <*...SAS-file-n*> </ MEMTYPE=*mtype*>;

Required Arguments

SAS-file-1 <...SAS-file-n>
specifies one or more SAS files that you do not want to delete from the SAS 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 356

Featured in: Example 3 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, make sure that you understand how the MEMTYPE= option affects which types of SAS files are saved and which types are deleted. △

- When you use the SAVE statement with generation groups, the SAVE statement treats the base version and all historical versions as a unit. You cannot save a specific version.

SELECT Statement

Selects SAS files for copying.

Restriction: Must follow a COPY statement
Restriction: Cannot appear with an EXCLUDE statement in the same COPY step
Featured in: Example 2 on page 374

SELECT *SAS-file-1* <*...SAS-file-n*>
</ <ALTER=*alter-password*>
<MEMTYPE= *mtype*>>;

Required Arguments

SAS-file-1 <...SAS-file-n>
specifies one or more SAS files that you want to copy. All of the SAS files that you name must be in the data library that is referenced by the libref named in the IN= option in the COPY statement. If the SAS files have generation groups, the SELECT statement allows only selection of the base versions.

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 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 355

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 324

"Restricting Member Types for Processing" on page 356

Featured in: Example 2 on page 374

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 25.

Concepts: DATASETS Procedure

Procedure Execution

Execution of Statements

When you start the DATASETS procedure, you specify the procedure input library in the PROC DATASETS statement. If you omit a procedure input library, the procedure processes the current default SAS library (usually the WORK library). To specify a new procedure input library, issue the DATASETS procedure again.

Statements execute in the order they are written. For example, if you want to see the contents of a data set, copy a data set, and then visually compare the contents of the second data set with the first, the statements that perform those tasks must appear in that order (that is, CONTENTS, COPY, CONTENTS).

RUN-Group Processing

PROC DATASETS supports RUN-group processing. RUN-group processing enables you to submit RUN groups without ending the procedure.

The DATASETS procedure supports four types of RUN groups. Each RUN group is defined by the statements that compose it and by what causes it to execute.

Some statements in PROC DATASETS act as *implied* RUN statements because they cause the RUN group preceding them to execute.

The following list discusses what statements compose a RUN group and what causes each RUN group to execute:

- The PROC DATASETS statement always executes immediately. No other statement is necessary to cause the PROC DATASETS statement to execute. Therefore, the PROC DATASETS statement alone is a RUN group.
- The MODIFY statement, and any of its subordinate statements, form a RUN group. These RUN groups always execute immediately. No other statement is necessary to cause a MODIFY RUN group to execute.
- The APPEND, CONTENTS, and COPY statements (including EXCLUDE and SELECT, if present), form their own separate RUN groups. Every APPEND statement forms a single-statement RUN group; every CONTENTS statement forms a single-statement RUN group; and every COPY step forms a RUN group. Any other statement in the procedure, except those that are subordinate to either the COPY or MODIFY statement, causes the RUN group to execute.

- One or more of the following statements form a RUN group:
  - AGE
  - CHANGE
  - DELETE
  - EXCHANGE
  - REPAIR
  - SAVE

  If any of these statements appear in sequence in the PROC step, the sequence forms a RUN group. For example, if a REPAIR statement appears immediately after a SAVE statement, the REPAIR statement does not force the SAVE statement to execute; it becomes part of the same RUN group. To execute the RUN group, submit one of the following statements:
  - PROC DATASETS
  - APPEND
  - CONTENTS
  - COPY
  - MODIFY
  - QUIT
  - RUN
  - another DATA or PROC step.

SAS reads the program statements that are associated with one task until it reaches a RUN statement or an implied RUN statement. SAS executes all of the preceding statements immediately and continues reading until it reaches another RUN statement or implied RUN statement. To execute the last task, you must use a RUN statement or a statement that stops the procedure.

The following PROC DATASETS step contains five RUN groups:

```
LIBNAME dest 'SAS-library';
   /* RUN group */

proc datasets;
      /* RUN group */
   change nutr=fatg;
   delete bldtest;
   exchange xray=chest;
      /* RUN group */
   copy out=dest;
      select report;
      /* RUN group */
   modify bp;
      label dias='Taken at Noon';
      rename weight=bodyfat;
      /* RUN group */
   append base=tissue data=newtiss;
quit;
```

Note: If you are running in interactive line mode, you can receive messages that statements have already executed before you submit a RUN statement. Plan your tasks carefully if you are using this environment for running PROC DATASETS. △

Error Handling

Generally, if an error occurs in a statement, the RUN group containing the error does not execute. RUN groups preceding or following the one containing the error execute normally. The MODIFY RUN group is an exception. If a syntax error occurs in a statement subordinate to the MODIFY statement, only the statement containing the error fails. The other statements in the RUN group execute.

Note that if the first word of the statement (the statement name) is in error and the procedure cannot recognize it, the procedure treats the statement as part of the preceding RUN group.

Password Errors

If there is an error involving an incorrect or omitted password in a statement, the error affects only the statement containing the error. The other statements in the RUN group execute.

Forcing a RUN Group with Errors to Execute

The FORCE option in the PROC DATASETS statement forces execution of the RUN group even if one or more of the statements contain errors. Only the statements that are error-free execute.

Ending the Procedure

To stop the DATASETS procedure, you must issue a QUIT statement, a RUN CANCEL statement, a new PROC statement, or a DATA statement. Submitting a QUIT statement executes any statements that have not executed. Submitting a RUN CANCEL statement cancels any statements that have not executed.

Using Passwords with the DATASETS Procedure

Several statements in the DATASETS procedure support options that manipulate passwords on SAS files. These options, ALTER=, PW=, READ=, and WRITE=, are also data set options.* 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 318 and "MODIFY Statement" on page 342. △

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:

* 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.

```
delete xplant bones(alter=red);
```

2 after a forward slash (/) in a subordinate statement. When you use a password option following a slash, the option refers to all SAS files named in the statement unless the same option appears in parentheses after the name of a SAS file. This method is convenient when you are working with more than one SAS file and they all have the same password.

   In the following statement, the ALTER= option in parentheses provides the password RED for the SAS file CHEST, and the ALTER= option after the slash provides the password BLUE for the SAS file VIRUS:

```
delete chest(alter=red) virus / alter=blue;
```

3 in the PROC DATASETS statement. Specifying the password in the PROC DATASETS statement can be useful if all the SAS files you are working with in the library have the same password. Do not specify the option in parentheses.

   In the following PROC DATASETS step, the PW= option provides the password RED for the SAS files INSULIN and ABNEG:

```
proc datasets pw=red;
   delete insulin;
   contents data=abneg;
run;
```

Note: For the password for a SAS file in a SELECT statement, SAS looks in the COPY statement before it looks in the PROC DATASETS statement. △

Restricting Member Types for Processing

In the PROC DATASETS Statement

If you reference more than one member type in subordinate statements and you have a specified member type in the PROC DATASETS statement, then include all of the member types in the PROC DATASETS statement. Only the member type or types in the original PROC DATASETS statement is in effect. The following example lists multiple member types:

```
proc datasets lib=library memtype=(data view);
```

In Subordinate Statements

Use the MEMTYPE= option in the following subordinate statements to limit the member types that are available for processing:

AGE

CHANGE

DELETE

EXCHANGE

EXCLUDE

REPAIR

SAVE

SELECT

Note: The MEMTYPE= option works slightly differently for the CONTENTS, COPY, and MODIFY statements. Refer to "CONTENTS Statement" on page 314, "COPY Statement" on page 318, and "MODIFY Statement" on page 342 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 17.7 on page 358 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 324. △

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 17.7 on page 358 shows the member types that you can use in each statement:

Table 17.7 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.

* For the APPEND and CONTENTS statements, use GENNUM= just as you use any SAS data set option, in parentheses after the SAS data set name.

Note: The GENNUM= option works slightly different for the MODIFY statement. See "MODIFY Statement" on page 342. △

Note: You cannot restrict processing to a specific version for the AGE, COPY, EXCHANGE, and SAVE statements. These statements apply to the entire generation group. △

SAS searches for a generation specification in the following order:

1. in parentheses after the name of the SAS data set in a subordinate statement. When used in parentheses, the option only refers to the name immediately preceding the option. If you are working with more than one SAS data set in a data library and you want a different generation version for each SAS data set, you must specify GENNUM= in parentheses after individual names.

   In the following statement, the GENNUM= option specifies the version of a generation group for the SAS data set BONES only:

   ```
   delete xplant bones (gennum=2);
   ```

2. after a forward slash (/) in a subordinate statement. When you use the GENNUM= option following a slash, the option refers to all SAS data sets named in the statement unless the same option appears in parentheses after the name of a SAS data set. This method is convenient when you are working with more than one file and you want the same version for all files.

   In the following statement, the GENNUM= option in parentheses specifies the generation version for SAS data set CHEST, and the GENNUM= option after the slash specifies the generation version for SAS data set VIRUS:

   ```
   delete chest (gennum=2) virus / gennum=1;
   ```

3. in the PROC DATASETS statement. Specifying the generation version in the PROC DATASETS statement can be useful if you want the same version for all of the SAS data sets you are working with in the library. Do not specify the option in parentheses.

   In the following PROC DATASETS step, the GENNUM= option specifies the generation version for the SAS files INSULIN and ABNEG:

   ```
   proc datasets gennum=2;
      delete insulin;
      contents data=abneg;
   run;
   ```

   Note: For the generation version for a SAS file in a SELECT statement, SAS looks in the COPY statement before it looks in the PROC DATASETS statement. △

Results: DATASETS Procedure

Directory Listing to the SAS Log

The PROC DATASETS statement lists the SAS files in the procedure input library unless the NOLIST option is specified. The NOLIST option prevents the creation of the

procedure results that go to the log. If you specify the MEMTYPE= option, only specified types are listed. If you specify the DETAILS option, PROC DATASETS prints these additional columns of information: **Obs, Entries or Indexes**, **Vars**, and **Label**.

Directory Listing as SAS Output

The CONTENTS statement lists the directory of the procedure input library if you use the DIRECTORY option or specify DATA=_ALL_.

If you want only a directory, use the NODS option and the _ALL_ keyword in the DATA= option. The NODS option suppresses the description of the SAS data sets; only the directory appears in the output.

Note: The CONTENTS statement does not put a directory in an output data set. If you try to create an output data set using the NODS option, you receive an empty output data set. Use the SQL procedure to create a SAS data set that contains information about a SAS library. △

Note: If you specify the ODS RTF destination, the PROC DATASETS output goes to both the SAS log and the ODS output area. The NOLIST option suppresses output to both. To see the output only in the SAS log, use the ODS EXCLUDE statement by specifying the member directory as the exclusion. △

Procedure Output

The CONTENTS Statement

The only statement in PROC DATASETS that produces procedure output is the CONTENTS statement. This section shows the output from the CONTENTS statement for the GROUP data set, which is shown in the following output.

Only the items in the output that require explanation are discussed.

Data Set Attributes

Here are descriptions of selected fields shown in the following output:

Member Type
is the type of library member (DATA or VIEW).

Protection
indicates whether the SAS data set is READ, WRITE, or ALTER password protected.

Data Set Type
names the special data set type (such as CORR, COV, SSPC, EST, or FACTOR), if any.

Observations
is the total number of observations currently in the file. Note that for a very large data set, if the number of observations exceeds the largest integer value that can be represented in a double precision floating point number, the count is shown as missing.

Deleted Observations
is the number of observations marked for deletion. These observations are not included in the total number of observations, shown in the **Observations** field.

Note that for a very large data set, if the number of deleted observations exceeds the number that can be stored in a double-precision integer, the count is shown as missing. Also, the count for **Deleted Observations** shows a missing value if you use the COMPRESS=YES option with one or both of the REUSE=YES and POINTOBS=NO options.

Compressed
indicates whether the data set is compressed. If the data set is compressed, the output includes an additional item, **Reuse Space** (with a value of YES or NO). This item indicates whether to reuse space that is made available when observations are deleted.

Sorted
indicates whether the data set is sorted. If you sort the data set with PROC SORT, PROC SQL, or specify sort information with the SORTEDBY= data set option, a value of YES appears here, and there is an additional section to the output. See "Sort Information" on page 364 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 17.3 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               0
Created              Wed, Sep 12, 2007 01:57:49 PM    Observation Length    96
Last Modified        Wed, Sep 12, 2007 01:57:49 PM    Deleted Observations  0
Protection                                            Compressed            NO
Data Set Type                                         Sorted                NO
Label
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 17.4 Engine and Operating Environment Dependent Information Section of CONTENTS Output

```
                      Engine/Host Dependent Information

Data Set Page Size          8192
Number of Data Set Pages    3
First Data Page             1
Max Obs per Page            84
Obs in First Data Page      63
Number of Data Set Repairs  0
Filename                    \myfiles\health\group.sas7bdat
Release Created             9.0201B0
Host Created                XP_PRO
```

Alphabetic List of Variables and Attributes

Here are descriptions of selected columns in the following output:

#
is the logical position of each variable in the observation. This number is assigned to the variable when the variable is defined.

Variable
is the name of each variable. By default, variables appear alphabetically.

Note: Variable names are sorted such that X1, X2, and X10 appear in that order and not in the true collating sequence of X1, X10, and X2. Variable names that contain an underscore and digits might appear in a nonstandard sort order. For example, P25 and P75 appear before P2_5. △

Type
specifies the type of variable: character or numeric.

Len
specifies the variable's length, which is the number of bytes used to store each of a variable's values in a SAS data set.

Transcode
specifies whether a character variable is transcoded. If the attribute is NO, then transcoding is suppressed. By default, character variables are transcoded when required. For more information on transcoding, see *SAS National Language Support (NLS): Reference Guide*.

Note: If none of the variables in the SAS data set has a format, informat, or label associated with it, or if all of the variables are set to TRANSCODE=YES, then the column for the attribute is NOT displayed. △

Output 17.5 Variable Attributes Section

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Informat
9	BIRTH	Num	8		
4	CITY	Char	15		
3	FNAME	Char	15		
10	HIRED	Num	8	DATE7.	DATE7.
11	HPHONE	Char	12		
1	IDNUM	Char	4		
7	JOBCODE	Char	4		
2	LNAME	Char	15		
8	SALARY	Num	8	COMMA8.	
6	SEX	Char	2		
5	STATE	Char	3		

Alphabetic List of Indexes and Attributes

The section shown in the following output appears only if the data set has indexes associated with it.

#
indicates the number of each index. The indexes are numbered sequentially as they are defined.

Index
displays the name of each index. For simple indexes, the name of the index is the same as a variable in the data set.

Unique Option
indicates whether the index must have unique values. If the column contains YES, the combination of values of the index variables is unique for each observation.

Nomiss Option
indicates whether the index excludes missing values for all index variables. If the column contains YES, the index does not contain observations with missing values for all index variables.

# of Unique Values
gives the number of unique values in the index.

Variables
names the variables in a composite index.

Output 17.6 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 the following output appears only if the **Sorted** field has a value of YES.

Sortedby
indicates how the data are currently sorted. This field contains either the variables and options 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 the data was sorted using PROC SORT or SORTEDBY. If PROC SORT or PROC SQL sorted the data set, the value is YES. If you assigned the sort indicator with the SORTEDBY= data set option, the value is NO.

Character Set
is the character set used to sort the data. The value for this field can be ASCII, EBCDIC, or PASCII.

Collating Sequence
is the collating sequence used to sort the data set, which can be a translation table name, an encoding value, or LINGUISTIC if the data set is sorted linguistically. This field does not appear if you do not specify a collating sequence that is different from the character set.

If the data set is sorted linguistically, additional linguistic collation sequence information displays after **Collating Sequence**, such as the locale, collation style, and so on. For a list of the collation rules that can be specified for linguistic collation, see the SORTSEQ= option in the "PROC SORT Statement" on page 1167 in the SORT procedure.

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 17.7 Sort Information Section

Sort Information

Sortedby	LNAME
Validated	YES
Character Set	ANSI
Collating Sequence	LINGUISTIC
Locale	en_US
Strength	3

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.

PROC CONTENTS generates the same ODS tables as PROC DATASETS with the CONTENTS statement.

Table 17.8 ODS Tables Produced by the DATASETS Procedure without the CONTENTS Statement

ODS Table	Description	Generates Table
Directory	General library information	unless you specify the NOLIST option.
Members	Library member information	unless you specify the NOLIST option.

Table 17.9 ODS Table Names Produced by PROC CONTENTS and PROC DATASETS with the CONTENTS Statement

ODS Table	Description	Generates Table
Attributes	Data set attributes	unless you specify the SHORT option.
Directory	General library information	if you specify DATA=<*libref*.>_ALL_ or the DIRECTORY option.*
EngineHost	Engine and operating environment information	unless you specify the SHORT option.
IntegrityConstraints	A detailed listing of integrity constraints	if the data set has integrity constraints and you do not specify the SHORT option.
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.*

ODS Table	Description	Generates Table
Position	A detailed listing of variables by logical position in the data set	if you specify the VARNUM option and you do not specify the SHORT option.
PositionShort	A concise listing of variables by logical position in the data set	if you specify the VARNUM option and the SHORT option.
Sortedby	Detailed sort information	if the data set is sorted and you do not specify the SHORT option.
SortedbyShort	Concise Sort information	if the data set is sorted and you specify the SHORT option.
Variables	A detailed listing of variables in alphabetical order	unless you specify the SHORT option.
VariablesShort	A concise listing of variables in alphabetical order	if you specify the SHORT option.

* For PROC DATASETS, if both the NOLIST option and either the DIRECTORY option or DATA=<*libref*.>_ALL_ are specified, then the NOLIST option is ignored.

Output Data Sets

The CONTENTS Statement

The CONTENTS statement is the only statement in the DATASETS procedure that generates output data sets.

The OUT= Data Set

The OUT= option in the CONTENTS statement creates an output data set. Each variable in each DATA= data set has one observation in the OUT= data set. Here are the variables in the output data set:

CHARSET
: the character set used to sort the data set. The value is ASCII, EBCDIC, or PASCII. A blank appears if the data set does not have a sort indicator stored with it.

COLLATE
: the collating sequence used to sort the data set. A blank appears if the sort indicator for the input data set does not include a collating sequence.

COMPRESS
: indicates whether the data set is compressed.

CRDATE
: date the data set was created.

DELOBS
: number of observations marked for deletion in the data set. (Observations can be marked for deletion but not actually deleted when you use the FSEDIT procedure of SAS/FSP software.)

ENCRYPT
: indicates whether the data set is encrypted.

ENGINE
name of the method used to read from and write to the data set.

FLAGS
indicates whether the variables in an SQL view are protected (**P**) or contribute (**C**) to a derived variable.

P indicates the variable is protected. The value of the variable can be displayed but not updated.

C indicates whether the variable contributes to a derived variable.

The value of FLAG is blank if **P** or **C** does not apply to an SQL view or if it is a data set view.

FORMAT
variable format. The value of FORMAT is a blank if you do not associate a format with the variable.

FORMATD
number of decimals 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 whether the data set can be addressed by observation.

PROTECT
the first letter of the level of protection. The value for PROTECT is one or more of the following:

A	indicates the data set is alter-protected.
R	indicates the data set is read-protected.
W	indicates the data set is write-protected.

REUSE
indicates whether the space made available when observations are deleted from a compressed data set should be reused. If the data set is not compressed, the REUSE variable has a value of NO.

SORTED
the value depends on the sorting characteristics of the input data set. The following are possible values:

. (period)	for not sorted.
0	for sorted but not validated.
1	for sorted and validated.

SORTEDBY
the value depends on that variable's role in the sort. The following are possible values:

. (period)
if the variable was not used to sort the input data set.

n
where *n* is an integer that denotes the position of that variable in the sort. A negative value of *n* indicates that the data set is sorted by the descending order of that variable.

TYPE
type of the variable (1=numeric, 2=character).

TYPEMEM
special data set type (blank if no TYPE= value is specified).

VARNUM
variable number in the data set. Variables are numbered in the order they appear.

The output data set is sorted by the variables LIBNAME and MEMNAME.

Note: The variable names are sorted so that the values X1, X2, and X10 are listed in that order, not in the true collating sequence of X1, X10, X2. Therefore, if you want to use a BY statement on MEMNAME in subsequent steps, run a PROC SORT step on the output data set first or use the NOTSORTED option in the BY statement. △

The following is an example of an output data set created from the GROUP data set, which is shown in Example 5 on page 384 and in "Procedure Output" on page 360.

Output 17.8 The Data Set HEALTH.GRPOUT

```
run;                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       4       7
  8 HEALTH    GROUP   Test Subjects           LNAME      2      15       2
  9 HEALTH    GROUP   Test Subjects           SALARY     1       8       8
 10 HEALTH    GROUP   Test Subjects           SEX        2       2       6
 11 HEALTH    GROUP   Test Subjects           STATE      2       3       5

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   24MAR05:10:29:38 24MAR05:13:08:53    0
  2    0      0   58   148   V9   24MAR05:10:29:38 24MAR05:13:08:53    0
  3    0      0   43   148   V9   24MAR05:10:29:38 24MAR05:13:08:53    0
  4    0      1   16   148   V9   24MAR05:10:29:38 24MAR05:13:08:53    0
  5    0      0   82   148   V9   24MAR05:10:29:38 24MAR05:13:08:53    0
  6    0      0   24   148   V9   24MAR05:10:29:38 24MAR05:13:08:53    0
  7    0      0   78   148   V9   24MAR05:10:29:38 24MAR05:13:08:53    0
  8    0      0   28   148   V9   24MAR05:10:29:38 24MAR05:13:08:53    0
  9    0      1    0   148   V9   24MAR05:10:29:38 24MAR05:13:08:53    0
 10    0      0   76   148   V9   24MAR05:10:29:38 24MAR05:13:08:53    0
 11    0      0   73   148   V9   24MAR05:10:29:38 24MAR05:13:08:53    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 8 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. Here are the variables in the output data set:

IC_OWN
contains YES if the index is owned by the integrity constraint.

INACTIVE
contains YES if the integrity constraint is inactive.

LIBNAME
libref used for the data library.

MEMNAME
SAS data set that contains the variable.

MG
the value of MESSAGE=, if it is used, in the IC CREATE statement.

MSGTYPE
the value is blank unless an integrity constraint is violated and you specified a message.

NAME
the name of the index or integrity constraint.

NOMISS
contains YES if the NOMISS option is defined for the index.

NUMVALS
the number of distinct values in the index (displayed for centiles).

NUMVARS
the number of variables involved in the index or integrity constraint.

ONDELETE
for a foreign key integrity constraint, contains RESTRICT or SET NULL if applicable (the ON DELETE option in the IC CREATE statement).

ONUPDATE

for a foreign key integrity constraint, contains RESTRICT or SET NULL if applicable (the ON UPDATE option in the IC CREATE statement).

RECREATE
the SAS statement necessary to 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, and so on).

UNIQUE
contains YES if the UNIQUE option is defined for the index.

UPERC
the percentage of the index that has been updated since the last refresh (displayed for centiles).

UPERCMX
the percentage of the index update that triggers a refresh (displayed for centiles).

WHERE
for a check integrity constraint, contains the WHERE statement.

Examples: DATASETS Procedure

Example 1: Removing All Labels and Formats in a Data Set

Procedure Features:

PROC CONTENTS
PROC DATASETS statement option:
MODIFY statement
ATTRIB
CONTENTS statement

The following example deletes all labels and formats within a data set.

Program

Set the following system options.

```
options ls=79 nodate nocenter;
title;
```

Create a user defined FORMAT with a value of CLSFMT.

```
proc format;
   value clsfmt 1='Freshman' 2='Sophmore' 3='Junior' 4='Senior';
```

```
run;
```

Create a data set named CLASS. Use the CLSFMT format on variable Z. Create labels for variables, X, Y, and Z.

```
data class;
   format z clsfmt.;
   label x='ID NUMBER'
      y='AGE'
      z='CLASS STATUS';
   input x y z;
datalines;
1 20 4
2 18 1
;
```

Use PROC CONTENTS to view the contents of the data set before removing the labels and format.

```
proc contents data=class;
run;
```

PROC CONTENTS with Labels and Format

```
The CONTENTS Procedure

Data Set Name        WORK.CLASS                      Observations          2
Member Type          DATA                            Variables             3
Engine               V9                              Indexes               0
Created              Friday, May 25, 2007 10:26:08 AM  Observation Length    24
Last Modified        Friday, May 25, 2007 10:26:08 AM  Deleted Observations  0
Protection                                           Compressed            NO
Data Set Type                                        Sorted                NO
Label
Data Representation  WINDOWS_32
Encoding             wlatin1  Western (Windows)


                      Engine/Host Dependent Information

Data Set Page Size          4096
Number of Data Set Pages    1
First Data Page             1
Max Obs per Page            168
Obs in First Data Page      2
Number of Data Set Repairs  0
File Name                   C:\DOCUME~1\mydir\LOCALS~1\Temp\SAS Temporary
                            Files\_TD3964\class.sas7bdat
Release Created             9.0201B0
Host Created                XP_PRO


      Alphabetic List of Variables and Attributes

#    Variable    Type    Len    Format     Label

2    x           Num       8               ID NUMBER
3    y           Num       8               AGE
1    z           Num       8    CLSFMT.    CLASS STATUS
```

Within PROC DATASETS, remove all the labels and formats using the MODIFY statement and the ATTRIB option.

```
proc datasets lib=work memtype=data;
   modify class;
     attrib _all_ label=' ';
     attrib _all_ format=;
run;
```

Use the CONTENTS statement within PROC DATASETS to view the contents of the data set without the labels and format.

```
contents data=class;
run;
quit;
```

CONTENTS Statement without Labels and Format

```
The DATASETS Procedure

Data Set Name        WORK.CLASS                      Observations          2
Member Type          DATA                            Variables             3
Engine               V9                              Indexes               0
Created              Friday, May 25, 2007 10:26:08 AM  Observation Length    24
Last Modified        Friday, May 25, 2007 10:26:08 AM  Deleted Observations  0
Protection                                           Compressed            NO
Data Set Type                                        Sorted                NO
Label
Data Representation  WINDOWS_32
Encoding             wlatin1  Western (Windows)


                      Engine/Host Dependent Information

Data Set Page Size          4096
Number of Data Set Pages    1
First Data Page             1
Max Obs per Page            168
Obs in First Data Page      2
Number of Data Set Repairs  0
File Name                   C:\DOCUME~1\mydir\LOCALS~1\Temp\SAS Temporary
                            Files\_TD3964\class.sas7bdat
Release Created             9.0201B0
Host Created                XP_PRO


Alphabetic List of Variables and Attributes

#    Variable    Type    Len

2    x           Num       8
3    y           Num       8
1    z           Num       8
```

Example 2: Manipulating SAS Files

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 does the following actions:

- changes the names of SAS files
- copies SAS files between SAS libraries
- deletes SAS files
- selects SAS files to copy
- exchanges the names of SAS files
- excludes SAS files from a copy operation

Program

Write the programming statements to the SAS log. The SOURCE system option accomplishes this.

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

```
LIBNAME dest1 'SAS-library-1';
LIBNAME dest2 'SAS-library-2';
LIBNAME health 'SAS-library-3';
```

Specify the procedure input library, and add more details to the directory. DETAILS prints these additional columns in the directory: **`Obs, Entries or Indexes`**, **`Vars`**, and **`Label`**. All member types are available for processing because the MEMTYPE= option does not appear in the PROC DATASETS statement.

```
proc datasets library=health details;
```

Delete two files in the library, and modify the names of a SAS data set and a catalog. The DELETE statement deletes the TENSION data set and the A2 catalog. MT=CATALOG applies only to A2 and is necessary because the default member type for the DELETE statement is DATA. The CHANGE statement changes the name of the A1 catalog to POSTDRUG. The EXCHANGE statement exchanges the names of the WEIGHT and BODYFAT data sets. MEMTYPE= is not necessary in the CHANGE or EXCHANGE statement because the default is MEMTYPE=ALL for each statement.

```
delete tension a2(mt=catalog);
change a1=postdrug;
exchange weight=bodyfat;
```

Restrict processing to one member type and delete and move data views. MEMTYPE=VIEW restricts processing to SAS views. MOVE specifies that all SAS views named in the SELECT statements in this step be deleted from the HEALTH data library and moved to the DEST1 data library.

```
copy out=dest1 move memtype=view;
```

Move the SAS view SPDATA from the HEALTH data library to the DEST1 data library.

```
   select spdata;
```

Move the catalogs to another data library. The SELECT statement specifies that the catalogs ETEST1 through ETEST5 be moved from the HEALTH data library to the DEST1 data library. MEMTYPE=CATALOG overrides the MEMTYPE=VIEW option in the COPY statement.

```
   select etest1-etest5 / memtype=catalog;
```

Exclude all files with specified criteria from processing. The EXCLUDE statement excludes from the COPY operation all SAS files that begin with the letter D and the other SAS files listed. All remaining SAS files in the HEALTH data library are copied to the DEST2 data library.

```
copy out=dest2;
   exclude d: mlscl oxygen test2 vision weight;
quit;
```

SAS Log

```
117    options pagesize=60 linesize=80 nodate pageno=1 source;
118  LIBNAME dest1 'c:\Documents and Settings\mydir\My
118! Documents\procdatasets\dest1';
NOTE: Libref DEST1 was successfully assigned as follows:
      Engine:        V9
      Physical Name: c:\Documents and Settings\mydir\My
      Documents\procdatasets\dest1
119  LIBNAME dest2 'c:\Documents and Settings\mydir\My
119! Documents\procdatasets\dest2';
NOTE: Libref DEST2 was successfully assigned as follows:
      Engine:        V9
      Physical Name: c:\Documents and Settings\mydir\My
      Documents\procdatasets\dest2
120  LIBNAME health 'c:\Documents and Settings\mydir\My
120! Documents\procdatasets\health';
NOTE: Libref HEALTH was successfully assigned as follows:
      Engine:        V9
      Physical Name: c:\Documents and Settings\mydir\My
      Documents\procdatasets\health


121  proc datasets library=health details;
                                    Directory

Libref         HEALTH
Engine         V9
Physical Name  \myfiles\health
Filename       \myfiles\health
```

#	Name	Member Type	Obs, Entries or Indexes	Vars	Label
1	A1	CATALOG	23		
2	ALL	DATA	23	17	
3	BODYFAT	DATA	1	2	
4	CONFOUND	DATA	8	4	
5	CORONARY	DATA	39	4	
6	DRUG1	DATA	6	2	JAN2005 DATA
7	DRUG2	DATA	13	2	MAY2005 DATA
8	DRUG3	DATA	11	2	JUL2005 DATA
9	DRUG4	DATA	7	2	JAN2002 DATA
10	DRUG5	DATA	1	2	JUL2002 DATA
11	ETEST1	CATALOG	1		
12	ETEST2	CATALOG	1		
13	ETEST3	CATALOG	1		
14	ETEST4	CATALOG	1		
15	ETEST5	CATALOG	1		
16	ETESTS	CATALOG	1		
17	FORMATS	CATALOG	6		
18	GROUP	DATA	148	11	
19	GRPOUT	DATA	11	40	
20	INFANT	DATA	149	6	
21	MLSCL	DATA	32	4	Multiple Sclerosis Data
22	NAMES	DATA	7	4	
23	OXYGEN	DATA	31	7	
24	PERSONL	DATA	148	11	
25	PHARM	DATA	6	3	Sugar Study
26	POINTS	DATA	6	6	
27	RESULTS	DATA	10	5	
28	SLEEP	DATA	108	6	
29	SPDATA	VIEW	.	2	
30	TEST2	DATA	15	5	
31	TRAIN	DATA	7	2	
32	VISION	DATA	16	3	
33	WEIGHT	DATA	83	13	California Results
34	WGHT	DATA	83	13	

#	File Size	Last Modified
1	62464	07Mar05:14:36:20
2	13312	12Sep07:13:57:48
3	5120	12Sep07:13:57:48
4	5120	12Sep07:13:57:48
5	5120	12Sep07:13:57:48
6	5120	12Sep07:13:57:49
7	5120	12Sep07:13:57:49
8	5120	12Sep07:13:57:49
9	5120	12Sep07:13:57:49
10	5120	12Sep07:13:57:49
11	17408	04Jan02:14:20:16
12	17408	04Jan02:14:20:16
13	17408	04Jan02:14:20:16
14	17408	04Jan02:14:20:16
15	17408	04Jan02:14:20:16
16	17408	24Mar05:16:12:20
17	17408	24Mar05:16:12:20
18	25600	12Sep07:13:57:50
19	17408	24Mar05:15:33:31
20	17408	12Sep07:13:57:51
21	5120	12Sep07:13:57:50
22	5120	12Sep07:13:57:50
23	9216	12Sep07:13:57:50
24	25600	12Sep07:13:57:51
25	5120	12Sep07:13:57:51
26	5120	12Sep07:13:57:51
27	5120	12Sep07:13:57:52
28	9216	12Sep07:13:57:52
29	5120	24Mar05:16:12:21
30	5120	12Sep07:13:57:52
31	5120	12Sep07:13:57:53
32	5120	12Sep07:13:57:53
33	13312	12Sep07:13:57:53
34	13312	12Sep07:13:57:53

```
122  delete tension a2(mt=catalog);
123  change a1=postdrug;
124  exchange weight=bodyfat;
NOTE: Changing the name HEALTH.A1 to HEALTH.POSTDRUG (memtype=CATALOG).
NOTE: Exchanging the names HEALTH.WEIGHT and HEALTH.BODYFAT (memtype=DATA).
125  copy out=dest1 move memtype=view;
126  select spdata;
127
128  select etest1-etest5 / memtype=catalog;
NOTE: Moving HEALTH.SPDATA to DEST1.SPDATA (memtype=VIEW).
NOTE: Moving HEALTH.ETEST1 to DEST1.ETEST1 (memtype=CATALOG).
NOTE: Moving HEALTH.ETEST2 to DEST1.ETEST2 (memtype=CATALOG).
NOTE: Moving HEALTH.ETEST3 to DEST1.ETEST3 (memtype=CATALOG).
NOTE: Moving HEALTH.ETEST4 to DEST1.ETEST4 (memtype=CATALOG).
NOTE: Moving HEALTH.ETEST5 to DEST1.ETEST5 (memtype=CATALOG).
129  copy out=dest2;
130  exclude d: mlscl oxygen test2 vision weight;
131  quit;

NOTE: Copying HEALTH.ALL to DEST2.ALL (memtype=DATA).
NOTE: There were 23 observations read from the data set HEALTH.ALL.
NOTE: The data set DEST2.ALL has 23 observations and 17 variables.
NOTE: Copying HEALTH.BODYFAT to DEST2.BODYFAT (memtype=DATA).
NOTE: There were 83 observations read from the data set HEALTH.BODYFAT.
NOTE: The data set DEST2.BODYFAT has 83 observations and 13 variables.
NOTE: Copying HEALTH.CONFOUND to DEST2.CONFOUND (memtype=DATA).
NOTE: There were 8 observations read from the data set HEALTH.CONFOUND.
NOTE: The data set DEST2.CONFOUND has 8 observations and 4 variables.
NOTE: Copying HEALTH.CORONARY to DEST2.CORONARY (memtype=DATA).
NOTE: There were 39 observations read from the data set HEALTH.CORONARY.
NOTE: The data set DEST2.CORONARY has 39 observations and 4 variables.
NOTE: Copying HEALTH.ETESTS to DEST2.ETESTS (memtype=CATALOG).
NOTE: Copying HEALTH.FORMATS to DEST2.FORMATS (memtype=CATALOG).
NOTE: Copying HEALTH.GROUP to DEST2.GROUP (memtype=DATA).
NOTE: There were 148 observations read from the data set HEALTH.GROUP.
NOTE: The data set DEST2.GROUP has 148 observations and 11 variables.
NOTE: Copying HEALTH.GRPOUT to DEST2.GRPOUT (memtype=DATA).
NOTE: There were 11 observations read from the data set HEALTH.GRPOUT.
NOTE: The data set DEST2.GRPOUT has 11 observations and 40 variables.
NOTE: Copying HEALTH.INFANT to DEST2.INFANT (memtype=DATA).
NOTE: There were 149 observations read from the data set HEALTH.INFANT.
NOTE: The data set DEST2.INFANT has 149 observations and 6 variables.
NOTE: Copying HEALTH.NAMES to DEST2.NAMES (memtype=DATA).
NOTE: There were 7 observations read from the data set HEALTH.NAMES.
NOTE: The data set DEST2.NAMES has 7 observations and 4 variables.
NOTE: Copying HEALTH.PERSONL to DEST2.PERSONL (memtype=DATA).
NOTE: There were 148 observations read from the data set HEALTH.PERSONL.
NOTE: The data set DEST2.PERSONL has 148 observations and 11 variables.
NOTE: Copying HEALTH.PHARM to DEST2.PHARM (memtype=DATA).
NOTE: There were 6 observations read from the data set HEALTH.PHARM.
NOTE: The data set DEST2.PHARM has 6 observations and 3 variables.
NOTE: Copying HEALTH.POINTS to DEST2.POINTS (memtype=DATA).
NOTE: There were 6 observations read from the data set HEALTH.POINTS.
NOTE: The data set DEST2.POINTS has 6 observations and 6 variables.
NOTE: Copying HEALTH.POSTDRUG to DEST2.POSTDRUG (memtype=CATALOG).
NOTE: Copying HEALTH.RESULTS to DEST2.RESULTS (memtype=DATA).
NOTE: There were 10 observations read from the data set HEALTH.RESULTS.
NOTE: The data set DEST2.RESULTS has 10 observations and 5 variables.
NOTE: Copying HEALTH.SLEEP to DEST2.SLEEP (memtype=DATA).
NOTE: There were 108 observations read from the data set HEALTH.SLEEP.
NOTE: The data set DEST2.SLEEP has 108 observations and 6 variables.
NOTE: Copying HEALTH.TRAIN to DEST2.TRAIN (memtype=DATA).
NOTE: There were 7 observations read from the data set HEALTH.TRAIN.
NOTE: The data set DEST2.TRAIN has 7 observations and 2 variables.
NOTE: Copying HEALTH.WGHT to DEST2.WGHT (memtype=DATA).
NOTE: There were 83 observations read from the data set HEALTH.WGHT.
NOTE: The data set DEST2.WGHT has 83 observations and 13 variables.
NOTE: PROCEDURE DATASETS used (Total process time):
      real time           44.04 seconds
      cpu time            0.60 seconds
```

Example 3: Saving SAS Files from Deletion

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-library';
```

Specify the procedure input library to process.

```
proc datasets lib=elder;
```

Save the data sets CHRONIC, AGING, and CLINICS, and delete all other SAS files (of all types) in the ELDER library. MEMTYPE=DATA is necessary because the ELDER library has a catalog named CLINICS and a data set named CLINICS.

```
      save chronic aging clinics / memtype=data;
run;
```

SAS Log

```
161
162   options pagesize=40 linesize=80 nodate pageno=1 source;
163  LIBNAME elder 'c:\Documents and Settings\mydir\My
163! Documents\procdatasets\elder';
NOTE: Libref ELDER was successfully assigned as follows:
      Engine:        V9
      Physical Name: c:\Documents and Settings\mydir\My
      Documents\procdatasets\elder
164  LIBNAME green 'c:\Documents and Settings\mydir\My
164! Documents\procdatasets\green';
NOTE: Libref GREEN was successfully assigned as follows:
      Engine:        V9
      Physical Name: c:\Documents and Settings\mydir\My
      Documents\procdatasets\green
165  proc copy in=green out=elder;

NOTE: Copying GREEN.AGING to ELDER.AGING (memtype=DATA).
NOTE: There were 1 observations read from the data set GREEN.AGING.
NOTE: The data set ELDER.AGING has 1 observations and 2 variables.
NOTE: Copying GREEN.ALCOHOL to ELDER.ALCOHOL (memtype=DATA).
NOTE: There were 1 observations read from the data set GREEN.ALCOHOL.
NOTE: The data set ELDER.ALCOHOL has 1 observations and 2 variables.
NOTE: Copying GREEN.BACKPAIN to ELDER.BACKPAIN (memtype=DATA).
NOTE: There were 1 observations read from the data set GREEN.BACKPAIN.
NOTE: The data set ELDER.BACKPAIN has 1 observations and 2 variables.
NOTE: Copying GREEN.CHRONIC to ELDER.CHRONIC (memtype=DATA).
NOTE: There were 1 observations read from the data set GREEN.CHRONIC.
NOTE: The data set ELDER.CHRONIC has 1 observations and 2 variables.
NOTE: Copying GREEN.CLINICS to ELDER.CLINICS (memtype=CATALOG).
NOTE: Copying GREEN.CLINICS to ELDER.CLINICS (memtype=DATA).
NOTE: There were 1 observations read from the data set GREEN.CLINICS.
NOTE: The data set ELDER.CLINICS has 1 observations and 2 variables.
NOTE: Copying GREEN.DISEASE to ELDER.DISEASE (memtype=DATA).
NOTE: There were 1 observations read from the data set GREEN.DISEASE.
NOTE: The data set ELDER.DISEASE has 1 observations and 2 variables.
NOTE: Copying GREEN.GROWTH to ELDER.GROWTH (memtype=DATA).
NOTE: There were 1 observations read from the data set GREEN.GROWTH.
NOTE: The data set ELDER.GROWTH has 1 observations and 2 variables.
NOTE: Copying GREEN.HOSPITAL to ELDER.HOSPITAL (memtype=CATALOG).
NOTE: PROCEDURE COPY used (Total process time):
      real time           2.42 seconds
      cpu time            0.04 seconds


166  proc datasets lib=elder;
                                    Directory

Libref         ELDER
Engine         V9
Physical Name  \myfiles\elder
Filename       \myfiles\elder


                      Member     File
           #  Name     Type       Size  Last Modified

           1  AGING    DATA       5120  12Sep07:15:52:52
           2  ALCOHOL  DATA       5120  12Sep07:15:52:52
           3  BACKPAIN DATA       5120  12Sep07:15:52:53
           4  CHRONIC  DATA       5120  12Sep07:15:52:53
           5  CLINICS  CATALOG   17408  12Sep07:15:52:53
           6  CLINICS  DATA       5120  12Sep07:15:52:53
           7  DISEASE  DATA       5120  12Sep07:15:52:54
           8  GROWTH   DATA       5120  12Sep07:15:52:54
           9  HOSPITAL CATALOG   17408  12Sep07:15:52:54
```

```
167  save chronic aging clinics / memtype=data;
168  run;

NOTE: Saving ELDER.CHRONIC (memtype=DATA).
NOTE: Saving ELDER.AGING (memtype=DATA).
NOTE: Saving ELDER.CLINICS (memtype=DATA).
NOTE: Deleting ELDER.ALCOHOL (memtype=DATA).
NOTE: Deleting ELDER.BACKPAIN (memtype=DATA).
NOTE: Deleting ELDER.CLINICS (memtype=CATALOG).
NOTE: Deleting ELDER.DISEASE (memtype=DATA).
NOTE: Deleting ELDER.GROWTH (memtype=DATA).
NOTE: Deleting ELDER.HOSPITAL (memtype=CATALOG).
```

Example 4: Modifying SAS Data Sets

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 5 on page 384 shows the modifications to the GROUP data set.

This example includes the following actions:

- modifying SAS files
- labeling a SAS data set
- adding a READ password to a SAS data set
- indicating how a SAS data set is currently sorted
- creating an index for a SAS data set
- assigning informats and formats to variables in a SAS data set
- renaming variables in a SAS data set
- labeling variables in a SAS data set

Program

Write the programming statements to the SAS log. SAS option SOURCE writes the programming statements to the log.

```
options pagesize=40 linesize=80 nodate pageno=1 source;


LIBNAME health 'SAS-library';
```

Specify HEALTH as the procedure input library to process. NOLIST suppresses the directory listing for the HEALTH data library.

```
proc datasets library=health nolist;
```

Add a label to a data set, assign a READ password, and specify how to sort the data. LABEL= adds a data set label to the data set GROUP. READ= assigns GREEN as the read password. The password appears as Xs in the SAS log. SAS issues a warning message if you specify a level of password protection on a SAS file that does not include alter protection. SORTEDBY= specifies how the data is sorted.

```
   modify group (label='Test Subjects' read=green sortedby=lname);
```

Create the composite index VITAL on the variables BIRTH and SALARY for the GROUP data set. NOMISS excludes all observations that have missing values for BIRTH and SALARY from the index. UNIQUE specifies that the index is created only if each observation has a unique combination of values for BIRTH and SALARY.

```
      index create vital=(birth salary) / nomiss unique;
```

Assign an informat and format, respectively, to the BIRTH variable.

```
      informat birth date7.;
      format birth date7.;
```

Assign a label to the variable SALARY.

```
      label salary='current salary excluding bonus';
```

Rename a variable, and assign a label. Modify the data set OXYGEN by renaming the variable OXYGEN to INTAKE and assigning a label to the variable INTAKE.

```
   modify oxygen;
      rename oxygen=intake;
      label intake='Intake Measurement';
quit;
```

SAS Log

```
169  options pagesize=40 linesize=80 nodate pageno=1 source;
170  LIBNAME health 'c:\Documents and Settings\mydir\My
170! Documents\procdatasets\health';
NOTE: Libref HEALTH was successfully assigned as follows:
      Engine:        V9
      Physical Name: c:\Documents and Settings\mydir\My
      Documents\procdatasets\health

NOTE: PROCEDURE DATASETS used (Total process time):
      real time           8:06.11
      cpu time            0.54 seconds


171  proc datasets library=health nolist;
172   modify group (label='Test Subjects' read=XXXXX sortedby=lname);
WARNING: The file HEALTH.GROUP.DATA is not ALTER protected.  It could be
         deleted or replaced without knowing the password.
173   index create vital=(birth salary) / nomiss unique;
NOTE: Composite index vital has been defined.
NOTE: MODIFY was successful for HEALTH.GROUP.DATA.
174  informat birth date7.;
175  format birth date7.;
176  label salary='current salary excluding bonus';
177  modify oxygen;
178  rename oxygen=intake;
NOTE: Renaming variable oxygen to intake.
179  label intake='Intake Measurement';
180  quit;

NOTE: MODIFY was successful for HEALTH.OXYGEN.DATA.
NOTE: PROCEDURE DATASETS used (Total process time):
      real time           15.09 seconds
      cpu time            0.06 seconds
```

Example 5: Describing a SAS Data Set

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 4 on page 382.

Program

```
options pagesize=40 linesize=80 nodate pageno=1;
```

```
LIBNAME health 'SAS-library';
```

Specify HEALTH as the procedure input library, and suppress the directory listing.

```
proc datasets library=health nolist;
```

Create the output data set GRPOUT from the data set GROUP. Specify GROUP as the data set to describe, give read access to the GROUP data set, and create the output data set GRPOUT, which appears in "The OUT= Data Set" on page 366.

```
   contents data=group (read=green) out=grpout;
   title  'The Contents of the GROUP Data Set';
run;
```

Output

Output 17.9 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              Wed, Sep 12, 2007 01:57:49 PM   Observation Length    96
Last Modified        Wed, Sep 12, 2007 04:01:15 PM   Deleted Observations  0
Protection           READ                            Compressed            NO
Data Set Type                                        Sorted                YES
Label                Test Subjects
Data Representation  WINDOWS_32
Encoding             wlatin1  Western (Windows)


                       Engine/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      63
Index File Page Size        4096
Number of Index File Pages  2
Number of Data Set Repairs  0
Filename                    c:\Documents and Settings\mydir\My
                            Documents\procdatasets\health\group.sas7bdat
Release Created             9.0201B0
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.

                      The Contents of the GROUP Data Set                     2

                           The DATASETS Procedure

                  Alphabetic List of Variables and Attributes

 #   Variable   Type   Len   Format    Informat   Label

11   HPHONE     Char    12
 1   IDNUM      Char     4
 7   JOBCODE    Char     4
 2   LNAME      Char    15
 8   SALARY     Num      8   COMMA8.              current salary excluding bonus
 6   SEX        Char     2
 5   STATE      Char     3


                   Alphabetic List of Indexes and Attributes

                                                 # 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 6: 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-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 17.10

```
                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 7: 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-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 8: 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 is directed.

```
ods output attributes=atr
           variables=var
           enginehost=eng;
```

Temporarily suppress output to the lst.

```
ods listing close;

proc contents data=a;
run;
```

Resume output to the lst.

```
ods listing;

title2 "all Attributes data";

proc print data=atr noobs;
run;

title2 "all Variables data";

proc print data=var noobs;
run;

title2 "all EngineHost data";

proc print data=eng noobs;
run;
```

Select specific data from ODS output.

```
ods output attributes=atr1(keep=member cvalue1 label1
   where=(attribute in ('Data Representation','Encoding'))
   rename=(label1=attribute cvalue1=value))
   attributes=atr2(keep=member cvalue2 label2
   where=(attribute in ('Observations', 'Variables'))
   rename=(label2=attribute cvalue2=value));


ods listing close;

proc contents data=a;
run;

ods listing;

data final;
   set atr1 atr2;
run;

title2 "example of post-processing of ODS output data";

proc print data=final noobs;
run;

ods listing close;
```

Results

Output 17.11 PROC CONTENTS ODS Output

PROC CONTENTS ODS Output
all Attributes data

Member	Label1	cValue1	nValue1	Label2	cValue2	nValue2
WORK.A	Data Set Name	WORK.A	.	Observations	1	1.000000
WORK.A	Member Type	DATA	.	Variables	1	1.000000
WORK.A	Engine	V9	.	Indexes	0	0
WORK.A	Created	Thu, Feb 08, 2007 12:38:50 PM	1486557531	Observation Length	8	8.000000
WORK.A	Last Modified	Thu, Feb 08, 2007 12:38:50 PM	1486557531	Deleted Observations	0	0
WORK.A	Protection		.	Compressed	NO	.
WORK.A	Data Set Type		.	Sorted	NO	.
WORK.A	Label		.			0
WORK.A	Data Representation	WINDOWS_32	.			0
WORK.A	Encoding	wlatin1 Western (Windows)	.			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	cValue1	nValue1
WORK.A	Data Set Page Size	4096	4096.000000
WORK.A	Number of Data Set Pages	1	1.000000
WORK.A	First Data Page	1	1.000000
WORK.A	Max Obs per Page	501	501.000000
WORK.A	Obs in First Data Page	1	1.000000
WORK.A	Number of Data Set Repairs	0	0
WORK.A	File Name	C:\\a.sas7bdat .	
WORK.A	Release Created	9.0201B0	.
WORK.A	Host Created	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, see *SAS Output Delivery System: User's Guide*.

Example 9: Getting Sort Indicator Information

Procedure features:

APPEND statement option:

GETSORT

SORTEDBY data set option

Program

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

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

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

Create another data set with the same structure, but with many observations. Sort the data set.

```
data phull;
    length var1 8.;
    do var1=1 to 100000;
    output;
end;
run;

proc sort data=phull;
    by DESCENDING var1;
run;

proc append base=mtea data=phull getsort;
run;

ods select sortedby;

proc contents data=mtea;
run;
```

Output 17.12 Sort Information Output

```
Sort Information
Sortedby DESCENDING var1
Validated YES
Character Set ANSI
```

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

A sort indicator is being created using the SORTEDBY data set option.

```
data mysort(sortedby=var1);
    length var1 8.;
    do var1=1 to 10;
    output;
end;
run;

ods select sortedby;

proc contents data=mysort;
run;
```

Output 17.13 Sort Information Output

```
  Sort Information
  Sortedby var1
  Validated NO
  Character Set ANSI
```

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

A sort indicator is being created by PROC SORT.

```
data mysort;
    length var1 8.;
    do var1=1 to 10;
    output;
end;
run;

proc sort data=mysort;
    by var1;
run;

ods select sortedby;

proc contents data=mysort;
run;
```

Output 17.14 Sort Information Output

```
  Sort Information
  Sortedby var1
  Validated YES
  Character Set ANSI
```

Example 10: Using the ORDER= Option with the CONTENTS Statement

Procedure features:

CONTENTS statement options:

ORDER=
COLLATE
CASECOLLATE
IGNORECASE
VARNUM

Program

Set up the data set.

```
options nonotes nodate nonumber nocenter formdlim  ='-';

data test;
   d=2;
   b001 =1;
   b002 =2;
   b003 =3;
   b001z=1;
   B001a=2;
   CaSeSeNsItIvE2=9;
   CASESENSITIVE3=9;
   D=2;
   casesensitive1=9;
   CaSeSeNsItIvE1a=9;
   d001z=1;
   CASESENSITIVE1C=9;
   D001a=2;
   casesensitive1b=9;
   A =1;
   a002 =2;
   a =3;
   a001z=1;
   A001a=2;

run;
```

To produce PROC CONTENTS output for a data set of your choice, change data set name to MYDATA.

```
%let mydata=WORK.test;

ods output Variables=var1(keep=Num Variable);
ods listing close;
```

```
proc contents data=&mydata;
run;

ods listing;
    title "Default options";

proc print data=var1 noobs;
run;

ods output Variables=var2(keep=Num Variable);
ods listing close;

proc contents order=collate data=&mydata;
run;

ods listing;
    title "order=collate option";

proc print data=var2 noobs;
run;

ods output Variables=var3(keep=Num Variable);
ods listing close;

proc contents order=casecollate data=&mydata;
run;

ods listing;
    title "order=casecollate option";

proc print data=var3 noobs;
run;

ods output Variables=var4(keep=Num Variable);
ods listing close;

proc contents order=ignorecase data=&mydata;
run;

ods listing;
    title "order=ignorecase option";

proc print data=var4 noobs;
run;
```

Note that the name of the ODS output object is different when the varnum option is used.

```
ods output Position=var5(keep=Num Variable);
ods listing close;

proc contents data=&mydata varnum;
run;
```

```
ods listing;
    title "varnum option";

proc print data=var5 noobs;
run;
```

Results

The following table shows the results of the ORDER= default, the COLLATE option, and the CASECOLLATE option:

Table 17.10 Using the COLLATE and CASECOLLATE Options

default		COLLATE		CASECOLLATE	
Num	**Variable**	**Num**	**Variable**	**Num**	**Variable**
15	**A**	**15**	**A**	**15**	**A**
18	**A001a**	**18**	**A001a**	**18**	**A001a**
6	**B001a**	**6**	**B001a**	**17**	**a001z**
8	**CASESENSITIVE3**	**12**	**CASESENSITIVE1C**	**16**	**a002**
12	**CASESENSITIVE1C**	**8**	**CASESENSITIVE3**	**2**	**b001**
7	**CaSeSeNsItIvE2**	**10**	**CaSeSeNsItIvE1a**	**6**	**B001a**
10	**CaSeSeNsItIvE1a**	**7**	**CaSeSeNsItIvE2**	**5**	**b001z**
13	**D001a**	**13**	**D001a**	**3**	**b002**
16	**a002**	**17**	**a001z**	**4**	**b003**
17	**a001z**	**16**	**a002**	**9**	**casesensitive1**
2	**b001**	**2**	**b001**	**10**	**CaSeSeNsItIvE1a**
3	**b002**	**5**	**b001z**	**14**	**casesensitive1b**
4	**b003**	**3**	**b002**	**12**	**CASESENSITIVE1C**
5	**b001z**	**4**	**b003**	**7**	**CaSeSeNsItIvE2**
9	**casesensitive1**	**9**	**casesensitive1**	**8**	**CASESENSITIVE3**
14	**casesensitive1b**	**14**	**casesensitive1b**	**1**	**d**
1	**d**	**1**	**d**	**13**	**D001a**
11	**d001z**	**11**	**d001z**	**11**	**d001z**

The following table shows the results of the ORDER= default, IGNORECASE option, and VARNUM option.

Table 17.11 Results of Using the IGNORECASE and VARNUM Options

default		IGNORECASE		VARNUM	
Num	**Variable**	**Num**	**Variable**	**Num**	**Variable**
15	A	15	A	1	d
18	A001a	16	a002	2	b001
6	B001a	18	A001a	3	b002
8	CASESENSITIVE3	17	a001z	4	b003
12	CASESENSITIVE1C	2	b001	5	b001z
7	CaSeSeNsItIvE2	3	b002	6	B001a
10	CaSeSeNsItIvE1a	4	b003	7	CaSeSeNsItIvE2
13	D001a	6	B001a	8	CASESENSITIVE3
16	a002	5	b001z	9	casesensitive1
17	a001z	9	casesensitive1	10	CaSeSeNsItIvE1a
2	b001	7	CaSeSeNsItIvE2	11	d001z
3	b002	8	CASESENSITIVE3	12	CASESENSITIVE1C
4	b003	10	CaSeSeNsItIvE1a	13	D001a
5	b001z	14	casesensitive1b	14	casesensitive1b
9	casesensitive1	12	CASESENSITIVE1C	15	A
14	casesensitive1b	1	d	16	a002
1	d	13	D001a	17	a001z
11	d001z	11	d001z	18	A001a

CHAPTER
18

The DBCSTAB Procedure

Information about the DBCSTAB Procedure **399**

Information about the DBCSTAB Procedure

See: For documentation about the DBCSTAB procedure, see *SAS National Language Support (NLS): Reference Guide*.

CHAPTER
19

The DISPLAY Procedure

Overview: DISPLAY Procedure **401**
Syntax: DISPLAY Procedure **401**
PROC DISPLAY Statement **401**
Example: DISPLAY Procedure **402**
Example 1: Executing a SAS/AF Application **402**

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 402

PROC DISPLAY CATALOG=*libref.catalog.entry.type* <BATCH>;

Required Argument

CATALOG=*libref.catalog.entry.type*
specifies a four-level name for the catalog entry.

libref
specifies the SAS library where the catalog is stored.

catalog
specifies the name of the catalog.

entry
specifies the name of the entry.

type
specifies the entry's type, which is one of the following. For details, see the description of catalog entry types in the BUILD procedure in online Help.

CBT
FRAME
HELP
MENU
PROGRAM
SCL

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 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;
```

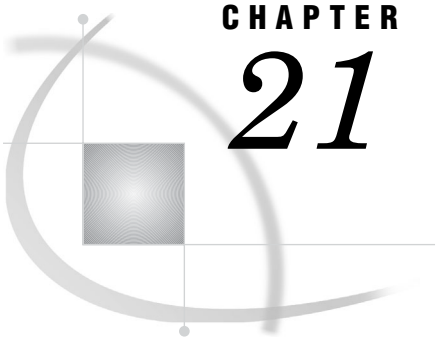
CHAPTER
20

The DOCUMENT Procedure

Information about the DOCUMENT Procedure **405**

Information about the DOCUMENT Procedure

See: For complete documentation about the DOCUMENT procedure, see *SAS Output Delivery System: User's Guide*.



CHAPTER 21

The EXPLODE Procedure

Information about the EXPLODE Procedure **407**

Information about the EXPLODE Procedure

See: For documentation about the EXPLODE procedure, go to **http://support.sas.com/documentation/onlinedoc/base/91/explode.pdf**.

CHAPTER
22

The EXPORT Procedure

Overview: EXPORT Procedure **409**
Syntax: EXPORT Procedure **409**
PROC EXPORT Statement **410**
Examples: EXPORT Procedure **412**
Example 1: Exporting a Delimited External File **412**
Example 2: Exporting a Subset of Observations to a CSV File **415**

Overview: EXPORT Procedure

PROC EXPORT reads data from a SAS data set and writes it to an external data source. External data sources can include such files as Microsoft Access Databases, Microsoft Excel Workbooks, Lotus spreadsheets, and delimited files. In delimited files, a delimiter such as a blank, comma, or tab separates columns of data values.

The EXPORT procedure uses one of these methods to export data:

- generated DATA step code
- generated SAS/ACCESS code
- translation engines

You control the results with options and statements that are specific to the output data source. The EXPORT procedure generates the specified output file and writes information about the export to the SAS log. The log displays the DATA step or the SAS/ACCESS code that the EXPORT procedure generates. If a translation engine is used, then no code is submitted.

You can also use the Export Wizard to guide you through the steps to export a SAS data set. The Export Wizard can generate EXPORT procedure statements, which you can save to a file for subsequent use. To open the Export Wizard, from the SAS windowing environment, select **File ▸ Export Data**. For more information about the Export Wizard, see the Base SAS online Help and documentation.

Syntax: EXPORT Procedure

Restriction: The EXPORT procedure is available for the following operating environments:

- Windows
- OpenVMS for Integrity servers
- UNIX

Table of Contents:

PROC EXPORT DATA=<*libref.*>*SAS data-set* <(*SAS data-set-options*)>
OUTFILE="*filename*" | OUTTABLE="*tablename*"
<DBMS=*identifier*> <REPLACE><LABEL>;
<*data-source-statement(s)*;>

PROC EXPORT Statement

Featured in: "Examples: EXPORT Procedure" on page 412

PROC EXPORT
DATA=<*libref.*>*SAS data-set* <(*SAS data-set-options*)>
OUTFILE="*filename*" | OUTTABLE="*tablename*"
<DBMS=*identifier*> <LABEL><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, the EXPORT procedure uses either the USER library (if assigned) or the WORK library.

The EXPORT procedure can export a SAS data set only if the data target supports the format of a SAS data set. The amount of data must also be within the limitations of the data target. For example, some data files have a maximum number of rows or columns. Some data files cannot support SAS user-defined formats and informats. If the SAS data set that you want to export exceeds the limits of the target file, the EXPORT procedure might not be able to export it correctly. In many cases, the procedure attempts to convert the data to the best of its ability. However, conversion is not possible for some types.

Default: If you do not specify a SAS data set to export, the EXPORT procedure uses the most recently created SAS data set. SAS keeps track of the data sets with the system variable _LAST_. To be certain that the EXPORT procedure uses the correct data set, you should identify the SAS data set.

Featured in: "Examples: EXPORT Procedure" on page 412

(*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 options. To export a subset of data that meets a specified condition, you can use the WHERE option. For information about SAS data set options, see "Data Set Options" in the *SAS Language Reference: Dictionary*.

Featured in: Example 2 on page 415

OUTFILE= '*filename*'
specifies the complete path and filename or a fileref for the output PC file, spreadsheet, or delimited external file. If you specify a fileref, or if the complete path and filename do not include special characters (such as the backslash in a path), lowercase characters, or spaces, you can omit the quotation marks. A fileref is a SAS

name that is associated with the physical location of a file. To assign a fileref, use the FILENAME statement. For more information about PC file formats, see *SAS/ACCESS Interface to PC Files: Reference*.

Alias: FILE

Restriction: The EXPORT procedure does not support device types or access methods for the FILENAME statement except for DISK. For example, the EXPORT procedure does not support the TEMP device type, which creates a temporary external file.

Featured in: Example 1 on page 412 and Example 2 on page 415.

OUTTABLE=*'tablename'*

specifies the table name of the output DBMS table. If the name does not include special characters (such as question marks), lowercase characters, or spaces, you can omit the quotation marks. Note that the DBMS table name might be case sensitive.

Requirement: When you export a DBMS table, you must specify the DBMS option.

Options

DBMS=*identifier*

specifies the type of data to export. To export a DBMS table, you must specify the DBMS option by using a valid database identifier. Valid identifiers for delimited data files are CSV, DLM, and TAB. For DBMS=DLM, the default delimiter character is a space. However, you can use DELIMITER='char'

The following values are valid for the DBMS= option:

Identifier	Output Data Source	Extension	Host Availability
CSV	delimited file (comma-separated values)	.csv	OpenVMS, UNIX, Microsoft Windows
DLM	delimited file (default delimiter is a blank)	.*	OpenVMS, UNIX, Microsoft Windows
TAB	delimited file (tab-delimited values)	.txt	OpenVMS, UNIX, Microsoft Windows

Restriction: The availability of an output external data source depends on these conditions:

- the operating environment, and in some cases the platform, as specified in the previous table
- whether your site has a license for SAS/ACCESS Interface to PC Files. If you do not have a license, only delimited files are available.

Featured in: Example 1 on page 412

LABEL

specifies a variable label name. SAS writes these to the exported table as column names. If the label names do not already exist, SAS writes them to the exported table.

REPLACE
overwrites an existing file. If you do not specify REPLACE, the EXPORT procedure does not overwrite an existing file.

Featured in: Example 2 on page 415

Data Source Statements

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 the DELIMITER option, the EXPORT procedure assumes that the delimiter is a blank.

Featured in: Example 1 on page 412

Interaction: You do not have to specify the DELIMITER option if

- DBMS=CSV
- DBMS=TAB
- output filename has an extension of .CSV
- output filename has an extension of .TXT

Examples: EXPORT Procedure

Example 1: Exporting a Delimited External File

Procedure features:
The EXPORT procedure statement arguments:

DATA=
DBMS=
OUTFILE=

Data source statement:

DELIMITER=

This example exports the SAS data set SASHELP.CLASS to a delimited external file.

Output 22.1 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

This example exports the SASHELP.CLASS data set and specifies the output filename. Note that the filename does not contain an extension. DBMS=DLM specifies that the output file is a delimited file. The DELIMITER option specifies that an & (ampersand) will delimit data fields in the output file.

```
proc export data=sashelp.class
   outfile='c:\myfiles\class'
   dbms=dlm;
   delimiter='&';
 run;
```

SAS Log

The SAS log displays this information about the successful export, including 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 DATA step 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:
      Filename=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: External File

The EXPORT procedure produces this external file:

```
Name&Sex&Age&Height&Weight
Alfred&M&14&69&112.5
Alice&F&13&56.5&84
Barbara&F&13&65.3&98
Carol&F&14&62.8&102.5
Henry&M&14&63.5&102.5
James&M&12&57.3&83
Jane&F&12&59.8&84.5
Janet&F&15&62.5&112.5
Jeffrey&M&13&62.5&84
John&M&12&59&99.5
Joyce&F&11&51.3&50.5
Judy&F&14&64.3&90
Louise&F&12&56.3&77
Mary&F&15&66.5&112
Philip&M&16&72&150
Robert&M&12&64.8&128
Ronald&M&15&67&133
Thomas&M&11&57.5&85
William&M&15&66.5&112
```

Example 2: Exporting a Subset of Observations to a CSV File

Procedure features:

The EXPORT procedure statement arguments:

DATA=
DBMS=
OUTFILE=
REPLACE

Program

This example exports the SAS data set, see PROC PRINT output Output 22.1i. The WHERE option requests a subset of the observations.The OUTFILE option specifies the output file. The DBMS option specifies that the output file is a CSV file, and overwrites the target CSV, if it exists.

```
proc export data=sashelp.class (where=(sex='F'))
   outfile='c:\myfiles\Femalelist.csv'
   dbms=csv
   replace;
run;
```

CHAPTER

23

The FCMP Procedure

Overview: FCMP Procedure **420**
What Does the FCMP Procedure Do? **420**
Syntax: FCMP Procedure **420**
PROC FCMP Statement **422**
ABORT Statement **424**
ARRAY Statement **424**
ATTRIB Statement **426**
DELETEFUNC | DELETESUBR Statement **426**
FUNCTION Statement **427**
LABEL Statement **429**
LISTFUNC | LISTSUBR Statement **429**
STRUCT Statement **429**
SUBROUTINE Statement **430**
OUTARGS Statement **431**
Concepts: FCMP Procedure **431**
Creating Functions and Subroutines **432**
Creating Functions and Subroutines: An Example **432**
Writing Your Own Functions **433**
Advantages of Writing Your Own Functions and CALL Routines **433**
Writing a User-Defined Function **433**
Using Library Options **434**
Declaring Functions **434**
Declaring CALL Routines **435**
Writing Program Statements **435**
PROC FCMP and DATA Step Differences **435**
Overview of PROC FCMP and DATA Step Differences **435**
Differences between PROC FCMP and the DATA Step **435**
ABORT Statement **436**
Arrays **436**
Data Set Input and Output **436**
DATA Step Debugger **436**
DO Statement **436**
File Input and Output **436**
IF Expressions **436**
PUT Statement **437**
WHEN and OTHERWISE Statements **437**
Additional Features in PROC FCMP **438**
PROC REPORT and Compute Blocks **438**
The FCmp Function Editor **438**
Computing Implicit Values of a Function **438**
PROC FCMP and Microsoft Excel **438**

Working with Arrays **438**
Passing Arrays **438**
Resizing Arrays **439**
Reading Arrays and Writing Arrays to a Data Set **439**
Overview **439**
The READ_ARRAY Function **439**
Syntax of the READ_ARRAY Function **439**
Details **440**
Example of the READ_ARRAY Function **440**
The WRITE_ARRAY Function **441**
Syntax of the WRITE_ARRAY Function **441**
Example 1: Using the WRITE_ARRAY Function with a PROC FCMP Array Variable **441**
Example 2: Using the WRITE_ARRAY Function to Specify Column Names **442**
Using Macros with PROC FCMP Routines **442**
Variable Scope in PROC FCMP Routines **442**
The Concept of Variable Scope **442**
When Local Variables in Different Routines Have the Same Name **443**
Recursion **443**
Directory Transversal **445**
Overview of Directory Transversal **445**
Directory Transversal Example **445**
Opening and Closing a Directory **446**
Gathering Filenames **446**
Calling DIR_ENTRIES from a DATA Step **447**
Identifying the Location of Compiled Functions and Subroutines: The CMPLIB= System Option **448**
Overview of the CMPLIB= System Option **448**
Syntax of the CMPLIB= System Option **448**
Example 1: Setting the CMPLIB= System Option **449**
Example 2: Compiling and Using Functions **449**
Special Functions and CALL Routines: Overview **451**
Special Functions and CALL Routines: Matrix CALL Routines **451**
CALL Routines and Matrix Operations **451**
ADDMATRIX CALL Routine **452**
CHOL CALL Routine **453**
DET CALL Routine **454**
ELEMMULT CALL Routine **455**
EXPMATRIX CALL Routine **456**
FILLMATRIX CALL Routine **457**
IDENTITY CALL Routine **457**
INV CALL Routine **458**
MULT CALL Routine **459**
POWER CALL Routine **460**
SUBTRACTMATRIX CALL Routine **461**
TRANSPOSE CALL Routine **462**
ZEROMATRIX CALL Routine **462**
Special Functions and CALL Routines: C Helper Functions and CALL Routines **463**
C Helper Functions and CALL Routines **463**
ISNULL C Helper Function **463**
Overview of the ISNULL C Helper Function **463**
Syntax of the ISNULL C Helper Function **463**
Example 1: Generating a Linked List **464**
Example 2: Using the ISNULL C Helper Function in a Loop **464**
SETNULL C Helper CALL Routine **465**
Overview of the SETNULL C Helper CALL Routine **465**

Syntax of the SETNULL C Helper CALL Routine **465**
Example: Setting an Element in a Linklist to Null **465**
STRUCTINDEX C Helper CALL Routine **465**
Overview of the STRUCTINDEX C Helper CALL Routine **465**
Syntax of the STRUCTINDEX C Helper CALL Routine **465**
Example: Setting Point Structures in an Array **466**
Special Functions and CALL Routines: Other Functions **467**
The SOLVE Function **467**
Overview of the SOLVE Function **467**
Syntax of the SOLVE Function **467**
Details of the SOLVE Function **468**
Example 1: Computing a Square Root Value **468**
Example 2: Calculating the Garman-Kohlhagen Implied Volatility **469**
Example 3: Calculating the Black-Scholes Implied Volatility **470**
The DYNAMIC_ARRAY Subroutine **471**
Overview of the DYNAMIC_ARRAY Subroutine **471**
Syntax of the DYNAMIC_ARRAY Subroutine **471**
Details **471**
Example: Creating a Temporary Array **472**
Functions for Calling SAS Code from Within Functions **472**
The RUN_MACRO Function **472**
Syntax of the RUN_MACRO Function **472**
Example 1: Executing a Predefined Macro with PROC FCMP **473**
Example 2: Executing a DATA Step within a DATA Step **474**
The RUN_SASFILE Function **476**
The Syntax of the RUN_SASFILE Function **476**
Example **476**
The FCmp Function Editor **477**
Introduction to the FCmp Function Editor **477**
Open the FCmp Function Editor **477**
Working with Existing Functions **478**
Open a Function **479**
Opening Multiple Functions **480**
Move a Function **480**
Close a Function **481**
Duplicate a Function **481**
Export a Function to a File **482**
Rename a Function **482**
Delete a Function **482**
Print a Function **483**
Creating a New Function **483**
Viewing the Log Window, Function Browser, and Data Explorer **485**
Log Window **485**
Function Browser **486**
Data Explorer **488**
Using the Function You Select in Your DATA Step Program **488**
Examples: FCMP Procedure **488**
Example 1: Creating a Function and Calling the Function from a DATA Step **488**
Example 2: Creating a CALL Routine and a Function **490**
Example 3: Executing PROC STANDARDIZE on Each Row of a Data Set **491**
Example 4: Using GTL with User-Defined Functions **493**

Overview: FCMP Procedure

What Does the FCMP Procedure Do?

The SAS Function Compiler (FCMP) procedure enables you to create, test, and store SAS functions, CALL routines, and subroutines before you use them in other SAS procedures or DATA steps. PROC FCMP provides the ability to build functions, CALL routines, and subroutines using DATA step syntax that is stored in a data set. The procedure accepts slight variations of DATA step statements, and you can use most features of the SAS programming language in functions and CALL routines that are created by PROC FCMP. You can call PROC FCMP functions and CALL routines from the DATA step just as you would any other SAS function, CALL routine or subroutine. This feature enables programmers to more easily read, write, and maintain complex code with independent and reusable subroutines. You can reuse the PROC FCMP routines in any DATA step or SAS procedure that has access to their storage location.

You can use the functions and subroutines that you create in PROC FCMP with the DATA step, the WHERE statement, the Output Delivery System (ODS), and with the following procedures:

- PROC CALIS
- PROC COMPILE
- PROC COMPUTAB
- PROC GA
- PROC GENMOD
- PROC MCMC
- PROC MODEL
- PROC NLIN
- PROC NLMIXED
- PROC NLP
- PROC PHREG
- PROC REPORT COMPUTE blocks
- Risk Dimensions procedures
- PROC SIMILARITY
- PROC SQL (functions with array arguments are not supported)

For more information about using PROC FCMP with ODS, see the *SAS Output Delivery System: User's Guide*.

Syntax: FCMP Procedure

PROC FCMP *options*;
ABORT;

ARRAY *array-name*[*dimensions*] </NOSYMBOLS | *variables* | *constants* | (*initial-values)*>;

ATTRIB *variables* <FORMAT=*format-name* LABEL=*'label'* LENGTH=*length*>;

DELETEFUNC | DELETESUBR *function-name*;

FUNCTION *function-name(argument-1, ..., argument-n)* <VARARGS > <$> <*length*> <KIND | GROUP=*'string'*>;

LABEL *variable='label'*;

LISTFUNC | LISTSUBR *function-name*;

STRUCT *structure-name variable*;

SUBROUTINE *subroutine-name (argument-1, ..., argument-n)* <VARARGS> <**LABEL**=*'label'*> <KIND | GROUP=*'string'*>;
OUTARGS *out-argument-1, ..., out-argument-n*;

Task	Statement
Create, test, and store SAS functions for use by other SAS procedures.	"PROC FCMP Statement" on page 422
Terminate the execution of the current DATA step, SAS job, or SAS session.	"ABORT Statement" on page 424
Associate a name with a list of variables and constants.	"ARRAY Statement" on page 424
Specify format, label, and length information for a variable.	"ATTRIB Statement" on page 426
Delete a function from the function library that is specified in the OUTLIB option.	"DELETEFUNC | DELETESUBR Statement" on page 426
Return changed variable values.	"FUNCTION Statement" on page 427
Specify a label for variables.	"LABEL Statement" on page 429
Write the source code of a function in the SAS listing.	"LISTFUNC | LISTSUBR Statement" on page 429
Declare (create) structure types.	"STRUCT Statement" on page 429
Declare (create) independent computational blocks of code.	"SUBROUTINE Statement" on page 430
(Use only with the SUBROUTINE statement.) Specify arguments from the argument list that the subroutine should update.	"OUTARGS Statement" on page 431

You can use DATA step statements with PROC FCMP. However, there are some differences in the syntax and functionality for PROC FCMP. See "PROC FCMP and DATA Step Differences" on page 435 for a list of statements and the differences.

The behaviors of the DROP, KEEP, FORMAT, and LENGTH statements are the same in PROC FCMP and in the DATA step.

The following DATA step statements are *not* supported in PROC FCMP:

- DATA
- SET

- MERGE
- UPDATE
- MODIFY
- INPUT
- INFILE

The support for the FILE statement is limited to LOG and PRINT destinations in PROC FCMP. The OUTPUT statement is supported in PROC FCMP, but it is not supported within a function or subroutine.

The following statements are supported in PROC FCMP but not in the DATA step:

- FUNCTION
- STRUCT
- SUBROUTINE
- OUTARGS

PROC FCMP Statement

PROC FCMP *options*;

Task	Option
Encode the source code in a data set.	ENCRYPT | HIDE on page 423
Specify printing a message for each statement in a program as it is executed.	FLOW on page 423
Link previously compiled libraries.	LIBRARY | INLIB= on page 423
Specify that both the LISTSOURCE and the LISTPROG options are in effect.	LIST on page 423
Specify that the LISTCODE, LISTPROG, and LISTSOURCE options are in effect.	LISTALL on page 423
Specify the printing of compiled program code.	LISTCODE on page 423
List the prototypes and subroutines for all visible FCMP functions in the SAS listing.	LISTFUNCS on page 423
Specify the printing of compiled programs.	LISTPROG on page 423
Specify the printing of source code statements.	LISTSOURCE on page 424
Specify the name of an output data set to which compiled subroutines and functions are written.	OUTLIB on page 424
Specify printing the result of each statement in a program as it is executed.	PRINT on page 424
Specify printing the results of each operation in each statement in a program as it is executed.	TRACE on page 424

Options

ENCRYPT | HIDE
specifies to encode the source code in a data set.

FLOW
specifies printing a message for each statement in a program as it is executed. This option produces extensive output.

LIBRARY | INLIB=*library.dataset*
LIBRARY | INLIB=(*(library-1.dataset library-2.dataset ... library-n.dataset)*
LIBRARY | INLIB=*library.datasetM - library.datasetN*
specifies that previously compiled libraries are to be linked into the program. These libraries are created by a previous PROC FCMP step or by using PROC PROTO (for external C routines).

Tip: Libraries are created by the OUTLIB= option and are stored as members of a SAS library that have the type CMPSUB. Only subroutines and functions are read into the program when you use the LIBRARY= option.

Tip: If the routines that are being declared do not call PROC FCMP routines in other packages, then you do not need to specify the INLIB= option.

Use the *libref.dataset* format to specify the two-level name of a library. The *libref* and *dataset* names must be valid SAS names that are not longer than eight characters. You can specify a list of files with the LIBRARY= option, and you can specify a range of names by using numeric suffixes. When you specify more than one file, you must enclose the list in parentheses, except in the case of a single range of names. The following are syntax examples:

```
proc fcmp library=sasuser.exsubs;
proc fcmp library=(sasuser.exsubs work.examples);
proc fcmp library=lib1-lib10;
```

LIST
specifies that both the LISTSOURCE and the LISTPROG options are in effect.

Tip: Printing both the source code and the compiled code and then comparing the two listings of assignment statements is one way of verifying that the assignments were compiled correctly.

LISTALL
specifies that the LISTCODE, LISTPROG, and LISTSOURCE options are in effect.

LISTCODE
specifies that the compiled program code be printed. LISTCODE lists the chain of operations that are generated by the compiler.

Tip: Because LISTCODE output is somewhat difficult to read, use the LISTPROG option to obtain a more readable listing of the compiled program code.

LISTFUNCS
specifies that prototypes for all visible FCMP functions or subroutines be written to the SAS listing.

LISTPROG
specifies that the compiled program be printed. The listing for assignment statements is generated from the chain of operations that are generated by the compiler. The source statement text is printed for other statements.

Tip: The expressions that are printed by the LISTPROG option do not necessarily represent the way that the expression is actually calculated, because intermediate results for common subexpressions can be re-used. However, the expressions are printed in expanded form by the LISTPROG option. To see how the expression is actually evaluated, refer to the listing from the LISTCODE option.

LISTSOURCE
specifies that source code statements for the program be printed.

OUTLIB=*libname.dataset.package*
specifies the three-level name of an output data set to which the compiled subroutines and functions are written when the PROC FCMP step ends. This argument is required. The following are syntax examples:

```
proc fcmp outlib=sasuser.fcmpsubs.pkt1;
proc fcmp outlib=sasuser.mysubs.math;
```

Tip: Use this option when you want to save subroutines and functions in an output library.

Tip: Only those subroutines that are declared inside the current PROC FCMP step are saved to the output file. Those subroutines that are loaded by using the LIBRARY= option are not saved to the output file. If you do not specify the OUTLIB= option, then no subroutines that are declared in the current PROC FCMP step are saved.

PRINT
specifies printing the result of each statement in a program as it is executed. This option produces extensive output.

TRACE
specifies printing the results of each operation in each statement in a program as it is executed. These results are produced in addition to the information that is printed by the FLOW option. The TRACE option produces extensive output.

Tip: Specifying TRACE is equivalent to specifying FLOW, PRINT, and PRINTALL.

ABORT Statement

Terminates the current DATA step, job, or SAS session.

ABORT;

Without Arguments

The ABORT statement in PROC FCMP has no arguments.

ARRAY Statement

Associates a name with a list of variables and constants.

ARRAY *array-name*[*dimensions*] </NOSYMBOLS | *variable(s)* | *constant(s)* | (*initial-values*)>;

Arguments

array-name

specifies the name of the array.

dimensions
is a numeric representation of the number of elements in a one-dimensional array or the number of elements in each dimension of a multidimensional array.

Options

/NOSYMBOLS
specifies that an array of numeric or character values be created without the associated element variables. In this case, the only way you can access elements in the array is by array subscripting.

Tip: /NOSYMBOLS is used in exactly the same way as _TEMPORARY_.

Tip: You can save memory if you do not need to access the individual array element variables by name.

variable
specifies the variables of the array.

constant
specifies a number or a character string that indicates a fixed value. Enclose character constants in quotation marks.

initial-values
gives initial values for the corresponding elements in the array. You can specify internal values inside parentheses.

Details

The Basics

The ARRAY statement in PROC FCMP is similar to the ARRAY statement that is used in the DATA step. The ARRAY statement associates a name with a list of variables and constants. You use the array name with subscripts to refer to items in the array.

The ARRAY statement that is used in PROC FCMP does not support all the features of the ARRAY statement in the DATA step. The following is a list of differences that apply only to PROC FCMP:

- All array references must have explicit subscript expressions.
- PROC FCMP uses parentheses after a name to represent a function call. When you reference an array, use square brackets [ ]or curly braces { }.
- The ARRAY statement in PROC FCMP does not support lower-bound specifications.
- You can use a maximum of six dimensions for an array.

You can use both variables and constants as array elements in the ARRAY statement that is used in PROC FCMP. You cannot assign elements to a constant array. Although dimension specification and the list of elements are optional, you must provide one of these values. If you do not specify a list of elements for the array, or if you list fewer elements than the size of the array, PROC FCMP creates array variables by adding a numeric suffix to the elements of the array to complete the element list.

Passing Array References to PROC FCMP Routines

If you want to pass an array to a CALL routine and have the CALL routine modify the values of the array, you must specify the name for the array argument in an OUTARGS statement in the CALL routine.

Examples

The following are examples of the ARRAY statement:

- `array spot_rate[3] 1 2 3;`
- `array spot_rate[3] (1 2 3);`
- `array y[4] y1-y4;`
- `array xx[2,3] x11 x12 x13 x21 x22 x23;`
- `array pp p1-p12;`
- `array q[1000] /nosymbols;`

ATTRIB Statement

Specifies format, label, and length information for variables.

ATTRIB *variable(s)* <FORMAT=*format-name* LABEL=*'label'* LENGTH=*length*>;

Required Arguments

variable
specifies the variables that you want to associate with attributes.

Options

FORMAT=*format-name*
associates a format with variables in the *variable* argument.

LABEL=*'label'*
associates a label with variables in the *variable* argument.

LENGTH=*length*
specifies the length of the variable in the *variable* argument.

Examples

The following are examples of the ATTRIB statement:

- `attrib x1 format=date7. label='variable x1' length=5;`
- ```
  attrib x1 format=date7. label='variable x1' length=5
         x2 length=5
         x3 label='var x3' format=4.
         x4 length=$2 format=$4.;
  ```

DELETEFUNC | DELETESUBR Statement

Causes a function to be deleted from the function library that is specified in the OUTLIB option.

DELETEFUNC | DELETESUBR*function-name*;
  ```

Arguments

function-name
specifies the name of a function to be deleted from the function library that is specified in the OUTLIB option.

FUNCTION Statement

Specifies a subroutine declaration for a routine that returns a value.

FUNCTION *function-name(argument-1, ..., argument-n)* <VARARGS> <$> <*length*> <KIND | GROUP='*string*'>;
... *more-program-statements* ...
RETURN (*expression*);

ENDSUB;

Arguments

function-name
specifies the name of the function.

argument
specifies one or more arguments for the function. You specify character arguments by placing a dollar sign ($) after the argument name. In the following example,

```
function myfunct(arg1, arg2 $, arg3, arg4 $);
```

arg1 and arg3 are numeric arguments, and arg2 and arg4 are character arguments.

VARARGS
specifies that the function supports a variable number of arguments. If you specify VARARGS, then the last argument in the function must be an array.
See: "Using Variable Arguments with an Array" on page 428

$
specifies that the function returns a character value. If $ is not specified, the function returns a numeric value.

length
specifies the length of a character value.
Default: 8

KIND | GROUP='*string*'
specifies a collection of items that have specific attributes.

expression
specifies the value that is returned from the function.

Details

The FUNCTION statement is a special case of the subroutine declaration that returns a value. You do not use a CALL statement to call a function. The definition of a function begins with the FUNCTION statement and ends with an ENDSUB statement.

Examples

Using Numeric Data in the FUNCTION Statement

The following example uses numeric data as input to the FUNCTION statement of PROC FCMP:

```
proc fcmp;
   function inverse(in);
      if in=0 then inv=.;
      else inv=1/in;
      return(inv);
   endsub;
run;
```

Using Character Data in the FUNCTION Statement

The following example uses character data as input to the FUNCTION statement of PROC FCMP. The output from FUNCTION **test** is assigned a length of 12 bytes.

```
options cmplib = work.funcs;

proc fcmp outlib=work.funcs.math;
   function test(x $) $ 12;
   if x = 'yes' then
      return('si si si');
      else
      return('no');
   endsub;
run;

data _null_;
   spanish=test('yes');
   put spanish=;
run;
```

SAS writes the following output to the log:

```
spanish=si si si
```

Using Variable Arguments with an Array

The following example shows an array that accepts variable arguments. The example implies that the summation function can be called as follows:
sum = summation(1, 2, 3, 4, 5);.

```
options cmplib=sasuser.funcs;

proc fcmp outlib=sasuser.funcs.temp;
function summation (b[*]) varargs;
    total = 0;
    do i = 1 to dim(b);
        total = total + b[i];
    end;
return(total);
endsub;
sum=summation(1,2,3,4,5);
   put sum=;
run;
```

LABEL Statement

Specifies a label of up to 256 characters.

LABEL *variable='label'*;

Arguments

variable
names the variable that you want to label.

'label'
specifies a label of up to 256 characters, including blanks.

Examples

The following are examples of the LABEL statement:

- `label date='Maturity Date';`
- `label bignum='Very very large numeric value';`

LISTFUNC | LISTSUBR Statement

Causes the source code for a function to be printed in the SAS listing.

LISTFUNC | LISTSUBR*function-name*;

Arguments

function-name
specifies the name of the function for which source code is printed in the SAS listing.

STRUCT Statement

Declares (creates) structure types that are defined in C-Language packages.

STRUCT *structure-name variable*;

Arguments

structure-name
specifies the name of a structure that is defined in a C-language package and declared in PROC FCMP.

variable

specifies the variable that you want to declare as this structure type.

Examples

The following is an example of the STRUCT statement.

```
struct DATESTR matdate;
matdate.month = 3;
matdate.day = 22;
matdate.year = 2009;
```

SUBROUTINE Statement

Declares (creates) an independent computational block of code that you can call using a CALL statement.

SUBROUTINE *subroutine-name (argument-1, ..., argument-n)* <VARARGS> <KIND | GROUP='*string*'>;

OUTARGS *out-argument-1, ..., out-argument-n*;

... more-program-statements ...

ENDSUB;

Arguments

subroutine-name
specifies the name of a subroutine.

argument
specifies one or more arguments for the subroutine. Character arguments are specified by placing a dollar sign ($) after the argument name. In the following example,

```
subroutine mysub(arg1, arg2 $, arg3, arg4 $);
```

arg1 and arg3 are numeric arguments, and arg2 and arg4 are character arguments.

VARARGS
specifies that the subroutine supports a variable number of arguments. If you specify VARARGS, then the last argument in the subroutine must be an array.

OUTARGS
specifies arguments from the argument list that the subroutine should update.

KIND | GROUP='*string*'
specifies a collection of items that have specific attributes.

out-argument
specifies arguments from the argument list that you want the subroutine to update.

Details

The SUBROUTINE statement enables you to declare (create) an independent computational block of code that you can call with a CALL statement. The definition of

a subroutine begins with the SUBROUTINE statement and ends with an ENDSUB statement. You can use the OUTARGS statement in a SUBROUTINE statement to specify arguments from the argument list that the subroutine should update.

Examples

The following is an example of the SUBROUTINE statement:

```
proc fcmp outlib=sasuser.funcs.temp;
subroutine inverse(in, inv) group="generic";
   outargs inv;
   if in=0 then inv=.;
   else inv=1/in;
endsub;

options cmplib=sasuser.funcs;
data _null_;
   x = 5;
   call inverse(x, y);
   put x= y=;
run;
```

SAS writes the following output to the log:

```
x=5 y=0.2
```

OUTARGS Statement

Specifies arguments in an argument list that you want a subroutine to update.

Restriction: Use OUTARGS only with the SUBROUTINE statement.

OUTARGS *out-argument-1, ..., out-argument-n*;

Arguments

out-argument
specifies arguments from the argument list that you want the subroutine to update.

Tip: If an array is listed in the OUTARGS statement within a routine, then the array is passed "by reference." Otherwise, it is passed "by value."

See "SUBROUTINE Statement" on page 430 for an example of how to use the OUTARGS statement in a subroutine.

Concepts: FCMP Procedure

Creating Functions and Subroutines

PROC FCMP enables you to write functions and CALL routines using DATA step syntax. PROC FCMP functions and CALL routines are stored in a data set and can be called from several SAS/STAT, SAS/ETS, or SAS/OR procedures such as the NLIN, MODEL, and NLP procedures. You can create multiple functions and CALL routines in a single FCMP procedure step.

Functions are equivalent to routines that are used in other programming languages. They are independent computational blocks that require zero or more arguments. A subroutine is a special type of function that has no return value. All variables that are created within a function or subroutine block are local to that subroutine.

Creating Functions and Subroutines: An Example

This following example defines a function and a subroutine. The function begins with the FUNCTION statement, and the subroutine begins with the SUBROUTINE statement. The DAY_DATE function converts a date to a numeric day of the week, and the INVERSE subroutine calculates a simple inverse. Each ends with an ENDSUB statement.

```
proc fcmp outlib = sasuser.MySubs.MathFncs;

   function day_date(indate, type $);
      if type = "DAYS" then wkday = weekday(indate);
      if type = "YEARS" then wkday = weekday(indate*365);
      return(wkday);
   endsub;

   subroutine inverse(in, inv);
      outargs inv;
      if in = 0 then inv = .;
      else inv = 1/in;
   endsub;

run;
```

The function and subroutine follow DATA step syntax. Functions and subroutines that are already defined in the current FCMP procedure step, as well as most DATA step functions, can be called from within these routines as well. In the example above, the DATA step function WEEKDAY is called by DAY_DATE.

The routines in the example are saved to the data set *sasuser.MySubs*, inside a package called MathFncs. A package is any collection of related routines that are specified by the user. It is a way of grouping related subroutines and functions within the data set. The OUTLIB= option in the PROC FCMP statement tells PROC FCMP where to store the subroutines it compiles, and the LIBRARY= option tells it where to read in libraries (C or SAS).

Note: Function and subroutine names must be unique within a package. However, different packages can have subroutines and functions with the same names. To select a specific subroutine when there is ambiguity, use the package name and a period as the prefix to the subroutine name. For example, to access the MthFncs version of INVERSE, use MthFncs.inverse. △

Writing Your Own Functions

Advantages of Writing Your Own Functions and CALL Routines

PROC FCMP enables you to write functions and CALL routines by using DATA step syntax. The advantages of writing user-defined functions and CALL routines include the following:

- The function or CALL routine makes a program easier to read, write, and modify.
- The function or CALL routine is independent. A program that calls a routine is not affected by the routine's implementation.
- The function or CALL routine is reusable. Any program that has access to the data set where the function or routine is stored can call the routine.

Note: PROC FCMP routines that you create cannot have the same name as built-in SAS functions. If the names are the same, then SAS generates an error message stating that a built-in SAS function or subroutine already exists with the same name. △

Writing a User-Defined Function

The following program shows the syntax that is used to create and call a PROC FCMP function from a DATA step. This example computes the study day during a drug trial.

The example creates a function named STUDY_DAY in a package named TRIAL. A package is a collection of routines that have unique names and is stored in the data set sasuser.funcs. STUDY_DAY accepts two numeric arguments, *intervention_date* and *event_date*. The body of the routine uses DATA step syntax to compute the difference between the two dates, where days that occur before *intervention_date* begin at -1 and become smaller, and days that occur after and including *intervention_date* begin at 1 and become larger. This function never returns 0 for a study day.

STUDY_DAY is called from DATA step code as if it were any other function. When the DATA step encounters a call to STUDY_DAY, it will not find this function in its traditional library of functions. Instead, SAS searches each of the libraries or data sets that are specified in the CMPLIB system option for a package that contains STUDY_DAY. In this example, STUDY_DAY is located in *sasuser.funcs.trial*. The program calls the function, passing the variable values for *start* and *today*, and returns the result in the variable SD.

```
options pageno=1 nodate;

proc fcmp outlib=sasuser.funcs.trial;
   function study_day(intervention_date, event_date);
       n=event_date-intervention_date;
       if n <= 0 then
       n=n+1;
       return (n);
   endsub;

options cmplib=sasuser.funcs;
data _null_;
   start = '15Feb2006'd;
   today = '27Mar2006'd;
```

```
   sd = study_day(start, today);
   put sd=;
run;
```

Output 23.1 Output from the STUDY_DAY User-Defined Function

```
sd=41
```

Using Library Options

You can use PROC FCMP with the OUTLIB= or INLIB= options. The syntax for this procedure has the following form:

```
proc fcmp outlib=libname.dataset.package
          inlib=in-libraries;
routine-declarations;
```

The OUTLIB= option is required and specifies the package where routines declared in the *routine-declarations* section are stored.

Routines that are declared in the *routine-declarations* section can call FCMP routines that exist in other packages. To find these routines and to check the validity of the call, SAS searches the data sets that are specified in the INLIB= option. The format for the INLIB= option is as follows:

```
inlib=library.dataset
inlib=(library1.dataset1 library2.dataset2 ...  libraryN.datasetN)
inlib=library.datasetM - library.datasetN
```

If the routines that are being declared do not call FCMP routines in other packages, then you do not need to specify the INLIB= option.

Declaring Functions

You declare one or more functions or CALL routines in the *routine-declarations* section of the program. A routine consists of four parts:

- a name
- one or more parameters
- a body of code
- a RETURN statement

You specify these four parts between the FUNCTION or SUBROUTINE keyword and an ENDSUB keyword. For functions, the syntax has the following form:

```
function name(argument-1, ...  , argument-n);
   program-statements;
   return (expression);
endsub;
```

After the FUNCTION keyword, you specify the name of the function and its arguments. Arguments in the function declaration are called formal arguments and can be used within the body of the function. To specify a string argument, place a dollar sign ($) after the argument name. For functions, all arguments are passed by value. This means that the value of the actual argument, variable, or value that is passed to the function from the calling environment is copied before being used by the function. This copying ensures that any modification of the formal argument by the function does not change the original value.

The RETURN statement is used to return a value to a function. The RETURN statement accepts an expression that is enclosed in parentheses, and contains the value that is returned to the calling environment. The function declaration ends with an ENDSUB statement.

Declaring CALL Routines

CALL routines are declared within *routine-declarations* by using the SUBROUTINE keyword instead of the FUNCTION keyword. Functions and CALL routines have the same form, except CALL routines do not return a value, and CALL routines can modify their parameters. You specify the arguments to be modified on an OUTARGS statement. The syntax of a CALL routine declaration is as follows:

```
subroutine name(argument-1, ..., argument-n);
   outargs out-argument-1, ..., out-argument-N;
   program-statements;
   return;
endsub;
```

The formal arguments that are listed in the OUTARGS statement are passed by reference instead of by value. This means that any modification of the formal argument by the CALL routine will modify the original variable that was passed. It also means that the value is not copied when the CALL routine is invoked. Reducing the number of copies can improve performance when you pass large amounts of data between a CALL routine and the calling environment.

A RETURN statement is optional within the definition of the CALL routine. When a RETURN statement executes, execution is immediately returned to the caller. A RETURN statement within a CALL routine does not return a value.

Writing Program Statements

The *program-statements* section of the program is a series of DATA step statements that describe the work to be done by the function or CALL routine. Most DATA step statements and functions are accessible from PROC FCMP routines. The DATA step file and the data set I/O statements (for example, INPUT, FILE, SET, and MERGE) are not available from PROC FCMP routines. However, some functionality of the PUT statement is supported. See "PROC FCMP and DATA Step Differences" on page 435 for more information.

PROC FCMP and DATA Step Differences

Overview of PROC FCMP and DATA Step Differences

PROC FCMP was originally developed as a programming language for several SAS/STAT, SAS/ETS, and SAS/OR procedures. Because the implementation is not identical to the DATA step, differences exist between the two languages. The following section describes some of the differences between PROC FCMP and the DATA step.

Differences between PROC FCMP and the DATA Step

ABORT Statement

The ABORT statement in PROC FCMP does not accept arguments.

The ABORT statement is not valid within functions or subroutines in PROC FCMP. It is valid only in the main body of the procedure.

Arrays

PROC FCMP uses parentheses after a name to represent a function call. When referencing an array, the recommended practice is to use square brackets [ ]or curly braces { }. For an array named ARR, the code would be **`ARR[i]`** or **`ARR{i}`**. PROC FCMP limits the number of dimensions for an array to six.

For more information about the differences in the ARRAY statement for PROC FCMP, see "Details" on page 425.

Data Set Input and Output

PROC FCMP does not support the DATA and the OUTPUT statements for creating and writing to an output data set. It does not support the SET, MERGE, UPDATE, or MODIFY statements for data set input. Data is typically transferred into and out of PROC FCMP routines by using parameters. If a large amount of data needs to be transferred, you can pass arrays to a PROC FCMP routine.

DATA Step Debugger

When you use the DATA step debugger, PROC FCMP routines perform like any other routine. That is, it is not possible to step into the function when debugging. Instead, use a PUT statement within the routine.

DO Statement

The following type of DO statement is supported by PROC FCMP:

```
do i = 1, 2, 3;
```

The DO statement in PROC FCMP does not support character loop control variables. You can execute the following code in the DATA step, but not in PROC FCMP:

```
do i = 'a', 'b', 'c';
```

The DO statement does not support a character index variable; therefore, the following code is not supported in PROC FCMP:

```
do i = 'one','two','three';
```

File Input and Output

PROC FCMP supports the PUT and FILE statements, but the FILE statement is limited to LOG and PRINT destinations. There are no INFILE or INPUT statements in PROC FCMP.

IF Expressions

An IF expression enables IF/THEN/ELSE conditions to be evaluated within an expression. IF expressions are supported by PROC FCMP but not by the DATA step. You can simplify some expressions with IF expressions by not having to split the expression among IF/THEN/ELSE statements. For example, the following two pieces of code are equivalent, but the IF expression (the first example) is not as complex:

- `x = if y < 100 then 1 else 0;`
- ```
  if y < 100 then
     x=1;
  else
     x=0;
  ```

The alternative to IF expressions are expressions. This means that parentheses are used to group operations instead of DO/END blocks.

PUT Statement

The syntax of the PUT statement is similar in PROC FCMP and in the DATA step, but their operations can be different. In PROC FCMP, the PUT statement is typically used for program debugging. In the DATA step, the PUT statement is used as a report or file creation tool, as well as a debugging tool. The following list describes other differences:

- The PUT statement in PROC FCMP does not support line pointers, format modifiers, column output, factored lists, iteration factors, overprinting, the _INFILE_ option, or the special character $. It does not support features that are provided by the FILE statement options, such as DLM= and DSD.
- The PUT statement in PROC FCMP supports evaluating an expression and writing the result by placing the expression in parentheses. The DATA step, however, does not support the evaluation of expressions in a PUT statement. In the following example for PROC FCMP, the expressions **`x/100`** and **`sqrt(y)/2`** are evaluated and the results are written to the SAS log:

  ```
  put (x/100) (sqrt(y)/2);
  ```

  Because parentheses are used for expression evaluation in PROC FCMP, they cannot be used for variable or format lists as in the DATA step.
- The PUT statement in PROC FCMP does not support subscripted array names unless they are enclosed in parentheses. For example, the statement **`PUT (A[i]);`** writes the i-th element of the array A, but the statement **`PUT A[i];`** results in an error message.
- An array name can be used in a PUT statement without superscripts. Therefore, the following statements are valid:
  - **`PUT A=;`** (when A is an array), writes all of the elements of array A with each value labeled with the name of the element variable.
  - **`PUT (A)*=;`** writes the same output as **`PUT A=;`**.
  - **`PUT A;`** writes all of the elements of array A.
- The PUT statement in PROC FCMP follows the output of each item with a space, which is similar to list mode output in the DATA step. Detailed control over column and line position are supported to a lesser extent than in the DATA step.
- The PUT statement in PROC FCMP supports the print item _PDV_, and prints a formatted listing of all of the variables in the routine's program data vector. The statement **`PUT _PDV_;`** prints a much more readable listing of the variables than is printed by the statement **`PUT _ALL_;`**.

WHEN and OTHERWISE Statements

The WHEN and OTHERWISE statements allow more than one target statement. That is, DO/END groups are not necessary for multiple WHEN statements. The following is an example:
  ```

```
SELECT;
   WHEN(expression-1) statement-1;
   statement-2;
   WHEN (expression-2) statement-3;
   statement-4;
END;
```

Additional Features in PROC FCMP

PROC REPORT and Compute Blocks

PROC REPORT uses the DATA step to evaluate compute blocks. Because the DATA step can call PROC FCMP routines, you can also call these routines from PROC REPORT compute blocks.

The FCmp Function Editor

The FCmp Function Editor is an application for traversing packages of functions and is built into the SAS Explorer. You can access the FCmp Function Editor from the Solutions menu of an interactive SAS session. For more information, see "The FCmp Function Editor" on page 477.

Computing Implicit Values of a Function

PROC FCMP uses a SOLVE function for computing implicit values of a function. For more information, see "The SOLVE Function" on page 467.

PROC FCMP and Microsoft Excel

Many Microsoft Excel functions, not typically available in SAS, are implemented in PROC FCMP. You can find these functions in the sashelp.slkwxl data set.

Working with Arrays

Passing Arrays

By default, PROC FCMP passes arrays "by value" between routines. However, if an array is listed in the OUTARGS statement within the routine, the array is passed "by reference."

This means that a modification to the formal parameter by the function modifies the array that is passed. Passing arrays by reference helps to efficiently pass large amounts of data between the function and the calling environment because the data does not need to be copied. The syntax for specifying a formal array has the following form:

```
function name(numeric-array-parameter[*], character-array-parameter[*] $);
```

You can pass DATA step temporary arrays to PROC FCMP routines.

Resizing Arrays

You can resize arrays in PROC FCMP routines by calling the built-in CALL routine DYNAMIC_ARRAY. The syntax for this CALL routine has the following form:

```
call dynamic_array(array, new-dim1-size, ..., new-dimN-size);
```

SAS passes to the DYNAMIC_ARRAY CALL routine both the array that is to be resized and a new size for each dimension of the array. A dynamic array enables the routine to allocate the amount of memory that is needed, instead of having to create an array that is large enough to handle all possible cases.

Support for dynamic arrays is limited to PROC FCMP routines. When an array is resized, the array is available only in the routine that resized it. It is not possible to resize a DATA step array or to return a PROC FCMP dynamic array to a DATA step.

Reading Arrays and Writing Arrays to a Data Set

Overview

PROC FCMP provides the READ_ARRAY function to read arrays, and the WRITE_ARRAY function to write arrays to a data set. This functionality enables PROC FCMP array data to be processed by SAS programs, macros, and procedures.

The READ_ARRAY Function

Syntax of the READ_ARRAY Function

The READ_ARRAY function reads data from a SAS data set into a PROC FCMP array variable.

The following two forms of the READ_ARRAY function are available:

rc = **READ_ARRAY** *(data_set_name, array_variable*);

rc = **READ_ARRAY** *(data_set_name, array_variable <, 'col_name_1', 'col_name_2',* ...>);

where

rc
is 0 if the function is able to successfully read the data set.

data_set_name
specifies the name of the data set from which the array data will be read. *data_set_name* must be a character literal or variable that contains the member name (libname.memname) of the data set to be read from.

array_variable
specifies the PROC FCMP array variable into which the data is read. *array_variable* must be a local temporary array variable because the function might need to grow or shrink its size to accommodate the size of the data set.

col_name
specifies optional names for the specific columns of the data set that will be read. If specified, *col_name* must be a literal string enclosed in quotation marks. *col_name* cannot be a PROC FCMP variable. If column names are not specified, PROC FCMP reads all of the columns in the data set.

Details

When SAS translates between an array and a data set, the array will be indexed as [row,column].

Arrays that are declared in functions and CALL routines can be resized, as well as arrays that are declared with the /NOSYMBOLS option. No other arrays can be resized.

The READ_ARRAY function attempts to dynamically resize the array to match the dimensions of the input data set. This means that the array must be dynamic. That is, the array must be declared either in a function or CALL routine or declared with the /NOSYMBOLS option.

Example of the READ_ARRAY Function

This example creates and reads a SAS data set into an FCMP array variable.

```
data account;
   input acct price cost;
   datalines;
1 2 3
4 5 6
;
run;

proc fcmp;
   array x[2,3] / nosymbols;
   rc = read_array('account', x);
   put x=;
run;

proc fcmp;
   array x[2,2] / nosymbols;
   rc = read_array('account', x, 'price', 'acct');
   put x=;
run;
```

Output 23.2 Output from the READ_ARRAY Function

```
                                The SAS System                                1

                              The FCMP Procedure

x[1, 1]=1 x[1, 2]=2 x[1, 3]=3 x[2, 1]=4 x[2, 2]=5 x[2, 3]=6
```

```
                                The SAS System                                2

                              The FCMP Procedure

x[1, 1]=2 x[1, 2]=1 x[2, 1]=5 x[2, 2]=4
```

The WRITE_ARRAY Function

Syntax of the WRITE_ARRAY Function

The WRITE_ARRAY function writes data from a PROC FCMP array variable to a data set that can then be used by SAS programs, macros, and procedures. When SAS translates between an array and a data set, the array will be indexed as [row, column].

The following two forms of the WRITE_ARRAY function are available:

rc = **WRITE_ARRAY** *(data_set_name, array_variable)*;

rc = **WRITE_ARRAY***(data_set_name, array_variable* <, *'col_name_1', 'col_name_2',* ...>);

where

rc
is 0 if the function is able to successfully write the data set.

data_set_name
specifies the name of the data set to which the array data will be written. *data_set_name* must be a character literal or variable that contains the member name (libname.memname) of the data set to be created.

array_variable
specifies the PROC FCMP array or matrix variable whose contents will be written to *data_set_name*.

col_name
specifies optional names for the columns of the data set that will be created.
If specified, *col_name* must be a literal string enclosed in quotation marks. *col_name* cannot be a PROC FCMP variable. If column names are not specified, the column name will be the array name with a numeric suffix.

Example 1: Using the WRITE_ARRAY Function with a PROC FCMP Array Variable

This example uses the ARRAY statement and the WRITE_ARRAY function with PROC FCMP to write output to a data set.

```
options nodate pageno=1 ls=80 ps=64;

proc fcmp;
   array x[4,5] (11 12 13 14 15 21 22 23 24 25 31 32 33 34 35 41 42 43 44 45);
   rc = write_array('work.numbers', x);
run;

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

Output 23.3 Output from Using the WRITE_ARRAY Function

```
                          The SAS System                                    1
                  Obs    x1    x2    x3    x4    x5

                    1    11    12    13    14    15
                    2    21    22    23    24    25
                    3    31    32    33    34    35
                    4    41    42    43    44    45
```

Example 2: Using the WRITE_ARRAY Function to Specify Column Names

This example uses the optional *colN* variable to write column names to the data set.

```
options pageno=1 nodate ps=64 ls=80;

proc fcmp;
   array x[2,3] (1 2 3 4 5 6);
   rc = write_array('numbers2', x, 'col1', 'col2', 'col3');
run;

proc print data = numbers2;
run;
```

Output 23.4 Output from Using the WRITE_ARRAY Function to Specify Column Names

```
                            The SAS System                          1

                          Obs    col1    col2    col3

                           1       1       2       3
                           2       4       5       6
```

Using Macros with PROC FCMP Routines

You can use the %SYSFUNC and the %SYSCALL macros to call routines that you create with PROC FCMP. All SAS CALL routines are accessible with %SYSCALL except LABEL, VNAME, SYMPUT, and EXECUTE.

Variable Scope in PROC FCMP Routines

The Concept of Variable Scope

A critical part of keeping routines and programs independent of one another is variable scope. A variable's *scope* is the section of code where a variable's value can be

used. In the case of PROC FCMP routines, variables that are declared outside a routine are not accessible inside a routine. Variables that are declared inside a routine is not accessible outside the routine. Variables that are created within a routine are called local variables because their scope is "local" to the routine.

Functions use local variables as scratch variables during computations, and the variables are not available when the function returns. When a function is called, space for local variables is pushed on the call stack. When the function returns, the space used by local variables is removed from the call stack.

When Local Variables in Different Routines Have the Same Name

The concept of variable scope can be confusing when local variables in different routines have the same name. When this occurs, each local variable is distinct. In the following example, the DATA step and CALL routines **subA** and **subB** contain a local variable named x. Each x is distinct from the other x variables. When the program executes, the DATA step calls **subA** and **subA** calls **subB**. Each environment writes the value of x to the log. The log output shows how each x is distinct from the others.

```
proc fcmp outlib=sasuser.funcs.math;
   subroutine subA();
      x=5;
      call subB();
      put 'In subA: ' x=;
   endsub;

   subroutine subB();
      x='subB';
      put 'In subB: ' x=;
   endsub;
run;

options cmplib=sasuser.funcs;
data _null_;
   x=99;
   call subA();
   put 'In DATA step: ' x=;
run;
```

SAS writes the following output to the log:

Output 23.5 Output from Local Variables in Different Routines Having the Same Name

```
In subB:  x=subB
In subA:  x=5
In DATA step: x=99
```

Recursion

PROC FCMP routines can be recursive. Recursion is a problem-solving technique that reduces a problem to a smaller one that is simpler to solve and then combines the

results of the simpler solution to form a complete solution. A recursive function is a function that calls itself, either directly or indirectly.

Each time a routine is called, space for the local variables is pushed on the call stack. The space on the call stack ensures independence of local variables for each call. When the routine returns, the space allocated on the call stack is removed, freeing the space used by local variables. Recursion relies on the call stack to store progress toward a complete solution.

When a routine calls itself, both the calling routine and the routine that is being called must have their own set of local variables for intermediate results. If the calling routine was able to modify the local variables of the routine that is being called, it would be difficult to program a recursive solution. A call stack ensures the independence of local variables for each call.

In the following example, the ALLPERMK routine in PROC FCMP has two arguments, n and k, and writes all $P(n, k) = n! / (n - k)!$ permutations that contain exactly k out of the n elements. The elements are represented as binary values (0, 1). The function ALLPERMK calls the recursive function PERMK to traverse the entire solution space and output only the items that match a particular filter.

```
proc fcmp outlib=sasuser.funcs.math;
   subroutine allpermk(n, k);
      array scratch[1] / nosymbols;
      call dynamic_array(scratch, n);
      call permk(n, k, scratch, 1, 0);
   endsub;

subroutine permk(n, k, scratch[*], m, i);
   outargs scratch;
   if m--1 = n then do;
      if i = k then
         put scratch[*];
      end;
   else do;
      scratch[m] = 1;
      call permk(n, k, scratch, m+1, i+1);
      scratch[m] = 0;
      call permk(n, k, scratch, m+1, i);
   end;
endsub;
run;
quit;

options cmplib=sasuser.funcs;
data _null_;
   call allpermk(5, 3);
run;
```

Output 23.6 Log Output from Recursion Example

```
1 1 1 0 0
1 1 0 1 0
1 1 0 0 1
1 0 1 1 0
1 0 1 0 1
1 0 0 1 1
0 1 1 1 0
0 1 1 0 1
0 1 0 1 1
0 0 1 1 1
```

This program uses the /NOSYMBOLS option in the ARRAY statement to create an array without a variable for each array element. A /NOSYMBOLS array can be accessed only with an array reference, **scratch[*m*]**, and is equivalent to a DATA step _temporary_ array. A /NOSYMBOLS array uses less memory than a regular array because no space is allocated for variables. ALLPERMK also uses PROC FCMP dynamic arrays.

Directory Transversal

Overview of Directory Transversal

Implementing functionality that enables functions to traverse a directory hierarchy is difficult if you use the DATA step or macros. With the DATA step and macro code recursion or pseudo-recursion is not easy to code. This section describes how to develop a routine named DIR_ENTRIES that fills an array with the full pathname of all of the files in a directory hierarchy. This example shows the similarity between PROC FCMP and DATA step syntax and underscores how PROC FCMP routines simplify a program and produce independent, reusable code. DIR_ENTRIES uses as input the following parameters:

- a starting directory
- a result array to fill with pathnames
- an output parameter that is the number of pathnames placed in the result array
- an output parameter that indicates whether the complete result set was truncated because the result array was not large enough

The flow of control for DIR_ENTRIES is as follows:

1. Open the starting directory.
2. For each entry in the directory, do one of the following tasks:
   - If the entry is a directory, call DIR_ENTRIES to fill the result array with the subdirectory's pathnames.
   - Otherwise, the entry is a file, and you need to add the file's path to the result array.
3. Close the starting directory.

Directory Transversal Example

Opening and Closing a Directory

Opening and closing a directory are handled by the CALL routines DIROPEN and DIRCLOSE. DIROPEN accepts a directory path and has the following flow of control:

1. Create a fileref for the path by using the FILENAME function.
2. If the FILENAME function fails, write an error message to the log and then return.
3. Otherwise, use the DOPEN function to open the directory and retrieve a directory ID.
4. Clear the directory fileref.
5. Return the directory ID.

The DIRCLOSE CALL routine is passed a directory ID, which is passed to DCLOSE. DIRCLOSE sets the passed directory ID to missing so that an error occurs if a program tries to use the directory ID after the directory has been closed. The following code implements the DIROPEN and DIRCLOSE CALL routines:

```
proc fcmp outlib=sasuser.funcs.dir;
   function diropen(dir $);
   length dir $ 256 fref $ 8;
   rc = filename(fref, dir);
   if rc = 0 then do;
      did = dopen(fref);
      rc = filename(fref);
   end;
   else do;
      msg = sysmsg();
      put msg '(DIROPEN(' dir= ')';
      did = .;
   end;
   return(did);
endsub;

subroutine dirclose(did);
   outargs did;
   rc = dclose(did);
   did = .;
endsub;
```

Gathering Filenames

File paths are collected by the DIR_ENTRIES CALL routine. DIR_ENTRIES uses the following arguments:

- a starting directory
- a result array to fill
- an output parameter to fill with the number of entries in the result array
- an output parameter to set to 0 if all pathnames fit in the result array; or an output parameter to set to 1 if some of the pathnames do not fit into the array

The body of DIR_ENTRIES is almost identical to the code that is used to implement this functionality in a DATA step; yet DIR_ENTRIES is a CALL routine that is easily reused in several programs.

DIR_ENTRIES calls DIROPEN to open a directory and retrieve a directory ID. The routine then calls DNUM to retrieve the number of entries in the directory. For each entry in the directory, DREAD is called to retrieve the name of the entry. Now that the

entry name is available, the routine calls MOPEN to determine whether the entry is a file or a directory.

If the entry is a file, then MOPEN returns a positive value. In this case, the full path to the file is added to the result array. If the result array is full, the truncation output argument is set to 1.

If the entry is a directory, then MOPEN returns a value that is less than or equal to 0. In this case, DIR_ENTRIES gathers the pathnames for the entries in this subdirectory. It gathers the pathnames by recursively calling DIR_ENTRIES and passing the subdirectory's path as the starting path. When DIR_ENTRIES returns, the result array contains the paths of the subdirectory's entries.

```
subroutine dir_entries(dir $, files[*] $, n, trunc);
   outargs files, n, trunc;
   length dir entry $ 256;

   if trunc then return;

   did = diropen(dir);
   if did <= 0 then return;

   dnum = dnum(did);
   do i = 1 to dnum;
      entry = dread(did, i);
      /* If this entry is a file, then add to array */
      /* Else entry is a directory, recurse. */
      fid = mopen(did, entry);
      entry = trim(dir) || '\' || entry;
      if fid > 0 then do;
         rc = fclose(fid);
         if n < dim(files) then do;
            trunc = 0;
            n = n + 1;
            files[n] = entry;
         end;
         else do;
            trunc = 1;
            call dirclose(did);
            return;
         end;
      end;
      else
         call dir_entries(entry, files, n, trunc);
   end;

   call dirclose(did);
   return;
endsub;
```

Calling DIR_ENTRIES from a DATA Step

You invoke DIR_ENTRIES like any other DATA step CALL routine. Declare an array with enough entries to hold all the files that might be found. Then call the DIR_ENTRIES routine. When the routine returns, the result array is looped over and each entry in the array is written to the SAS log.

```
options cmplib=sasuser.funcs;
data _null_;
   array files[1000] $ 256 _temporary_;
   dnum = 0;
   trunc = 0;
   call dir_entries("c:\logs", files, dnum, trunc);
   if trunc then put 'ERROR: Not enough result array entries. Increase array size.';
   do i = 1 to dnum;
      put files[i];
   end;
run;
```

Output 23.7 Output from Calling DIR_ENTRIES from a DATA Step

```
c:\logs\2004\qtr1.log
c:\logs\2004\qtr2.log
c:\logs\2004\qtr3.log
c:\logs\2004\qtr4.log
c:\logs\2005\qtr1.log
c:\logs\2005\qtr2.log
c:\logs\2005\qtr3.log
c:\logs\2005\qtr4.log
c:\logs\2006\qtr1.log
c:\logs\2006\qtr2.log
```

This example shows the similarity between PROC FCMP syntax and the DATA step. For example, numeric expressions and flow of control statements are identical. The abstraction of DIROPEN into a PROC FCMP function simplifies DIR_ENTRIES. All of the PROC FCMP routines that are created can be reused by other DATA steps without any need to modify the routines to work in a new context.

Identifying the Location of Compiled Functions and Subroutines: The CMPLIB= System Option

Overview of the CMPLIB= System Option

The SAS system option CMPLIB= specifies where to look for previously compiled functions and subroutines. All procedures (including FCMP) that support the use of FCMP functions and subroutines use this system option.

Instead of specifying the LIBRARY= option on every procedure statement that supports functions and subroutines, you can use the CMPLIB= system option to set libraries that can be used by all procedures.

Syntax of the CMPLIB= System Option

The syntax for the CMPLIB= option has the following form:

OPTIONS CMPLIB = *library*

OPTIONS CMPLIB = *(library–1, ..., library-n)*

OPTIONS CMPLIB = *list-1, ..., list-n*

where

OPTIONS
identifies the statement as an OPTIONS statement.

library
specifies that the previously compiled libraries be linked into the program.

list
specifies a list of libraries.

Example 1: Setting the CMPLIB= System Option

The following example shows how to set the CMPLIB= system option.

```
□ OPTIONS cmplib = sasuser.funcs;

□ OPTIONS cmplib = (sasuser.funcs work.functions mycat.funcs);

□ OPTIONS cmplib =( sasuser.func1 - sasuser.func10);
```

Example 2: Compiling and Using Functions

In the following example, PROC FCMP compiles the SIMPLE function and stores it in the **sasuser.models** data set. Then the CMPLIB= system option is set, and the function is called by PROC MODEL.

```
proc fcmp outlib = sasuser.models.yval;
   function simple(a,b,x);
      y=a+b*x;
      return(y);
   endsub;
run;

options cmplib = sasuser.models nodate ls=80;

data a;
   input y @@;
   x=_n_;
   datalines;
08 06 08 10 08 10
;

proc model data=a;
   y=simple(a,b,x);
   fit y / outest=est1 out=out1;
quit;
```

Output 23.8 Output from Using the SIMPLE Function with PROC MODEL

```
                                The SAS System                                1

                             The MODEL Procedure

                                Model Summary

                          Model Variables         1
                          Parameters              2
                          Equations               1
                          Number of Statements    1


                            Model Variables  y
                                 Parameters  a b
                                  Equations  y


                        The Equation to Estimate is

                                y =  F(a, b)


      NOTE: At OLS Iteration 1 CONVERGE=0.001 Criteria Met.
```

```
                                The SAS System                                2

                             The MODEL Procedure
                           OLS Estimation Summary

                                Data Set Options

                                DATA=      A
                                OUT=       OUT1
                                OUTEST=    EST1


                              Minimization Summary

                     Parameters Estimated            2
                     Method                      Gauss
                     Iterations                      1


                        Final Convergence Criteria

                        R                        0
                        PPC                      0
                        RPC(a)               64685.48
                        Object               0.984333
                        Trace(S)              1.67619
                        Objective Value       1.11746


                           Observations Processed

                                Read      6
                                Solved    6
```

```
                                The SAS System                                3

                              The MODEL Procedure

                     Nonlinear OLS Summary of Residual Errors

                  DF       DF                                                 Adj
Equation        Model    Error         SSE         MSE    Root MSE    R-Square       R-Sq

y                   2        4      6.7048      1.6762      1.2947      0.4084     0.2605


                      Nonlinear OLS Parameter Estimates

                                    Approx                    Approx
            Parameter       Estimate     Std Err    t Value     Pr > |t|

            a               6.533333      1.2053       5.42       0.0056
            b               0.514286      0.3095       1.66       0.1719


                 Number of Observations     Statistics for System

                 Used                6    Objective         1.1175
                 Missing             0    Objective*N       6.7048
```

For information about PROC MODELS, see *SAS/ETS User's Guide*.

Special Functions and CALL Routines: Overview

The FCMP procedure provides a small set of special use functions. You can call these functions from user-defined FCMP functions but you cannot call these functions directly from the DATA step. To use these functions in a DATA step, you must wrap the special function inside another user-defined FCMP function.

Note: You can call special functions directly in a procedure, but not in the DATA step. △

Special Functions and CALL Routines: Matrix CALL Routines

CALL Routines and Matrix Operations

The FCMP procedure provides you with a number of CALL routines for performing simple matrix operations on declared arrays. These CALL routines are automatically provided by the FCMP procedure. With the exception of ZEROMATRIX, FILLMATRIX, and IDENTITY, the CALL routines listed below do not support matrices or arrays that contain missing values.

Function or CALL routine	Description
“ADDMATRIX CALL Routine” on page 452	performs an element-wise addition of two matrices or a matrix and a scalar.
“CHOL CALL Routine” on page 453	(CHOLESKY_DECOMP CALL routine) calculates the Cholesky decomposition for a given symmetric matrix.
“DET CALL Routine” on page 454	calculates the determinant of a specified matrix that should be square.
“ELEMMULT CALL Routine” on page 455	performs an element-wise multiplication of two matrices.
“EXPMATRIX CALL Routine” on page 456	returns a matrix e^{tA} given the input matrix A and a multiplier t. The CALL routine uses a Padé approximation algorithm.
“FILLMATRIX CALL Routine” on page 457	replaces all of the element values of the input matrix with the specified value. You can use the FILLMATRIX CALL routine with multidimensional numeric arrays.
“IDENTITY CALL Routine” on page 457	converts the input matrix to an identity matrix. Diagonal element values of the matrix will be set to 1, and the rest of the values will be set to 0.
“INV CALL Routine” on page 458	calculates a matrix that is the inverse of the provided input matrix that should be a square, non-singular matrix.
“MULT CALL Routine” on page 459	calculates the multiplicative product of two input matrices.
“POWER CALL Routine” on page 460	raises a square matrix to a given scalar value.
“SUBTRACTMATRIX CALL Routine” on page 461	performs an element-wide subtraction of two matrices or a matrix and a scalar.
“TRANSPOSE CALL Routine” on page 462	returns the transpose of a matrix.
“ZEROMATRIX CALL Routine” on page 462	replaces all of the element values of the numeric input matrix with 0.

ADDMATRIX CALL Routine

The ADDMATRIX CALL routine performs an element-wise addition of two matrices or a matrix and a scalar.

The syntax of the ADDMATRIX CALL routine has the following form:

CALL ADDMATRIX (*X*, *Y*, *Z*)

where

X

specifies an input matrix with dimensions m x n (that is, X[m, n]) or a scalar.

Y

specifies an input matrix with dimensions m x n (that is, Y[m, n]) or a scalar.

Z

specifies an output matrix with dimensions m x n (that is, Z[m, n]).

such that

$$Z = X + Y$$

Note that all input and output matrices need to have the same dimensions.
The following example uses the ADDMATRIX CALL routine:

```
options pageno=1 nodate;

proc fcmp;
   array mat1[3,2] (0.3, -0.78, -0.82, 0.54, 1.74, 1.2);
   array mat2[3,2] (0.2, 0.38, -0.12, 0.98, 2, 5.2);
   array result[3,2];
   call addmatrix (mat1, mat2, result);
   call addmatrix (2, mat1, result);
   put result=;
 quit;
```

Output 23.9 Output from the ADDMATRIX CALL Routine

```
                                The SAS System                                1

                              The FCMP Procedure

result[1, 1]=2.3 result[1, 2]=1.22 result[2, 1]=1.18 result[2, 2]=2.54
result[3, 1]=3.74 result[3, 2]=3.2
```

CHOL CALL Routine

The CHOL CALL routine (CHOLESKY_DECOMP CALL routine) calculates the Cholesky decomposition for a given symmetric matrix.

The syntax of the CHOL CALL routine has the following form:

CALL CHOL (*X*, *Y* <, *validate*>)

where

X

specifies a symmetric positive-definite input matrix with dimensions m x m (that is, X[m, m]).

Y

specifies an output matrix with dimensions m x m (that is, Y[m, m]). This variable contains the Cholesky decomposition.

validate

specifies an optional argument which can increase the processing speed by avoiding error checking.

If *validate* = 0 or is not specified, then the matrix X will be checked for symmetry.

If *validate* = 1, then the matrix is assumed to be symmetric.

such that

$$Z = YY^*$$

where Y is a lower triangular matrix with strictly positive diagonal entries and Y* denotes the conjugate transpose of Y.

Note that both input and output matrices need to be square and have the same dimensions. X must be symmetric positive-definite, and Y will be a lower triangle matrix.

If X is not symmetric positive-definite, Y will be filled with missing values.

The following example uses the CHOL CALL routine:

```
proc fcmp;
   array xx[3,3] 2 2 3 2 4 2 3 2 6;
   array yy[3,3];
   call chol(xx, yy, 0);
   do i = 1 to 3;
      put yy[i, 1] yy[i, 2] yy[i, 3];
   end;
run;
```

SAS produces the following output:

Output 23.10 Output from PROC FCMP and the CHOL CALL Routine

```
                                The SAS System                          1

                              The FCMP Procedure

1.4142135624 0 0
1.4142135624 1.4142135624 0
2.1213203436 -0.707106781 1
```

DET CALL Routine

The DET CALL routine calculates the determinant of a specified matrix that should be square.

The determinant, the product of the eigenvalues, is a single numeric value. If the determinant of a matrix is zero, then that matrix is singular; that is, it does not have an inverse. The method performs an LU decomposition and collects the product of the diagonals (Forsythe, Malcolm, and Moler 1967). See the *SAS/IML User's Guide* for more information.

The syntax of the DET CALL routine has the following form:

CALL DET (*X*, *a*)

where

X

specifies an input matrix with dimensions *m* x *n* (that is, X[*m*, *m*]).

a

specifies the returned determinate value.

such that

$$a = |X|$$

Note that the input matrix X needs to be square.
The following example uses the DET CALL routine:

```
options pageno=1 nodate;

proc fcmp;
   array mat1[3,3] (.03, -0.78, -0.82, 0.54, 1.74,
                    1.2, -1.3, 0.25, 1.49);
   call det (mat1, result);
   put result=;
quit;
```

Output 23.11 Output from the DET CALL Routine

```
                                The SAS System                          1

                              The FCMP Procedure

result=-0.052374
```

ELEMMULT CALL Routine

The ELEMMULT CALL routine performs an element-wise multiplication of two matrices.
The syntax of the ELEMMULT CALL routine has the following form:

CALL ELEMMULT (*X*, *Y*, *Z*)

where

X
specifies an input matrix with dimensions *m* x *n* (that is, X[*m*, *n*]).

Y
specifies an input matrix with dimensions *m* x *n* (that is, Y[*m*, *n*]).

Z
specifies an output matrix with dimensions *m* x *n* (that is, Z[*m*, *n*]).

Note that all input and output matrices need to have the same dimensions.
The following example uses the ELEMMULT CALL routine:

```
options pageno=1 nodate;

proc fcmp;
   array mat1[3,2] (0.3, -0.78, -0.82, 0.54, 1.74, 1.2);
   array mat2[3,2] (0.2, 0.38, -0.12, 0.98, 2, 5.2);
   array result[3,2];
   call elemmult (mat1, mat2, result);
```

```
   call elemmult (2.5, mat1, result);
   put result=;
quit;
```

Output 23.12 Output from the ELEMMULT CALL Routine

```
                              The SAS System                              1

                            The FCMP Procedure

result[1, 1]=0.75 result[1, 2]=-1.95 result[2, 1]=-2.05 result[2, 2]=1.35
result[3, 1]=4.35 result[3, 2]=3
```

EXPMATRIX CALL Routine

The EXPMATRIX CALL routine returns a matrix e^{tA} given the input matrix A and a multiplier t. The CALL routine uses a Padé approximation algorithm as presented in Golub and van Loan (1989), p. 558. Note that this module does not exponentiate each entry of a matrix. Refer to the EXPMATRIX documentation in the *SAS/IML User's Guide* for more information.

The syntax of the EXPMATRIX CALL routine has the following form:

CALL EXPMATRIX (*X*, *t*, *Y*)

where

X
specifies an input matrix with dimensions *m* x *m* (that is, X[*m*, *m*]).

t
specifies a double scalar value.

Y
specifies an output matrix with dimensions *m* x *m* (that is, Y[*m*, *m*]).

such that

$$Y = e^{tX}$$

Note that both input and output matrices need to be square and have the same dimensions. *t* can be any scalar value.

The following example uses the EXPMATRIX CALL routine:

```
options pageno=1 nodate;

proc fcmp;
   array mat1[3,3] (0.3, -0.78, -0.82, 0.54, 1.74,
                    1.2, -1.3, 0.25, 1.49);
   array result[3,3];
   call expmatrix (mat1, 3, result);
   put result=;
quit;
```

Output 23.13 Output from the EXPMATRIX CALL Routine

```
                                The SAS System                                1

                              The FCMP Procedure

result[1, 1]=365.58043585 result[1, 2]=-589.6358476 result[1, 3]=-897.1034008
result[2, 1]=-507.0874798 result[2, 2]=838.64570481 result[2, 3]=1267.3598426
result[3, 1]=-551.588816 result[3, 2]=858.97629382 result[3, 3]=1324.8187125
```

FILLMATRIX CALL Routine

The FILLMATRIX CALL routine replaces all of the element values of the input matrix with the specified value. You can use the FILLMATRIX CALL routine with multidimensional numeric arrays.

The syntax of the FILLMATRIX CALL routine has the following form:

CALL FILLMATRIX (*X*, *Y*)

where

X
specifies an input numeric matrix.

Y
specifies the numeric value that will fill the matrix.

The following example shows how to use the FILLMATRIX CALL routine.

```
options pageno=1 nodate ls=80 ps=64;

proc fcmp;
   array mat1[3, 2] (0.3, -0.78, -0.82, 0.54, 1.74, 1.2);
   call fillmatrix(mat1, 99);
   put mat1=;
quit;
```

Output 23.14 Output from the FILLMATRIX CALL Routine

```
                                The SAS System                                1

                              The FCMP Procedure

mat1[1, 1]=99 mat1[1, 2]=99 mat1[2, 1]=99 mat1[2, 2]=99 mat1[3, 1]=99
mat1[3, 2]=99
```

IDENTITY CALL Routine

The IDENTITY CALL routine converts the input matrix to an identity matrix. Diagonal element values of the matrix will be set to 1, and the rest of the values will be set to 0.

The syntax of the IDENTITY CALL routine has the following form:

CALL IDENTITY (*X*)

where

X
specifies an input matrix with dimensions *m* x *m* (that is, X[*m*, *m*]).

Note that the input matrix needs to be square.
The following example uses the IDENTITY CALL routine:

```
options pageno=1 nodate;

proc fcmp;
   array mat1[3,3] (0.3, -0.78, -0.82, 0.54, 1.74, 1.2,
                    -1.3, 0.25, 1.49);
   call identity (mat1);
   put mat1=;
quit;
```

Output 23.15 Output from the IDENTITY CALL Routine

```
                                The SAS System                                1

                             The FCMP Procedure

mat1[1, 1]=1 mat1[1, 2]=0 mat1[1, 3]=0 mat1[2, 1]=0 mat1[2, 2]=1 mat1[2, 3]=0
mat1[3, 1]=0 mat1[3, 2]=0 mat1[3, 3]=1
```

INV CALL Routine

The INV CALL routine calculates a matrix that is the inverse of the provided input matrix that should be a square, non-singular matrix.
The syntax of the INV CALL routine has the following form:

CALL INV (*X*, *Y*)

where

X
specifies an input matrix with dimensions *m* x *m* (that is, X[*m*, *m*]).

Y
specifies an output matrix with dimensions *m* x *m* (that is, Y[*m*, *m*]).

such that

$$Y\,[m, m] = X'\,[m, m]$$

where ’denotes inverse

$$X \times Y = Y \times X = I$$

and I is the identity matrix.
Note that both the input and output matrices need to be square and have the same dimensions.

The following example uses the INV CALL routine:

```
options pageno=1 nodate;

proc fcmp;
   array mat1[3,3] (0.3, -0.78, -0.82, 0.54, 1.74,
                    1.2, -1.3, 0.25, 1.49);
   array result[3,3];
   call inv(mat1, result);
   put result=;
quit;
```

Output 23.16 Output from the INV CALL Routine

```
                                The SAS System                                1

                              The FCMP Procedure

result[1, 1]=4.0460407887 result[1, 2]=1.6892917399 result[1, 3]=0.8661767509
result[2, 1]=-4.173108283 result[2, 2]=-1.092427483 result[2, 3]=-1.416802558
result[3, 1]=4.230288655 result[3, 2]=1.6571719011 result[3, 3]=1.6645841716
```

MULT CALL Routine

The MULT CALL routine calculates the multiplicative product of two input matrices. The syntax of the MULT CALL routine has the following form:

CALL MULT (*X*, *Y*, *Z*)

where

X
specifies an input matrix with dimensions *m* x *n* (that is, X[*m*, *n*]).

Y
specifies an input matrix with dimensions *n* x *p* (that is, Y[*n*, *p*]).

Z
specifies an output matrix with dimensions *m* x *p* (that is, Z[*m*, *p*]).

such that

$$Z\left[m,p\right]=X\left[m,n\right]\times Y\left[n,p\right]$$

Note that the number of columns for the first input matrix needs to be the same as the number of rows for the second matrix. The calculated matrix is the last argument.

The following example uses the MULT CALL routine:

```
options pageno=1 nodate;

proc fcmp;
   array mat1[2,3] (0.3, -0.78, -0.82, 0.54, 1.74, 1.2);
   array mat2[3,2] (1, 0, 0, 1, 1, 0);
   array result[2,2];
   call mult(mat1, mat2, result);
```

```
   put result=;
quit;
```

Output 23.17 Output from the MULT CALL Routine

```
                                The SAS System                                1

                              The FCMP Procedure

result[1, 1]=-0.52 result[1, 2]=-0.78 result[2, 1]=1.74 result[2, 2]=1.74
```

POWER CALL Routine

The POWER CALL routine raises a square matrix to a given scalar value. Large scalar values should be avoided because the POWER CALL routine's internal use of the matrix multiplication routine might cause numerical precision problems. If the scalar is not an integer, it is truncated to an integer. If the scalar is less than 0, then it is changed to 0. See the *SAS/IML User's Guide* for more information.

The syntax of the POWER CALL routine has the following form:

CALL POWER (*X*, *a*, *Y*)

where

X
specifies an input matrix with dimensions *m* x *m* (that is, X[*m*, *m*]).

a
specifies an integer scalar value (power).

Y
specifies an output matrix with dimensions *m* x *m* (that is, Y[*m*, *m*]).

such that

$$Y = X^a$$

Note that both input and output matrices need to be square and have the same dimensions.

The following example uses the POWER CALL routine:

```
options pageno=1 nodate;

proc fcmp;
   array mat1[3,3] (0.3, -0.78, -0.82, 0.54, 1.74,
                    1.2, -1.3, 0.25, 1.49);
   array result[3,3];
   call power (mat1, 3, result);
   put result=;
quit;
```

Output 23.18 Output from the POWER CALL Routine

```
                                The SAS System                                1

                              The FCMP Procedure

result[1, 1]=2.375432 result[1, 2]=-4.299482 result[1, 3]=-6.339638
result[2, 1]=-3.031224 result[2, 2]=6.272988 result[2, 3]=8.979036
result[3, 1]=-4.33592 result[3, 2]=5.775695 result[3, 3]=9.326529
```

SUBTRACTMATRIX CALL Routine

The SUBTRACTMATRIX CALL routine performs an element-wide subtraction of two matrices or a matrix and a scalar.

The syntax of the SUBTRACTMATRIX CALL routine has the following form:

CALL SUBTRACTMATRIX (*X*, *Y*, *Z*)

where

X
specifies an input matrix with dimensions *m* x *n* (that is, X[*m*, *n*]) or a scalar.

Y
specifies an input matrix with dimensions *m* x *n* (that is, Y[*m*, *n*]) or a scalar.

Z
specifies an output matrix with dimensions *m* x *n* (that is, Z[*m*, *n*]).

such that

$$Z = X - Y$$

Note that all input and output matrices need to have the same dimensions.

The following example uses the SUBTRACTMATRIX CALL routine:

```
options pageno=1 nodate;

proc fcmp;
   array mat1[3,2] (0.3, -0.78, -0.82, 0.54, 1.74, 1.2);
   array mat2[3,2] (0.2, 0.38, -0.12, 0.98, 2, 5.2);
   array result[3,2];
   call subtractmatrix (mat1, mat2, result);
   call subtractmatrix (2, mat1, result);
   put result=;
 quit;
```

Output 23.19 Output from the SUBTRACTMATRIX CALL Routine

```
                                The SAS System                                1

                              The FCMP Procedure

result[1, 1]=1.7 result[1, 2]=2.78 result[2, 1]=2.82 result[2, 2]=1.46
result[3, 1]=0.26 result[3, 2]=0.8
```

TRANSPOSE CALL Routine

The TRANSPOSE CALL routine returns the transpose of a matrix.
The syntax of the TRANSPOSE CALL routine has the following form:

CALL TRANSPOSE (*X*, *Y*)

where

X
specifies an input matrix with dimensions *m* x *n* (that is, X[*m*, *n*]).

Y
specifies an output matrix with dimensions *n* x *m* (that is, Y[*n*, *m*])

such that

$$Y = X'$$

Note that the number of rows for the input matrix should be equal to the number of columns of the output matrix, and the number of rows for the output matrix should be equal to the number of columns of the input matrix.
The following example uses the TRANSPOSE CALL routine:

```
options pageno=1 nodate;

proc fcmp;
   array mat1[3,2] (0.3, -0.78, -0.82, 0.54, 1.74, 1.2);
   array result[2,3];
   call transpose (mat1, result);
   put result=;
quit;
```

Output 23.20 Output from the TRANSPOSE CALL Routine

```
                                The SAS System                                1

                              The FCMP Procedure

result[1, 1]=0.3 result[1, 2]=-0.82 result[1, 3]=1.74 result[2, 1]=-0.78
result[2, 2]=0.54 result[2, 3]=1.2
```

ZEROMATRIX CALL Routine

The ZEROMATRIX CALL routine replaces all of the element values of the numeric input matrix with 0. You can use the ZEROMATRIX CALL routine with multi-dimensional numeric arrays.
The syntax of the ZEROMATRIX CALL routine has the following form:

CALL ZEROMATRIX (*X*)

where

X
specifies a numeric input matrix.

The following example uses the ZEROMATRIX CALL routine:

```
options pageno=1 nodate;

proc fcmp;
   array mat1[3,2]  (0.3, -0.78, -0.82, 0.54, 1.74, 1.2);
   call zeromatrix (mat1);
   put mat1=;
quit;
```

Output 23.21 Output from the ZEROMATRIX CALL Routine

```
                                The SAS System                                1

                              The FCMP Procedure

mat1[1, 1]=0 mat1[1, 2]=0 mat1[2, 1]=0 mat1[2, 2]=0 mat1[3, 1]=0 mat1[3, 2]=0
```

Special Functions and CALL Routines: C Helper Functions and CALL Routines

C Helper Functions and CALL Routines

Several helper functions are provided with the package to handle C-language constructs in PROC FCMP. Most C-language constructs must be defined in a package that is created by PROC PROTO before the constructs can be referenced or used by PROC FCMP. The ISNULL function and the SETNULL and STRUCTINDEX CALL routines have been added to extend the SAS language to handle C-language constructs that do not naturally fit into the SAS language.

ISNULL C Helper Function

Overview of the ISNULL C Helper Function

The ISNULL function determines whether a pointer element of a structure is null.

Syntax of the ISNULL C Helper Function

The syntax of the ISNULL function has the following form:

numeric-variable **ISNULL** (*pointer-element*);

where

numeric-variable
specifies a numeric value.

pointer-element
specifies a variable that contains the address of another variable.

Example 1: Generating a Linked List

In the following example, the LINKLIST structure and GET_LIST function are defined by using PROC PROTO. The GET_LIST function is an external C routine that generates a linked list with as many elements as requested.

```
struct linklist{
   double value;
   struct linklist * next;
};

struct linklist * get_list(int);
```

Example 2: Using the ISNULL C Helper Function in a Loop

The following code segment shows that the ISNULL C helper function loops over the linked list that is created by GET_LIST and writes out the elements.

```
proc proto package=sasuser.mylib.str2;
struct linklist{
   double value;
   struct linklist * next;
};

struct linklist * get_list(int);

externc get_list;
   struct linklist * get_list(int len){
      int i;
      struct linklist * list=0;
      list=(struct linklist*)
         malloc(len*sizeof(struct linklist));
      for (i=0;i<len-1;i++){
         list[i].value=i;
         list[i].next=&list[i+1];
      }
      list[i].value=i;
      list[i].next=0;
      return list;
   }
externcend;
run;

options pageno=1 nodate ls=80 ps=64;
proc fcmp libname=sasuser.mylib;
   struct linklist list;
   list=get_list(3);
   put list.value=;

   do while (^isnull(list.next));
      list = list.next;
      put list.value=;
   end;
run;
```

Output 23.22 Results from Using the ISNULL C Helper Function

```
                                The SAS System                                1

                              The FCMP Procedure

list.value=0
list.value=1
list.value=2
```

SETNULL C Helper CALL Routine

Overview of the SETNULL C Helper CALL Routine

The SETNULL CALL routine sets a pointer element of a structure to null.

Syntax of the SETNULL C Helper CALL Routine

The syntax of the SETNULL C Helper CALL routine has the following form:

CALL SETNULL(*pointer-element*);

where *pointer-element* is a pointer to a structure.

Example: Setting an Element in a Linklist to Null

The following example assumes that the same LINKLIST structure that is described in "Example 1: Generating a Linked List" on page 464 is defined using PROC PROTO. The CALL SETNULL routine can be used to set the NEXT element to null:

```
struct linklist list;
call setnull(list.next);
```

STRUCTINDEX C Helper CALL Routine

Overview of the STRUCTINDEX C Helper CALL Routine

The STRUCTINDEX CALL routine enables you to access each structure element in an array of structures.

Syntax of the STRUCTINDEX C Helper CALL Routine

The syntax of the STRUCTINDEX routine has the following form:

CALL STRUCTINDEX(*struct_array*, *index*, *struct_element*);

where

struct_array
specifies an array.

index

is a 1–based index as used in most SAS arrays.

struct_element
points to an element in the array.

Example: Setting Point Structures in an Array

In the first part of this example, the following structures and function are defined by using PROC PROTO.

```
proc proto package=sasuser.mylib.str2;
struct point{
   short s;
   int i;
   long l;
   double d;
};

struct point_array {
   int            length;
   struct point p[2];
   char           name[32];
};
run;
```

In the second part of this example, the PROC FCMP code segment shows how to use the STRUCTINDEX CALL routine to retrieve and set each point structure element of an array called P in the POINT_ARRAY structure:

```
options pageno=1 nodate ls=80 ps=64;

proc fcmp libname=sasuser.mylib;
   struct point_array pntarray;
   struct point pnt;

   pntarray.length = 2;
   pntarray.name = "My funny structure";

   /* Get each element using the STRUCTINDEX CALL routine and set values. */
do i = 1 to 2;
   call structindex(pntarray.p, i, pnt);
   put "Before setting the" i "element: " pnt=;
   pnt.s = 1;
   pnt.i = 2;
   pnt.l = 3;
   pnt.d = 4.5;
   put "After setting the" i "element: " pnt=;
end;
run;
```

Output 23.23 Results of Setting the Point Structure Elements of an Array

```
                                The SAS System                                1

                              The FCMP Procedure

Before setting the 1 element:  pnt {s=0, i=0, l=0, d=0}
After setting the 1 element:  pnt {s=1, i=2, l=3, d=4.5}
Before setting the 2 element:  pnt {s=0, i=0, l=0, d=0}
After setting the 2 element:  pnt {s=1, i=2, l=3, d=4.5}
```

Special Functions and CALL Routines: Other Functions

The SOLVE Function

Overview of the SOLVE Function

The SOLVE function computes implicit values of a function. It is a special purpose function that is automatically provided by the FCMP procedure for convenience.

Syntax of the SOLVE Function

The syntax of the SOLVE function has the following form:

answer = **SOLVE** ('*function-name*', *options-array*, *expected-value*, *argument-1, argument-2, ..., argument-n*);

where

answer
specifies the value that is returned from the SOLVE function.

'*function-name*'
specifies the name of the function. Enclose *function-name* in quotation marks.

options-array
specifies an array of options to use with the SOLVE function. *Options-array* is used to control and monitor the root-finding process. *Options-array* can be a missing value (.), or it can have up to five of the following elements in the following order:

initial-value
specifies the starting value for the implied value. The default for the first call is 0.001. If the same line of code is executed again, then *options-array* uses the previously found implied value.

absolute-criterion
specifies a value for convergence. The absolute value of the difference between the expected value and the predicted value must be less than the value of *absolute-criterion* for convergence.

Default: 1.0e–12

relative-criterion

specifies a value for convergence. When the change in the computed implied value is less than the value of *relative-criterion*, then convergence is assumed.

Default: 1.0e–6

maximum-iterations
specifies the maximum number of iterations to use to find the solution.

Default: 100

solve-status
can be one of the following values:

0	successful.
1	could not decrease the error.
2	could not compute a change vector.
3	maximum number of iterations exceeded.
4	initial objective function is missing.

expected-value
specifies the expected value of the function of interest.

argument
specifies the arguments to pass to the function that is being minimized.

Details of the SOLVE Function

The SOLVE function finds the value of the specified argument that makes the expression of the following form equal to zero.

```
expected-value - function-name (argument-1,argument-2, ..., argument-n)
```

You specify the argument of interest with a missing value (.), which appears in place of the argument in the parameter list that is shown above. If the SOLVE function finds the value, then the value that is returned for this function is the implied value.

The following is an example of an options array:

```
array opts[5] initial abconv relconv maxiter (.5 .001 1.0e-6 100);
```

where

- *initial* (initial-value) = .5
- *abconv* (absolute-criterion) = .001
- *relconv* (relative-criterion) = 1.0e-6
- *maxiter* (maximum-iterations) = 100

The solve status is the fifth element in the array. You can display this value by specifying opts[5] in the output list.

Example 1: Computing a Square Root Value

The following SOLVE function example computes a value of x that satisfies the equation y=1/sqrt(x). Note that you must first define functions and subroutines before you can use them in the SOLVE function. In this example, the function INVERSESQRT is first defined and then used in the SOLVE function.

```
options pageno=1 nodate ls=80 ps=64;

proc fcmp;
      /* define the function */
```

```
   function inversesqrt(x);
      return(1/sqrt(x));
   endsub;

   y = 20;
   x = solve("inversesqrt", {.}, y, .);
   put x;
run;
```

Output 23.24 Results from Computing a Square Root Value

```
                                The SAS System                                1

                              The FCMP Procedure

0.0025
```

Example 2: Calculating the Garman-Kohlhagen Implied Volatility

In this example, the subroutine GKIMPVOL calculates the Garman-Kohlhagen implied volatility for FX options by using the SOLVE function with the GARKHPRC function.

In this example, note the following:

- The options_array is SOLVOPTS, which requires an initial value.
- The expected value is the price of the FX option.
- The missing argument in the subroutine is the volatility (sigma).

```
proc fcmp;
   function garkhprc(type$, buysell$, amount, E, t, S, rd, rf, sig)
      kind=pricing label='FX option pricing';

   if buysell='Buy' then sign=1.;
   else do;
      if buysell='Sell' then sign=-1.;
      else sign=.;
   end;

   if type='Call' then
      garkhprc=sign*amount*(E+t+S+rd+rf+sig);
   else do;
      if type='Put' then
         garkhprc=sign*amount*(E+t+S+rd+rf+sig);
      else garkhprc=.;
   end;
   return(garkhprc);
   endsub;

   subroutine gkimpvol(n, premium[*], typeflag[*], amt_lc[*],
                       strike[*], matdate[*], valudate, xrate,
                       rd, rf, sigma);
   outargs sigma;
```

```
   array solvopts[1] initial (0.20);
   sigma = 0;
   do i = 1 to n;
      maturity = (matdate[i] - valudate) / 365.25;
      stk_opt = 1./strike[i];
      amt_opt = amt_lc[i] * strike[i];
      price = premium[i] * amt_lc[i];

      if typeflag[i] eq 0 then type = "Call";
      if typeflag[i] eq 1 then type = "Put";

         /* solve for volatility */
      sigma = sigma + solve("GARKHPRC", solvopts, price,
                            type, "Buy", amt_opt, stk_opt,
                            maturity, xrate, rd, rf, .);
   end;
   sigma = sigma / n;
endsub;
run;
```

Example 3: Calculating the Black-Scholes Implied Volatility

This SOLVE function example defines the function BLKSCH by using the built-in SAS function BLKSHCLPRC. The SOLVE function uses the BLKSCH function to calculate the Black-Scholes implied volatility of an option.

In this example, note the following:

- The options_array is OPTS.
- The missing argument in the function is the volatility (VOLTY).
- PUT statements are used to write the implied volatility (BSVOLTY), the initial value, and the solve status.

```
options pageno=1 nodate ls=80 ps=64;

proc fcmp;
   opt_price = 5;
   strike = 50;
   exp = '01jul2001'd;
   eq_price = 50;
   intrate = .05;
   time = exp - date();
   array opts[5] initial abconv relconv maxiter
                 (.5 .001 1.0e-6 100);
   function blksch(strike, time, eq_price, intrate, volty);
      return(blkshclprc(strike, time/365.25,
                        eq_price, intrate, volty));
   endsub;
   bsvolty = solve("blksch", opts, opt_price, strike,
                             time, eq_price, intrate, .);

   put 'Option Implied Volatility:' bsvolty
       'Initial value: ' opts[1]
       'Solve status: ' opts[5];
run;
```

Output 23.25 Results of Calculating the Black-Scholes Implied Volatility

```
                                The SAS System                                1

                              The FCMP Procedure

Option Implied Volatility: . Initial value:  0.5 Solve status:  2
```

Note: SAS functions and external C functions cannot be used directly in the SOLVE function. They must be enclosed in a PROC FCMP function. In this example, the built-in SAS function BLKSHCLPRC is enclosed in the PROC FCMP function BLKSCH, and then BLKSCH is called in the SOLVE function. △

The DYNAMIC_ARRAY Subroutine

Overview of the DYNAMIC_ARRAY Subroutine

The DYNAMIC_ARRAY subroutine enables an array that is declared within a function to change size in an efficient manner.

Syntax of the DYNAMIC_ARRAY Subroutine

The syntax of the DYNAMIC_ARRAY subroutine has the following form:

CALL DYNAMIC_ARRAY(*array–name*, *new-dim1–size, ..., new-dimN-size*);

where

array-name
specifies the name of a temporary array.

new-dim-size
specifies a new size for the temporary array.

/NOSYMBOLS
specifies that an array of numeric or character values be created without the associated element variables. In this case, the only way you can access elements in the array is by array subscripting.

Tip: You can save memory if you do not need to access the individual array element variables by name.

The DYNAMIC_ARRAY CALL routine is passed the array to be resized and a new size for each dimension of the array. In the ALLPERMK routine, a scratch array that is the size of the number of elements being permuted is needed. When the function is created, this value is not known because it is passed in as parameter n. A dynamic array enables the routine to allocate the amount of memory that is needed, instead of having to create an array that is large enough to handle all possible cases.

When using dynamic arrays, support is limited to PROC FCMP routines. When an array is resized, the resized array is available only within the routine that resized it. It is not possible to resize a DATA step array or to return a PROC FCMP dynamic array to the DATA step.

Details

Arrays that are declared in functions and CALL routines can be resized, as well as arrays that are declared with the /NOSYMBOLS option. No other array can be resized.

The DYNAMIC_ARRAY CALL routine attempts to dynamically resize the array to match the dimensions of the target that you provide. This means that the array must be dynamic. That is, the array must be declared either in a function or subroutine, or declared with the /NOSYMBOLS option.

Example: Creating a Temporary Array

The following example creates a temporary array named TEMP. The size of the array area depends on parameters that are passed to the function.

```
proc fcmp;
   function avedev_wacky(data[*]);

   length = dim(data);
   array temp[1] /nosymbols;
   call dynamic_array(temp, length);

   mean=0;
   do i=1 to datalen;
      mean += data[i];
      if i>1 then temp[i]=data[i-1];
      else temp[i]=0;
   end;
   mean=mean/length;

   avedev=0;
   do i=1 to length;
      avedev += abs((data[i])-temp[i] /2-mean);
   end;
   avedev=avedev/datalen;

   return(avedev);
endsub;
run;
```

Functions for Calling SAS Code from Within Functions

The RUN_MACRO Function

Syntax of the RUN_MACRO Function

The RUN_MACRO function executes a predefined SAS macro. Its behavior is similar to executing **`%macro_name;`** in SAS.

The following two forms of the RUN_MACRO function are available:

rc = **RUN_MACRO** *('macro_name')*

rc = **RUN_MACRO** *('macro_name', variable_1, variable_2, ...)*

where

rc
is 0 if the function is able to submit the macro. The return code indicates only that the macro call was attempted. The macro itself should set the value of a SAS macro variable that corresponds to a PROC FCMP variable to determine whether the macro executed as expected.

macro_name
specifies the name of the macro to be run.

Requirement: *macro_name* must be a string enclosed in quotation marks or a character variable that contains the macro to be executed.

variable
specifies optional PROC FCMP variables which are set by macro variables of the same name. These arguments must be PROC FCMP double or character variables.

Before SAS executes the macro, SAS macro variables are defined with the same name and value as the PROC FCMP variables. After SAS executes the macro, the macro variable values are copied back to the corresponding PROC FCMP variables.

Example 1: Executing a Predefined Macro with PROC FCMP

This example creates a macro called TESTMACRO, and then uses the macro within PROC FCMP to subtract two numbers.

```
      /* Create a macro called TESTMACRO. */
%macro testmacro;
   %let p = %sysevalf(&a - &b);
%mend testmacro;

    /* Use TESTMACRO within a function in PROC FCMP to subtract two numbers. */
proc fcmp outlib = sasuser.ds.functions;
   function subtract_macro(a, b);
      rc = run_macro('testmacro', a, b, p);
      if rc eq 0 then return(p);
      else return(.);
   endsub;
run;

   /* Make a call from the DATA step. */
option cmplib = (sasuser.ds);

data _null_;
   a = 5.3;
   b = 0.7;
   p = .;
   p = subtract_macro(a, b);
   put p=;
run;
```

Output 23.26 Output from Executing a Predefined Macro with PROC FCMP

```
p=4.6
```

Example 2: Executing a DATA Step within a DATA Step

This example shows how to execute a DATA step from within another DATA step. The program consists of the following sections:

- The first section of the program creates a macro called APPEND_DS. This macro can write to a data set or append a data set to another data set.
- The second section of the program calls the macro from function writeDataset in PROC FCMP.
- The third section of the program creates the SALARIES data set and divides the data set into four separate data sets depending on the value of the variable Department.
- The fourth section of the program writes the results to the output window.

```
   /* Create a macro called APPEND_DS. */
%macro append_ds;
   /* Character values that are passed to RUN_MACRO are put              */
   /* into their corresponding macro variables inside of quotation           */
   /* marks. The quotation marks are part of the macro variable value.   */
   /* The DEQUOTE function is called to remove the quotation marks. */
   %let dsname = %sysfunc(dequote(&dsname));
   data &dsname
      %if %sysfunc(exist(&dsname)) %then %do;
         modify &dsname;
      %end;
      Name = &Name;
      WageCategory = &WageCategory;
      WageRate = &WageRate;
      output;
      stop;
   run;
%mend append_ds;

   /* Call the APPEND_DS macro from function writeDataset in PROC FCMP. */
proc fcmp outlib = sasuser.ds.functions;
   function writeDataset (DsName $, Name $, WageCategory $, WageRate);
      rc = run_macro('append_ds', dsname, DsName, Name, WageCategory, WageRate);
      return(rc);
   endsub;
run;

   /* Use the DATA step to separate the salaries data set into four separate */
   /* departmental data sets (NAD, DDG, PPD, and STD).                          */
data salaries;
   input Department $ Name $ WageCategory $ WageRate;
   datalines;
BAD Carol Salaried 20000
BAD Beth Salaried 5000
BAD Linda Salaried 7000
BAD Thomas Salaried 9000
BAD Lynne Hourly 230
DDG Jason Hourly 200
DDG Paul Salaried 4000
PPD Kevin Salaried 5500
```

```
PPD Amber Hourly 150
PPD Tina Salaried 13000
STD Helen Hourly 200
STD Jim Salaried 8000
;
run;

options cmplib = (sasuser.ds) pageno=1 nodate;
data _null_;
   set salaries;
   by Department;
  length dsName $ 64;
   retain dsName;
   if first.Department then do;
      dsName = 'work.' || trim(left(Department));
   end;
   rc = writeDataset(dsName, Name, WageCategory, wageRate);
run;

proc print data = work.BAD; run;
proc print data = work.DDG; run;
proc print data = work.PPD; run;
proc print data = work.STD; run;
```

Output 23.27 Output for Calling a DATA Step within a DATA Step

```
                    The SAS System                              1

                              Wage       Wage
            Obs     Name      Category     Rate

              1     Carol     Salaried    20000
              2     Beth      Salaried     5000
              3     Linda     Salaried     7000
              4     Thomas    Salaried     9000
              5     Lynne     Hourly        230
```

```
                    The SAS System                              2

                              Wage       Wage
             Obs    Name      Category    Rate

               1    Jason     Hourly       200
               2    Paul      Salaried    4000
```

```
                    The SAS System                              3

                              Wage       Wage
             Obs    Name      Category     Rate

               1    Kevin     Salaried     5500
               2    Amber     Hourly        150
               3    Tina      Salaried    13000
```

```
                     The SAS System                              4

                          Wage      Wage
             Obs    Name     Category    Rate

              1     Helen    Hourly       200
              2     Jim      Salaried    8000
```

The RUN_SASFILE Function

The Syntax of the RUN_SASFILE Function

The RUN_SASFILE function executes the SAS code in the fileref that you specify. The following two forms of the RUN_SASFILE function are available:

rc = **RUN_SASFILE** *('fileref_name');*

rc = **RUN_SASFILE***('fileref_name', variable-1, variable-2, ...)*

where

rc
: is 0 if the function is able to submit a request to execute the code that processes the SAS file. The return code indicates only that the call was attempted.

fileref_name
: specifies the name of the SAS fileref that points to the SAS code.

  Requirement: *fileref_name* must be a string enclosed in quotation marks or a character variable that contains the name of the SAS fileref.

variable
: specifies optional PROC FCMP variables which will be set by macro variables of the same name. These arguments must be PROC FCMP double or character variables.

  Before SAS executes the code that references the SAS file, the SAS macro variables are defined with the same name and value as the PROC FCMP variables. After execution, these macro variable values are copied back to the corresponding PROC FCMP variables.

Example

The following example is similar to the first example for RUN_MACRO except that RUN_SASFILE uses a SAS file instead of a predefined macro. This example assumes that **`test.sas(a, b, c)`** is located in the current directory.

```
      /* test.sas(a,b,c) */
data _null_;
   call symput('p', &a * &b);
run;

   /* Set a SAS fileref to point to the data set. */
filename myfileref "test.sas";

   /* Set up a function in PROC FCMP and call it from the DATA step. */
```

```
proc fcmp outlib = sasuser.ds.functions;
   function subtract_sasfile(a,b);
   rc = run_sasfile('myfileref', a, b, p);
   if rc = 0 then return(p);
      else return(.);
   endsub;
run;

options cmplib = (sasuser.ds);
data _null_;
   a = 5.3;
   b = 0.7;
   p = .;
   p = subtract_sasfile(a, b);
   put p=;
run;
```

The FCmp Function Editor

Introduction to the FCmp Function Editor

SAS language functions and CALL routines that are created with PROC FCMP are stored in SAS data sets that are contained in package declarations. Each package declaration contains one or more functions or CALL routines. The FCmp Function Editor displays all of the functions and CALL routines that are included in a package.

With the FCmp Function Editor, you can view functions in a package declaration as well as create new functions. You can add these new functions to an existing package, or create a new package declaration.

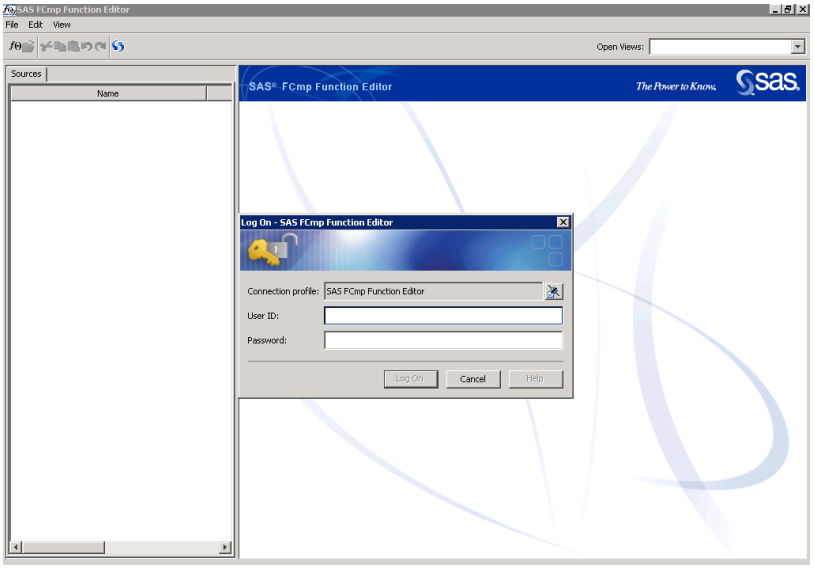
Open the FCmp Function Editor

If you are working in the Windows operating environment and SAS is installed locally on your computer, the sign-on dialog box is bypassed because Windows supports single sign-on functionality.

If you are not working in the Windows operating environment, or if you do not have SAS installed locally, then you will be prompted for your authorization credentials, which are your user ID and password.

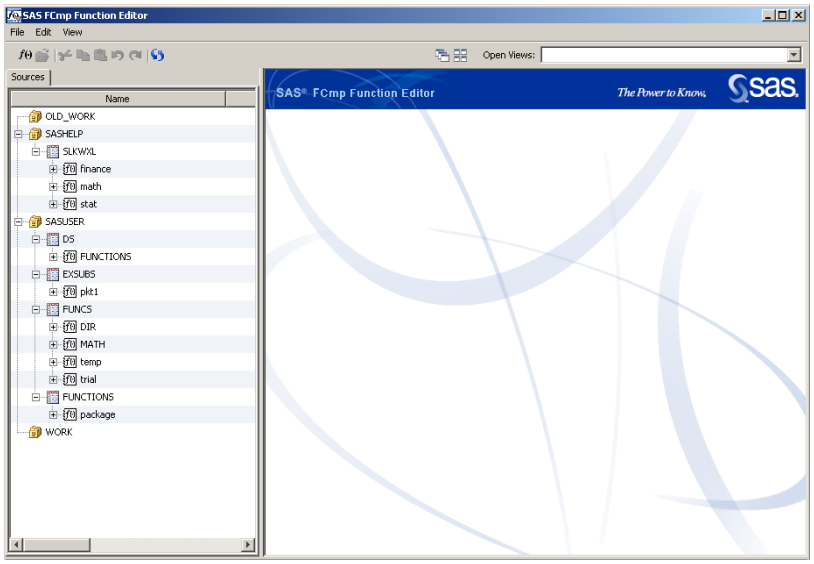
To open the FCmp Function Editor, select **Solutions ▸ Analysis ▸ FCmp Function Editor** from the menu in your SAS session. The following dialog box appears:

Display 23.1 Initial Dialog Box for the FCmp Function Editor



After you enter your user ID and password and click **`Log On`**, SAS establishes a connection to a port. The following window then appears:

Display 23.2 The FCmp Function Editor with Libraries Displayed



In the window above, you can see that the left pane lists the functions that are in the SASHELP and SASUSER libraries. The WORK library is empty. You cannot access the WORK library directly from a spawning SAS session. The FCmp Function Editor remaps the WORK library from the spawning SAS session to the location of OLD_WORK so that you can access the contents of WORK from OLD_WORK.

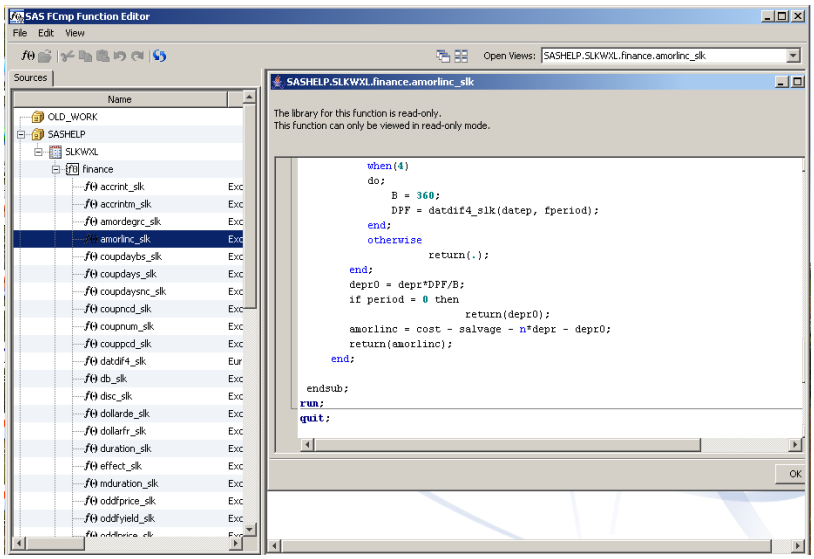
Working with Existing Functions

Open a Function

To open a function, select a library from the left pane, expand the library, and drill down until a list of functions appears. Double-click the name of the function you want to open.

If you open a function from a read-only library, a window similar to the following appears:

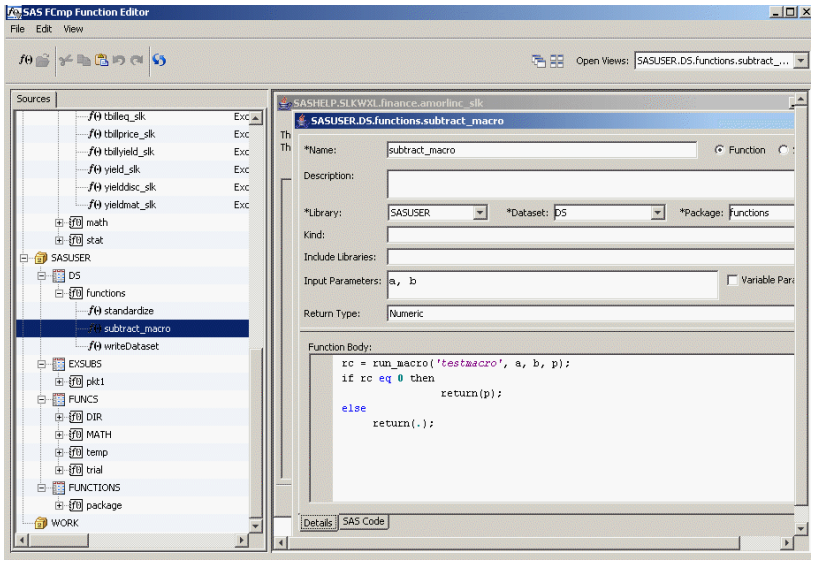
Display 23.3 A Function in a Library That Has Read-Only Access



In the window above, the ARMORLINK_SLK function is selected from the read-only SASHELP library. Use the scrollbar to scroll to the top of the function.

If you open a function from a library to which you have write access, a window similar to the following appears:

Display 23.4 A Function in a Library That Has Write Access



In the window above, SUBTRACT_MACRO is selected from the write-enabled SASUSER library.

Use the scrollbar to scroll to the top of the function.

You can see that there is a difference in the windows that display depending on whether the library has read-only access or write access. If the library has write access, you can enter information in the top section of the window you are viewing. These fields are the same fields you use when you create a new function. For a description of the fields, see "Creating a New Function" on page 483.

If you do not change the function name, the function is moved from its original position in the hierarchy to the library, data set, or function package that you designate.

Opening Multiple Functions

Opening multiple functions results in multiple windows being opened. For example, if you open a second function, a second window displays that shows the code for that function.

The upper right corner of the FCmp Function Editor window contains a field called **Open Views**. Click the arrow to list the functions that are open. When you select a function, the window for that function is brought to the foreground.

Two icons that you can use to alter the display of your functions are located at the left of the **Open Views** field:

cascades the display of the functions that are open.

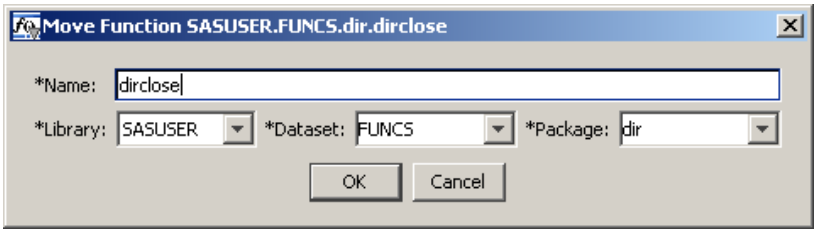
arranges the functions to display side by side.

Move a Function

You can move a function to a different library, data set, or package by first opening a function in a library that has write access. Then enter information for the fields that display at the top section of the window.

You can also highlight a function, right-click, and select **Move** from the menu. The following dialog box appears:

Display 23.5 The Move Dialog Box



In the Move dialog box, you can do the following:

- enter a new name for the function
- select a library into which the function is to be moved
- enter a new data set name
- enter a Package name

The descriptions of the fields in the Move Dialog box are as follows:

Name specifies the new name for the function.

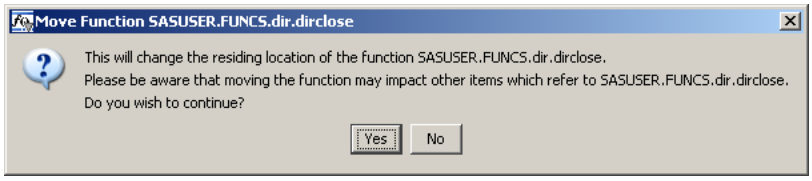
Library specifies the library that will contain the function that you copy. Use the menu in the **Library** field to select a library.

Dataset specifies the data set that will contain the function that you copy. Enter the name of the data set, or click the down arrow in the **Dataset** field to select a data set. If you do not choose a data set, then the value in this field defaults to Functions.

Package specifies the name of the package that will contain the new function that you copy. Enter the name of the data set, or click the down arrow in the **Package** field to select a data set. If you do not choose a data set, then the value in this field defaults to Package.

When you click **OK**, the following dialog box appears, cautioning you about the move:

Display 23.6 The Move Function Confirmation Dialog Box



CAUTION:
Other functions and macros that reference the function you want to move will not be updated with the new function location. This situation can cause referencing objects such as macros to be out of synchronization. △

Click **Yes** or **No**.

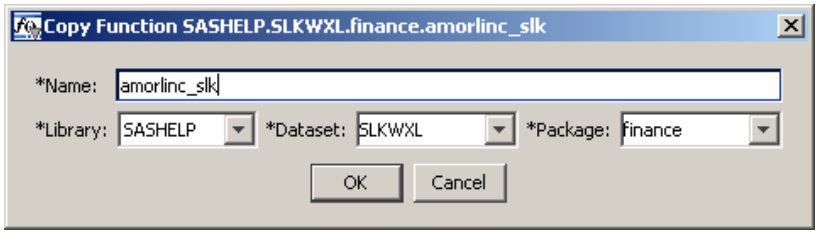
Close a Function

When you right-click the function name in the left pane and select **Close**, the window that displays that function closes. You can also close the function by clicking **OK** in the bottom right corner of the window.

Duplicate a Function

You can copy a function that you are viewing to an existing or new package or library to which you have write access. To do this task, click the function name in the left pane to select it, and then right-click the function name and select **Duplicate** from the menu. The following dialog box appears:

Display 23.7 The Duplicate Function Dialog Box



The fields in this dialog box automatically display the function name, library, data set, and package of the function you want to copy. You can change these fields when you copy the function.

For a description of these fields, see “Move a Function” on page 480.

Export a Function to a File

To export a function to a file, click a function name in a library to select the function. Then right-click the function and select **Export to File** from the menu. At the top of the Save dialog box that appears, you can see the current location of the function, (for example, SASUSER.EXSUBS.pkt1.calc-years). The function CALC-YEARS resides in the package called pkt1, in the data set EXSUBS, in the SASUSER library.

In the **Save in** field, select the directory to which you want the function exported.

Rename a Function

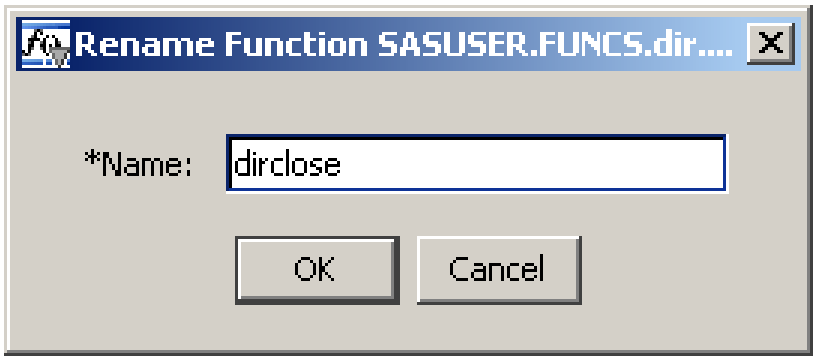
Use the Rename dialog box to rename a function within a given package. You must have write access to the library that contains the function. When you rename a function, the new function resides in the same library as the original function.

CAUTION:

Rename enables you to rename a function within a given package. Just as with moving a function, the renaming of a function does not modify dependent macros and other entities. △

To rename a function, first select the function and then right-click the function and select **Rename** from the menu. The following dialog box appears:

Display 23.8 The Rename Dialog Box

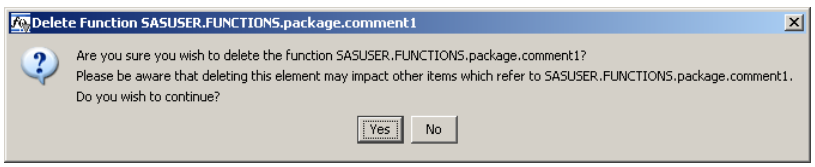


Enter the new name of the function and click **OK**.

Delete a Function

You can delete a function from a library to which you have write access. To delete a function, first select the function you want to delete. Right-click the function and select **Delete** from the menu. The following dialog box appears, cautioning you about the impact that **Delete** has on other items:

Display 23.9 Delete Function Confirmation Dialog Box



Click **Yes** or **No**.

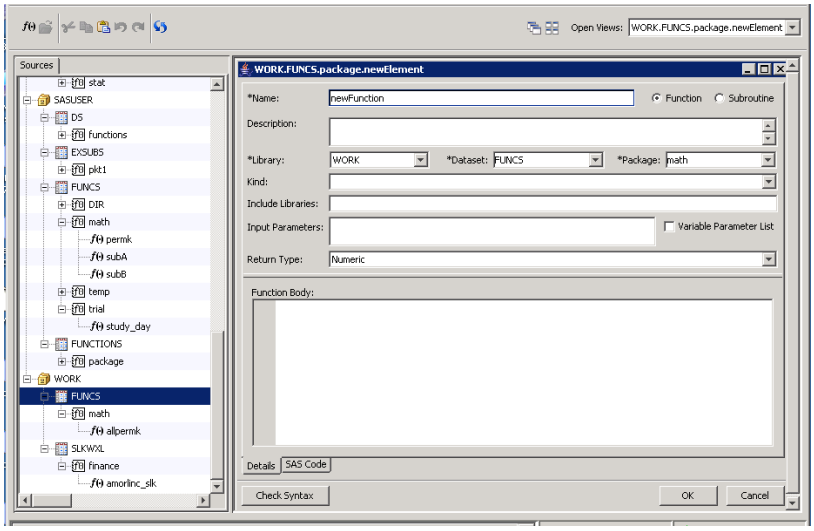
Print a Function

Creating a New Function

You can create a new function whenever you have a library, data set, or package selected. To create a new function in a library, position your cursor on the library into which the new function will be added. Right-click the library and select **New Function**. You can also select **File ▶ New Function** from the menu or click

in the upper left corner below the menu bar. The following window appears:

Display 23.10 The newElement Window



The upper right corner of the window contains two buttons: **Function** and **Subroutine**. Click one of the buttons depending on whether you want to create a new function or a new subroutine.

The newElement window contains the following fields:

Name specifies the name of the new function.

Description describes the new function.

Library specifies the library that will contain the new function. Enter the name of the data set, or click the down arrow in the **Library** field to select a library.

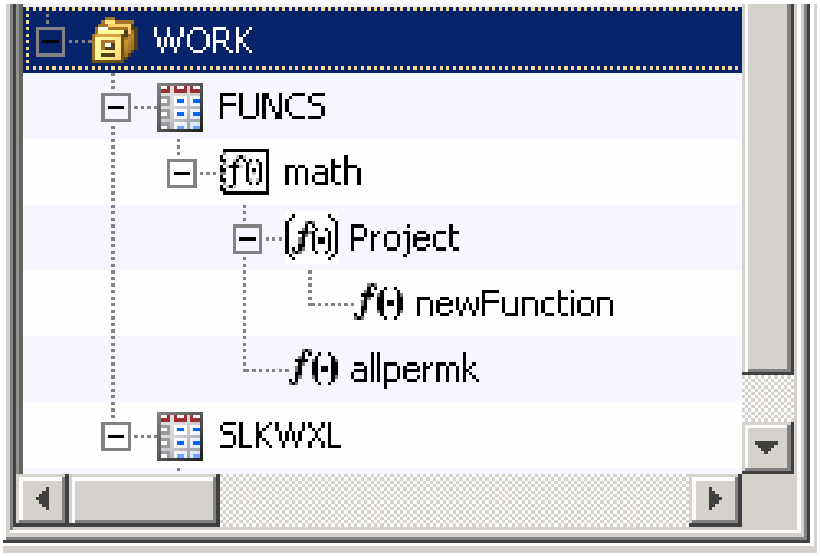
Dataset specifies the data set that will contain the new function. Enter the name of the data set, or click the down arrow in the **Dataset** field to select a data set.
If you do not specify a value, the value in this field defaults to Functions.

Package specifies the name of the package that will contain the new function. Enter the name of the data set, or click the down arrow in the **Package** field to select a data set.
The **Package** field is a required field If you do not specify a value, the value in this field defaults to Package.

Kind enables you to group functions within a given package. Four predefined kind groupings are available and are typically used with SAS Risk Management:

- Project
- Risk Factor Transformation
- Instrument Pricing
- Instrument Input

You can use one of these four groupings, or enter your own kind value in the **Kind** field. The function tree in the left pane groups the functions in a package into their kind grouping, if you specified a value for **Kind**. In this example, the function newFunction was created with a **Kind** value of Project, in the package math, in the data set FUNCS, and in the WORK library.



Include Libraries specifies libraries that contain SAS code that you want to include in your function.

Input Parameters specifies the arguments that you use as input to the function.

Variable Parameter List specifies whether the function supports a variable number of arguments.

Return Type specifies whether the function returns a character or numeric value.

Function Body is the area in the window in which you code your function.

Two tabs are located at the bottom left of the newElement window:

Details tab provides you with an area in which to write descriptive information (name of the new function, list of include libraries, input parameters,

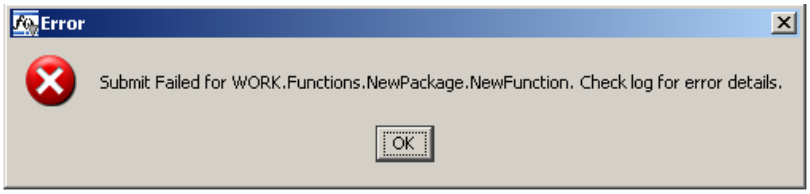
and so forth) about your function. You code your new function in the **Function Body** section. The **Details** tab is selected by default.

SAS Code tab enables you to view the function you have written. The **SAS Code** selection provides read-only capabilities.

When you enter information in the descriptive portion of the **Details** tab, as well as in the **Function Body** section, the information is converted to SAS code that you can see when you select the **SAS Code** tab.

The newElement window contains a **Check Syntax** button that is located at the bottom of the two tabs. When you click this button, SAS checks the syntax of the function that you wrote. If the syntax contains an error, the following dialog box appears:

Display 23.11 Program Error Dialog Box



You are instructed to check the log for errors. To check the log, select **View ▶ Show Log** from the menu.

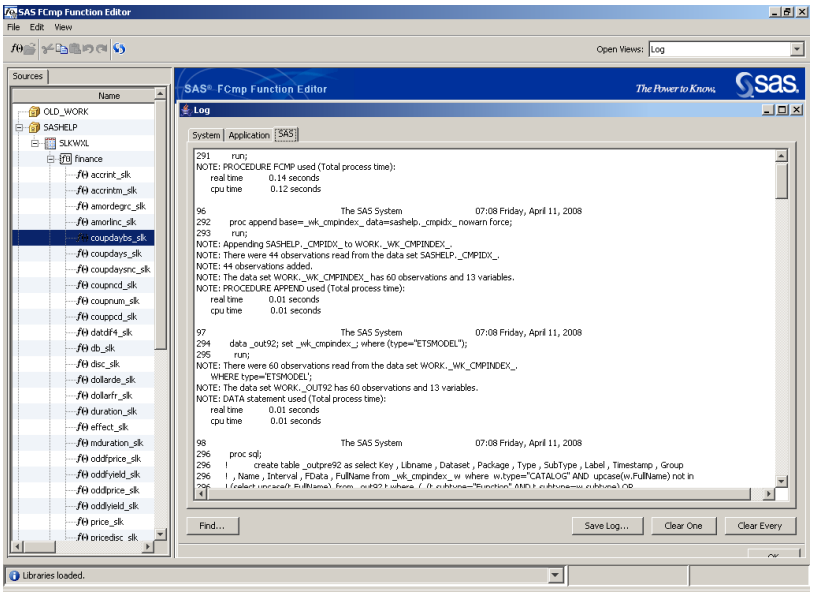
Viewing the Log Window, Function Browser, and Data Explorer

Log Window

You can display the log window by selecting **View ▶ Show Log** from the menu. The content of the log represents output from the SAS server. In addition, commands that are sent to SAS are also present to add context to the log output.

When you display the Log window, you can view system, application, and program results by selecting the tabs that are located in the upper left corner of the window. The following display shows the Log window with the SAS log displayed:

Display 23.12 The Log Window



You can see different results when you select the following tabs:

`System` tab — displays the System window. Detailed information in the form of system messages is located here. The window will be blank if no messages are logged.

The System window contains two vertical tabs that are located in the upper right section of the window. These tabs provide complete information about messages that might be of interest:

- **`System.out`** tab — displays system output if messages are routed to this location.
- **`System.err`** tab — displays error messages if the messages are routed to this location.

`Application` tab — displays the processing messages for the application.

`SAS` tab — displays the execution results in the SAS log.

You can save the log results to a file, and then clear the System, Application, and Log results. Three buttons that are located at the bottom right of the Log window enable you to perform these tasks:

`Save Log` — saves the log output to a file that you choose in your directory.

`Clear One` — clears the results in the active window.

`Clear All` — clears the results in all three of the windows.

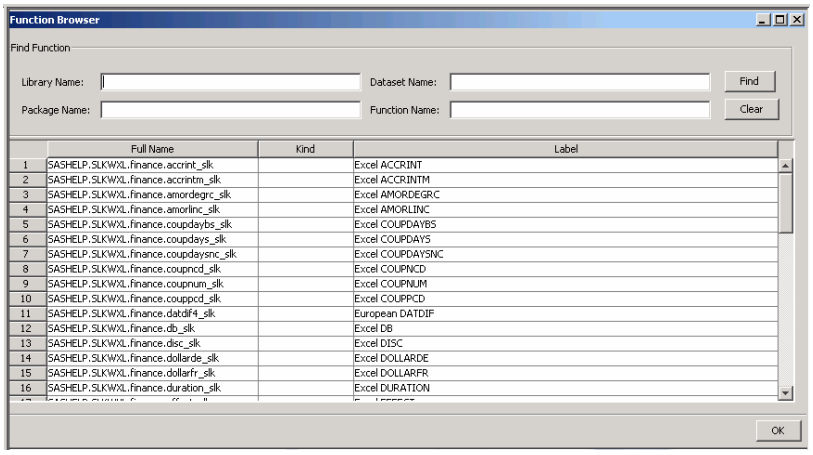
The **`Find`** button is located at the bottom left of the window. Use this button to search through your text to find a string that you entered in the **`Find`** field.

Function Browser

The Function Browser initially displays all of the functions that are listed in the left pane of the window. You can filter this list of functions to display a subset of the functions.

You display the Function Browser by selecting **View ▸ Show Function Browser** from the menu. A window similar to the following appears:

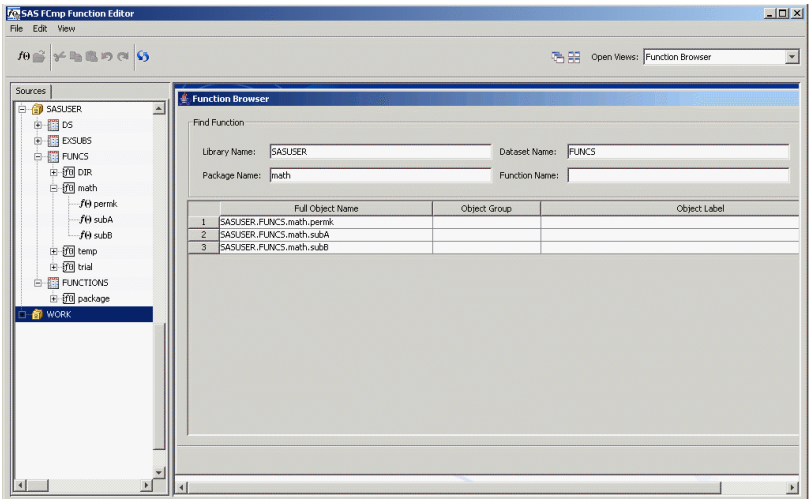
Display 23.13 The Function Browser Window



The output listed above shows all of the functions in the application tree. You can filter the output and create a subset of the functions by entering your criteria in the Function Browser fields that are located above the list of functions, These fields are **`Library Name`**, **`Dataset Name`**, **`Package Name`**, and **`Function Name`**.

The following display shows that three fields are filled in. When you press the **`OK`** button that is located in the bottom right corner of the window, or if you press the **`Find`** button that is located in the upper right corner, the following window appears:

Display 23.14 Filtered Output from the Function Browser



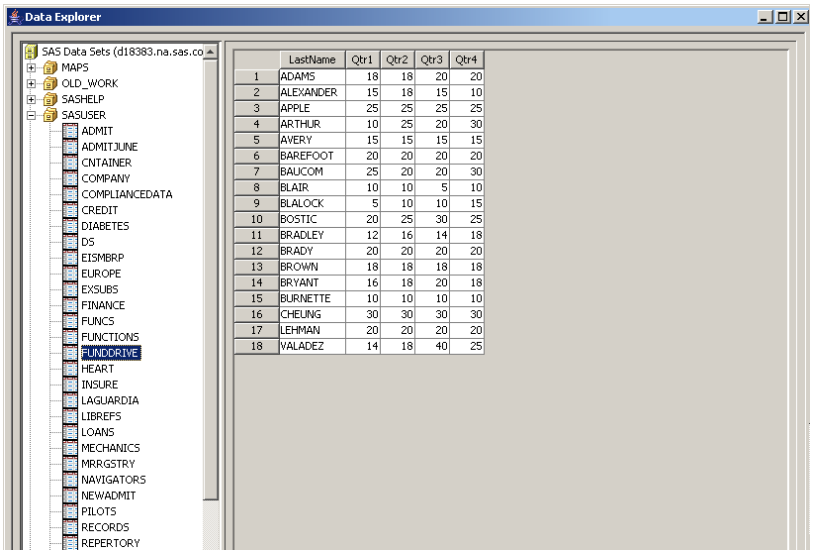
The functions that are listed are located in the SASUSER library, the FUNCS data set, and the math package.

You can enter information in the fields you choose. For example, if you enter a value in the **`Library Name`** field only, then all of the functions that are in the SASHELP library display.

Data Explorer

The Data Explorer enables you to view the data in a data set that you select. To display the Data Explorer, select **View ▸ Show Data Explorer** from the menu. A window similar to the following appears:

Display 23.15 The Data Explorer Window



	LastName	Qtr1	Qtr2	Qtr3	Qtr4
1	ADAMS	18	18	20	20
2	ALEXANDER	15	18	15	10
3	APPLE	25	25	25	25
4	ARTHUR	10	25	20	30
5	AVERY	15	15	15	15
6	BAREFOOT	20	20	20	20
7	BAUCOM	25	20	20	30
8	BLAIR	10	10	5	10
9	BLALOCK	5	10	10	15
10	BOSTIC	20	25	30	25
11	BRADLEY	12	16	14	18
12	BRADY	20	20	20	20
13	BROWN	18	18	18	18
14	BRYANT	16	18	20	18
15	BURNETTE	10	10	10	10
16	CHEUNG	30	30	30	30
17	LEHMAN	20	20	20	20
18	VALADEZ	14	18	40	25

The Data Explorer window displays data set information based on which data set you select from the left pane.

By clicking the column headings, you can move the columns to reposition them in the display. When you click **OK** in the lower right section of the window, the changes you made are saved.

Using the Function You Select in Your DATA Step Program

For an example of how PROC FCMP and DATA step syntax work together, see “Directory Transversal” on page 445.

Examples: FCMP Procedure

Example 1: Creating a Function and Calling the Function from a DATA Step

Procedure features:

PROC FCMP statement option

OUTLIB=

DATA step

This example shows how to compute a study day during a drug trial by creating a function in FCMP and using that function in a DATA step.

Program

Specify the name of an output package to which the compiled function and CALL routine are written. The package is stored in the data set Sasuser.Funcs.

```
proc fcmp outlib=sasuser.funcs.trial;
```

Create a function called STUDY_DAY. STUDY_DAY is created in a package called Trial, and contains two numeric input arguments.

```
    function study_day(intervention_date, event_date);
```

Use a DATA step IF statement to calculate EVENT-DATE. Use DATA step syntax to compute the difference between EVENT_DATE and INTERVENTION_DATE. The days before INTERVENTION_DATE begin at -1 and become smaller. The days after and including INTERVENTION_DATE begin at 1 and become larger. (This function never returns 0 for a study date.)

```
       n = event_date - intervention_date;
          if n >= 0 then
             n = n + 1;
          return (n);
    endsub;
```

Use the CMPLIB= system option to specify a SAS data set that contains the compiler subroutine to include during program compilation.

```
options cmplib=sasuser.funcs;
```

Create a DATA step to produce a value for the function STUDY_DAY. The function uses a start date and today's date to compute the value. STUDY_DAY is called from the DATA step. When the DATA step encounters a call to STUDY_DAY, it does not find this function in its traditional library of functions. It searches each of the data sets that are specified in the CMPLIB system option for a package that contains STUDY_DAY. In this case, it finds STUDY_DAY in sasuser.funcs.trial.

```
data _null_;
   start = '15Feb2008'd;
   today = '27Mar2008'd;
   sd = study_day(start, today);
```

Write the output to the SAS log.

```
    put sd=;
```

Execute the SAS program.

```
run;
```

Output

```
sd=42
```

Example 2: Creating a CALL Routine and a Function

Procedure features:

PROC FCMP statement option
OUTLIB=
OUTARGS statement

This example shows how to use PROC FCMP to create and store CALL routines and functions.

Program

Specify the entry where the function package information is saved. The package is a three-level name.

```
proc fcmp outlib = sasuser.exsubs.pkt1;
```

Create a function to calculate years to maturity. A generic function called CALC_YEARS is declared to calculate years to maturity from date variables that are stored as the number of days. The OUTARGS statement specifies the variable that will be updated by CALC_YEARS.

```
subroutine calc_years(maturity, current_date, years);
   outargs years;
   years = (maturity - current_date) / 365.25;
endsub;
```

Create a function for Garman-Kohlhagen pricing for FX options. A function called GARKHPRC is declared, which calculates Garman-Kohlhagen pricing for FX options. The function uses the SAS functions GARKHCLPRC and GARKHPTPRC.

```
function garkhprc (type$, buysell$, amount,
                   E, t, S, rd, rf, sig);
   if buysell = "Buy" then sign = 1.;
   else do;
      if buysell = "Sell" then sign = -1.;
      else sign = .;
   end;

   if type = "Call" then
      garkhprc = sign * amount
                      * garkhptprc (E, t, S, rd, rf, sig);
   else do;
      if type = "Put" then
         garkhprc = sign * amount
                         * garkhptprc (E, t, S, rd, rf, sig);
      else garkhprc = .;
   end;
```

The RETURN statement returns the value of the GARKHPRC function.

```
   return (garkhprc);
endsub;
```

Execute the FCMP procedure. The RUN statement executes the FCMP procedure.

```
run;
```

SAS Log

Output 23.28

```
NOTE: Function garkhprc saved to sasuser.exsubs.pkt1.
NOTE: Function calc_years saved to sasuser.exsubs.pkt1.
```

Example 3: Executing PROC STANDARDIZE on Each Row of a Data Set

Procedure features:

PROC FCMP functions

RUN_MACRO
RUN_SASFILE
READ_ARRAY
WRITE_ARRAY

This example shows how to execute PROC STANDARDIZE on each row of a data set.

Program

Create a data set that contains five rows of random numbers.

```
data numbers;
   drop i j;
   array a[5];
   do j = 1 to 5;
   do i = 1 to 5;
         a[i] = ranuni(12345) * (i+123.234);
      end;
      output;
      end;
   run;
```

Create a macro to standardize a data set with a given value for mean and std.

```
%macro standardize;
%let dsname = %sysfunc(dequote(&dsname));
%let colname = %sysfunc(dequote(&colname));
proc standard data = &dsname mean = &MEAN std = &STD out=_out;
   var &colname;
run;
data &dsname;
   set_out;
run;
%mend standardize;
```

Use the FCMP function to call WRITE_ARRAY, which writes the data to a data set. Call RUN_MACRO to standardize the data in the data set. Call WRITE_ARRAY to write data to a data set. Call READ_ARRAY to read the standardized data back into the array.

```
proc fcmp outlib = sasuser.ds.functions;
   subroutine standardize(x[*], mean, std);
      outargs x;

      rc = write_array('work._TMP_', x, 'x1');
      dsname = 'work._TMP_';
         colname = 'x1';
         rc = run_macro('standardize', dsname, colname, mean, std);
      array x2[1]_temporary_;
         rc = read_array('work._TMP_', x2);
      if dim(x2) = dim(x) then do;
         do i = 1to  dim(x);
            x[i] = x2[i];
          end;
         end;
   endsub;
run;
```

Execute the function for each row in the DATA step.

```
options cmplib = (sasuser.ds);
data numbers2;
   set numbers;
   array a[5];
   array t[5]_temporary_;
   do i = 1 to 5;
      t[i] = a[i];
   end;
   call standardize(t, 0, 1);
   do i = 1 to 5;
      a[i] = t[i];
   end;
   output;
run;

data numbers;
       drop i j;
       array a[5];
       do j = 1 to 5;
          do i = 1 to 5;
             a[i] = ranuni(12345) * (i+123.234);
         end;
          output;
       end;
    run;
```

Write the output.

```
options nodate pageno=1 ls=80 ps=64;
proc print data=work.numbers;
run;
```

Output

```
                              The SAS System                                1

          Obs       a1         a2         a3         a4         a5

            1      45.088    93.3237    104.908     35.152    23.5725
            2      90.552     9.7548     92.696     89.987    97.9810
            3      60.596    22.7409     19.284     50.079    58.9264
            4     106.778    49.1589     22.885     20.641    30.1756
            5      34.812    71.3746     44.248    101.808    79.3731
```

Example 4: Using GTL with User-Defined Functions

Procedure features:

PROC FCMP functions

OSCILLATE
OSCILLATEBOUND

Other procedures

PROC TEMPLATE
PROC SGRENDER

The following example shows how to define functions that define new curve types (oscillate and oscillateBound). These functions can be used in a GTL EVAL function to compute new columns that are presented with a seriesplot and bandplot.

Program

Create the OSCILLATE function.

```
proc fcmp outlib=sasuser.funcs.curves;
   function oscillate(x,amplitude,frequency);
         if amplitude le 0 then amp=1; else amp=amplitude;
       if frequency le 0 then freq=1; else freq=frequency;
         y=sin(freq*x)*constant("e")**(-amp*x);
         return (y);
  endsub;
```

Create the OSCILLATEBOUND function.

```
 function oscillateBound(x,amplitude);
       if amplitude le 0 then amp=1; else amp=amplitude;
       y=constant("e")**(-amp*x);
       return (y);
 endsub;
run;
```

Create a data set called RANGE that will be used by PROC SGRENDER.

```
 options cmplib=sasuser.funcs;

data range;
   do Time=0 to 2 by .01;
   output;
   end;
run;
```

Use the TEMPLATE procedure to customize the appearance of your SAS output.

```
proc template ;
   define statgraph damping;
   dynamic X AMP FREQ;
   begingraph;
       entrytitle "Damped Harmonic Oscillation";
       layout overlay / yaxisopts=(label="Displacement");
          if (exists(X) and exists(AMP) and exists(FREQ))
             bandplot x=X  limitlower=eval(-oscillateBound(X,AMP))
                limitupper=eval(oscillateBound(X,AMP));
                seriesplot x=X y=eval(oscillate(X,AMP,FREQ));
       endif;
     endlayout;
   endgraph;
   end;
   run;
```

Open the HTML destination to view graphic output.

```
ods html;
```

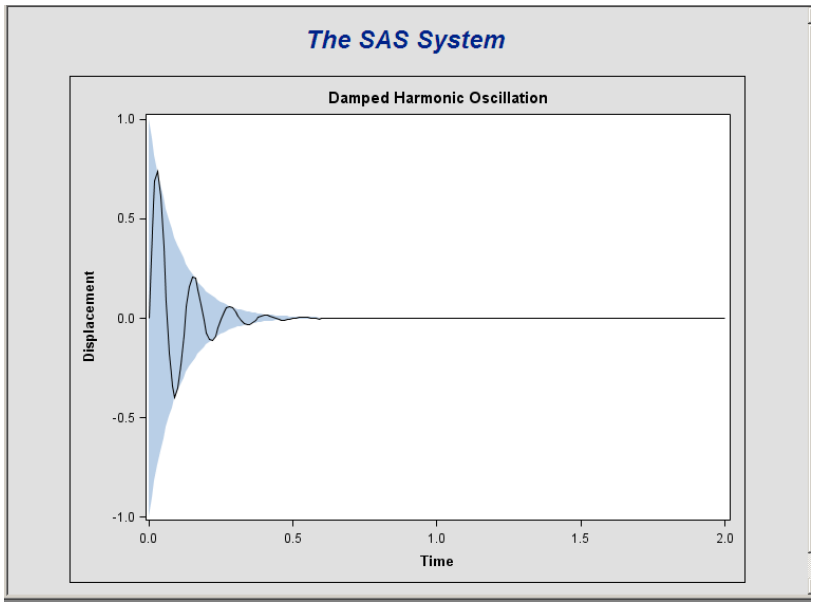
Use the SGRENDER procedure to identify the data set that contains the input variables and to assign a statgraph template for the output.

```
proc sgrender data=range template=damping;
     dynamic x="Time" amp=10 freq=50 ;
run;
```

Close the HTML destination.

```
     ods html close;
```

Output


The SAS System
Damped Harmonic Oscillation
Displacement
1.0
0.5
0.0
-0.5
-1.0
0.0
0.5
1.0
1.5
2.0
Time

CHAPTER

24

The FONTREG Procedure

Overview: FONTREG Procedure **497**
Syntax: FONTREG Procedure **498**
PROC FONTREG Statement **498**
FONTFILE Statement **499**
FONTPATH Statement **501**
REMOVE Statement **502**
TRUETYPE Statement **503**
TYPE1 Statement **503**
Concepts: FONTREG Procedure **504**
Supported Font Types and Font Naming Conventions **504**
Removing Fonts from the SAS Registry **505**
Font Aliases and Locales **506**
Examples: FONTREG Procedure **506**
Example 1: Adding a Single Font File **506**
Example 2: Adding All Font Files from Multiple Directories **507**
Example 3: Replacing Existing TrueType Font Files from a Directory **508**

Overview: FONTREG Procedure

The FONTREG procedure enables you to update the SAS registry to include system fonts, which can then be used in SAS output. PROC FONTREG uses FreeType font-rendering to recognize and incorporate various types of font definitions. Fonts of any type that can be incorporated and used by SAS are known collectively in this documentation as fonts in the FreeType library.

Note: Including a system font in the SAS registry means that SAS knows where to find the font file. The font file is not actually used until the font is called for in a SAS program. Therefore, do not move or delete font files after you have included the fonts in the SAS registry. △

For more information about font-rendering, see Font and Font Rendering in *SAS/GRAPH: Reference*.

Syntax: FONTREG Procedure

Interaction: If no statements are specified, then PROC FONTREG searches for TrueType font files in the directory that is indicated in the FONTSLOC= SAS system option.

Tip: If you specify more than one statement, then the statements are executed in the order in which they appear, except for REMOVE statements, which are always executed first. You can use the same statement more than once in a single PROC FONTREG step.

See FONTREG Procedure in the *SAS Companion for z/OS*

PROC FONTREG <*option(s)*>;
FONTFILE '*file*' <*…'file'*> | | '*file-1*, *pfm-file-1*, *afm-file-1*' <*...'file-n'*>;
FONTPATH '*directory*' <*…'directory'*>;
REMOVE '*family-name*' | '*alias*' | *family-type* | _ALL_;
TRUETYPE '*directory*' <*…'directory'*>;
TYPE1 '*directory*' <*…'directory'*>;

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. △

Task	Statement
Specify how to handle new and existing fonts.	"PROC FONTREG Statement" on page 498
Indentify which font files to process.	
Search directories to identify valid font files to process. (In the Windows operating environmen only, locate the fonts folder if you don't know where the folder is located.)	"FONTPATH Statement" on page 501
Remove a font family, all fonts of a particular type, or all fonts from the Core\Printing\Freetype\Fonts location of the SAS registry.	"REMOVE Statement" on page 502
Search directories to identify TrueType font files.	"TRUETYPE Statement" on page 503
Search directories to identify valid Type 1 font files.	"TYPE1 Statement" on page 503

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 508

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 507

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 506

FONTFILE *'file'* <*…'file'*> | | *'file-1*, *pfm-file-1*, *afm-file-1'* <*...'file-n'*>;

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.

pfm-file
specifies a Windows-specific file that contains font metrics as well as the value of the Windows font name.

afm-file
specifies a file that contains font metrics.

Details

Processing a Type1 Font

When a valid Type1 font is processed by the TYPE1 or the FONTPATH statements, SAS attempts to find a corresponding PFM or AFM font metric file in the same directory that contains the font file. The font filename prefix is used with the .PFM and .AFM extensions to generate metric filenames. If these files are opened successfully and are determined to be valid metric files, then they will be associated with the font in the font family when they are added to the SAS registry.

If you specify a Type1 font on the FONTFILE statement, and you do not specify a PFM or an AFM file, then SAS does not search for the PFM or the AFM files.

Specifying a PFM or an AFM File

If the font file contains a Type1 font, then you can also specify its corresponding PFM or AFM file as well. You must specify the full host name (directory and filename) for each file, and all files must be grouped together and enclosed in quotation marks, as in this example:

```
fontfile 'c:\winnt\fonts\alpinerg.pfb,
          c:\winnt\fonts\alpinerg.pfm,
          c:\winnt\fonts\alpinerg.afm';
```

If you specify an AFM file but do not specify a PFM file, then you must use a comma as a placeholder for the missing PFM file, as in this example:

```
fontfile 'c:\winnt\fonts\alpinerg.pfb, , c:\winnt\fonts\alpinerg.afm';
```

If you specify a PFM file but do not specify an AFM file, then you do not need a comma as a placeholder for the missing AFM file, as in this example:

```
fontfile 'c:\winnt\fonts\alpinerg.pfb, c:\winnt\fonts\alpinerg.pfm';
```

When you specify a PFM or an AFM file, SAS attempts to open the file and determine whether the file is of the specified type. If it is not, then SAS writes a message to the log and the file is not used.

The PFM file is a Windows-specific file that contains font metrics as well as a value for the Windows Font Name field. If you specify a valid PFM file, then SAS opens the file, retrieves the value in Windows Font Name, and saves it with the font in the SAS

registry. SAS uses this field when it creates a file (such as an EMF formatted file) to export into a Windows application.

Not Specifying a PFM or an AFM File

You do not need to specify a PFM or an AFM file along with a Type1 font file on a FONTFILE statement. In this case, no metric file information is added to the font in the font family in the SAS registry. If an existing font family that contains multiple styles and weights already exists in the SAS registry, and the FONTFILE statement is used to replace one of the fonts in that family, then all of the information for that font will be updated. The replacement also updates the Host Filename, PFM Name, AFM Name, and Windows Font Name.

Note: If you replace a font in a family and the font contains values for the PFM Name or AFM Name, specifying a missing or invalid value for the metric on the FONTFILE statement causes the corresponding metric value to be deleted from the font in the registry. △

Note: You cannot use a PFM or an AFM file specification if you specify a TrueType font. △

FONTPATH Statement

Specifies one or more directories to be searched for valid font files to process.

Featured in: Example 2 on page 507

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.

Operating Environment Information: In the Windows operating environment only, you can locate the fonts folder if you do not know where the folder resides. In addition, you can register system fonts without having to know where the fonts are located. To find this information, submit the following program:

```
proc fontreg;
   fontpath "%sysget(systemroot)\fonts";
run;
```

The %SYSGET macro retrieves the value of the Windowing environment variable SYSTEMROOT, and resolves to the location of your system directory. The fonts subdirectory is located one level below the system directory. △

REMOVE Statement

Removes a font family, all fonts of a particular type (such as TrueType or Type1), or all fonts from the Core\Printing\Freetype\Fonts location of the SAS registry.

REMOVE *'family-name'* | *'alias'* | *family-type* | _ALL_;

Arguments

family-name
specifies the family name of the font that you want to remove from the Core\Printing\Freetype\Fonts key in the SAS registry. Enclose *family-name* in quotation marks if the value contains one or more spaces.

alias
specifies an alternative name, usually in a shortened form, for *family-name*. Enclose the alias name in quotation marks if the value contains one or more spaces.

family-type
specifies the name of a font type (such as TrueType or Type1) that SAS supports and that you want removed from the SAS registry.

Note: The font type is not removed from the operating system location in which they reside. The registration of the font type from the SAS registry is removed so that SAS does not recognize the fonts. △

ALL
specifies that all font families in the Core\Printing\Freetype\Fonts key in the SAS registry will be deleted.

Details

Removing Fonts from the Registry

The REMOVE statement removes a font family, all fonts of a particular type, or all fonts from the Core\Printing\Freetype\Fonts location in the SAS registry. If you specify the USESASHELP procedure option, then fonts are removed from the SASHELP portion of the registry. If you do not specify USESASHELP, then fonts are removed from the SASUSER portion of the registry. Removal from the SASUSER portion of the registry is the default.

Note that when you specify the *family-name* argument in the REMOVE statement, SAS removes font families rather than individual fonts within the family. For example, you might register several fonts within the Arial family. When you use the **`REMOVE Arial;`** statement, all fonts in the Arial family are removed from the registry. Similarly, when you specify the *family-type* argument and use the **`REMOVE Type1;`** statement, all Type1 font families are removed from the registry.

The Order in Which Fonts Are Added or Removed

Fonts are removed from the SAS registry before any fonts are added or replaced in the registry using other procedure statements. The REMOVE statement removes a font family from the registry as soon as the statement is processed. Other font statements (FONTFILE, FONTPATH, TRUETYPE, and TYPE1) are processed in the order that

they are received, but the font information is stored until all of the statements are processed. SAS then updates the registry.

Searching for a Font That Is Specified in the REMOVE Statement

If the name that you specify in a REMOVE statement does not exist, then SAS adds a font tag prefix (for example, <ttf>) to the specified name to determine whether it exists in the SAS registry. For example, if you specify Arial, SAS uses the <ttf> prefix tag and first searches for a TrueType font type so that it can be removed from the registry. If the search is not successful, then SAS uses the <at1> prefix tag and searches for a Type1 font type so that it can be removed from the registry.

When SAS Is Unable to Remove a Font Family

If SAS is unable to remove a font family after processing the information in the _ALL_, *family-type*, or *family-name* arguments, then SAS looks in the Core\Printing\Alias\Fonts\Freetype key in the SAS registry to determine whether the specified value is an alias. If the specified value exists as an alias in this key, then SAS deletes the font family that corresponds to the alias and deletes the alias as well. For example, if an alias of Test refers to the Arial font family, and you specify the **`REMOVE test;`** statement with PROC FONTREG, then SAS determines that Test is an alias for Arial. SAS removes the Arial font family from the Core\Printing\Freetype\Fonts key and the Test alias from Core\Printing\Alias\Fonts\Freetype key in the SAS registry.

If SAS is unable to remove a font family at this point, then SAS writes a message to the log indicating that the specified value on the REMOVE statement is invalid.

TRUETYPE Statement

Specifies one or more directories to be searched for TrueType font files.

Featured in: Example 3 on page 508

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

Specifies one or more directories to be searched for valid Type1 font files.

TYPE1 *'directory'* <*...'directory'*>;

Argument

directory
specifies a directory to search. Only files that are recognized as valid Type1 font files are processed. Each directory must be enclosed in quotation marks. If you specify more than one directory, then you must separate the directories with a space.

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 found, then SAS uses the font that has the highest rank in the following table.

Table 24.1 Supported Font Types

Rank	Type	Tag	File extension(s)
1	TrueType	**<ttf>**	**.ttf**
2	Type1	**<at1>**	**.pfa**
			.pfb

Note: SAS does not support any type of nonscalable fonts that require Free-Type font-rendering. Even if they are recognized as valid fonts, they will not be added to the SAS registry. △

Font files that are not produced by major vendors can be unreliable, and in some cases SAS might not be able to use them.

The following SAS output methods and device drivers can use FreeType font-rendering:

- SAS/GRAPH GIF, GIF733, GIFANIM
- SAS/GRAPH JPEG
- SAS/GRAPH PCL

- SAS/GRAPH PNG
- SAS/GRAPH SASEMF
- SAS/GRAPH SASWMF
- SAS/GRAPH TIFFP, TIFFB
- Universal PNG
- Universal Printing GIF
- Universal Printing PCL
- Universal Printing PDF
- Universal PS
- Universal SVG

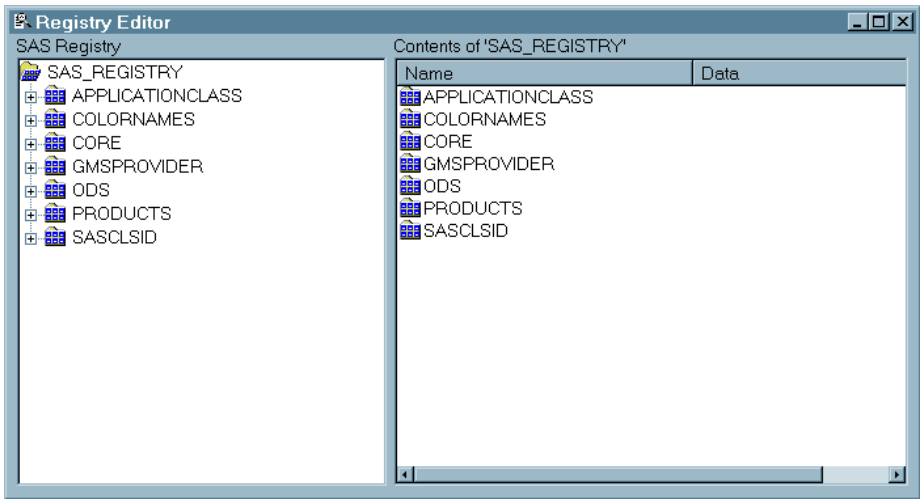
Removing Fonts from the SAS Registry

You can remove a font from the SAS registry in the following ways:

- by using the SAS Registry Editor
- by using PROC REGISTRY
- by using the REMOVE statement in PROC FONTREG

To remove a font by using the SAS Registry Editor, select **Solutions ▶ Accessories ▶ Registry Editor**. Alternatively, you can type **`regedit`** in the command window or **`Command ===>`** prompt.

Display 24.1 SAS Registry Editor



In the left pane of the Registry Editor window, navigate to the [CORE\PRINTING\FREETYPE\FONTS] key. Select the font that you want to delete, and use one of these methods to delete it:

- Right-click the font name and select **`Delete`** from the menu.
- Select the **`Delete`** button  .
- Select **Edit ▶ Delete ▶ Key**.

To delete a font by using PROC REGISTRY, submit a program similar to the following example. This example removes the **`<ttf> Arial`** font.

```
/* Write the key name for the font to an external file */
proc registry export='external-filename'
              startat='core\printing\freetype\fonts\<ttf> Arial';
run;

/* Remove the "<ttf> Arial" font from the SAS registry */
proc registry uninstall='external-filename' fullstatus;
run;
```

To delete a font by using the REMOVE statement in PROC FONTREG, see the "REMOVE Statement" on page 502.

For more information about PROC REGISTRY, see Chapter 50, "The REGISTRY Procedure," on page 963.

Font Aliases and Locales

The FONTFILE, FONTPATH, and TRUETYPE statements support aliases and locales. If the font being processed contains a localized name in the same locale as the current SAS session, then an alias of that localized name will be added to the SAS registry to reference the font family.

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;
```

Output: SAS 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
  Font Families removed from SAS registry: 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;
```

Output: SAS Log (Partial)

```
1    proc fontreg msglevel=verbose;
2    fontpath 'your-font-directory-1'
3             'your-font-directory-2';
4    run;

ERROR: FreeType base module FT_New_Face -- unknown file format.
ERROR: A problem was encountered with file
       "your-font-directory-2\MODERN.FON".

. . .  more log entries . . .

WARNING: The "Sasfont" font in file
         "your-font-directory-2\SAS1252.FON" is non-scalable. Only
         scalable fonts are supported.

. . .  more log entries . . .


NOTE: The font "Albertus Medium" (Style: Regular, Weight: Normal) has been
      added to the SAS Registry at
      [CORE\PRINTING\FREETYPE\FONTS\<ttf>Albertus Medium]. Because it is a TRUETYPE
      font, it can be referenced as "Albertus Medium" or "<ttf>Albertus Medium" in
      SAS. The font resides in file
      "your-font-directory-1\albr55w.ttf".

. . .  more log entries . . .


WARNING: The font "Georgia" (Style: Regular, Weight: Normal) will not be added
because it already exists in the "<ttf>Georgia" font family of the SAS Registry.

. . .  more log entries . . .

SUMMARY:
  Files processed: 138
  Unusable files: 3
  Files identified as fonts: 135
  Fonts that were processed: 135
  Fonts replaced in the SAS registry: 0
  Fonts added to the SAS registry: 91
  Fonts that could not be used: 44
  Font Families removed from SAS registry: 0

NOTE: PROCEDURE FONTREG used (Total process time):
      real time           27.81 seconds
      cpu time            1.18 seconds
```

Example 3: Replacing Existing TrueType Font Files from a Directory

Procedure features:

MODE= option

TRUETYPE statement

This example reads all the TrueType fonts in the specified directory and replaces the ones 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;
```

Output: SAS Log

```
53   proc fontreg mode=replace;
54      truetype 'your-font-directory';
55   run;
SUMMARY:
  Files processed: 49
  Unusable files: 3
  Files identified as fonts: 46
  Fonts that were processed: 40
  Fonts replaced in the SAS registry: 40
  Fonts added to the SAS registry: 0
  Fonts that could not be used: 0
  Font Families removed from SAS registry: 0

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*

The FONTSLOC and SYSPRINTFONT SAS system options in *SAS Language Reference: Dictionary*

http://www.freetype.org for more information about the FreeType project.

CHAPTER

25

The FORMAT Procedure

Overview: FORMAT Procedure **512**
What Does the FORMAT Procedure Do? **512**
What Are Formats and Informats? **512**
How Are Formats and Informats Associated with a Variable? **512**
Syntax: FORMAT Procedure **513**
PROC FORMAT Statement **514**
EXCLUDE Statement **516**
INVALUE Statement **517**
PICTURE Statement **520**
SELECT Statement **530**
VALUE Statement **531**
Informat and Format Options **534**
Specifying Values or Ranges **536**
Concepts: FORMAT Procedure **537**
Associating Informats and Formats with Variables **538**
Methods of Associating Informats and Formats with Variables **538**
Differences between the FORMAT Statement and PROC FORMAT **538**
Assigning Formats and Informats to a Variable **538**
Storing Informats and Formats **539**
Format Catalogs **539**
Temporary Informats and Formats **539**
Permanent Informats and Formats **539**
Accessing Permanent Informats and Formats **539**
Missing Informats and Formats **540**
Printing Informats and Formats **540**
Results: FORMAT Procedure **541**
Output Control Data Set **541**
Input Control Data Set **543**
Procedure Output **544**
Examples: FORMAT Procedure **546**
Example 1: Creating a Picture Format **547**
Example 2: Creating a Format for Character Values **549**
Example 3: Writing a Format for Dates Using a Standard SAS Format **552**
Example 4: Converting Raw Character Data to Numeric Values **554**
Example 5: Creating a Format from a Data Set **557**
Example 6: Printing the Description of Informats and Formats **561**
Example 7: Retrieving a Permanent Format **563**
Example 8: Writing Ranges for Character Strings **566**
Example 9: Filling a Picture Format **568**
Example 10: Creating a Format in a non-English Language **570**

Overview: FORMAT Procedure

What Does the FORMAT Procedure Do?

The FORMAT procedure enables you to define your own informats and formats for variables. In addition, you can print the parts of a catalog that contain informats or formats, store descriptions of informats or formats in a SAS data set, and use a SAS data set to create informats or formats.

What Are Formats and Informats?

Informats determine how raw data values are read and stored. *Formats* determine how variable values are printed. For simplicity, this section uses the terminology *the informat converts* and *the format prints*.

Informats and formats tell SAS the data's type (character or numeric) and form (such as how many bytes it occupies; decimal placement for numbers; how to handle leading, trailing, or embedded blanks and zeros; and so on). SAS provides informats and formats for reading and writing variables. For a thorough description of informats and formats that SAS provides, see the sections on formats and informats in *SAS Language Reference: Dictionary*.

With informats, you can do the following:

- Convert a number to a character string (for example, convert 1 to **YES**).
- Convert a character string to a different character string (for example, convert **'YES'** to **'OUI'**).
- Convert a character string to a number (for example, convert **YES** to 1).
- Convert a number to another number (for example, convert 0 through 9 to 1, 10 through 100 to 2, and so on).

Note: User-defined informats read only character data. They can convert character values into real numeric values, but they cannot convert real numbers into characters. △

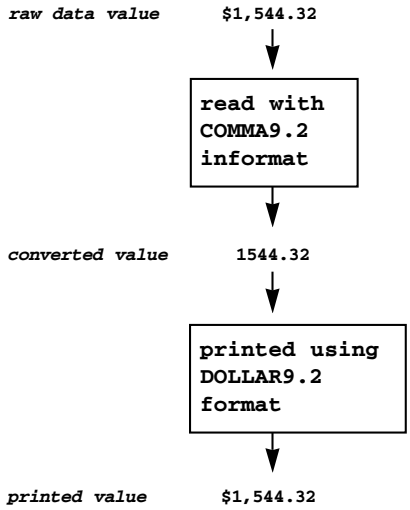
With formats, you can do the following:

- Print numeric values as character values (for example, print 1 as **MALE** and 2 as **FEMALE**).
- Print one character string as a different character string (for example, print **YES** as **OUI**).
- Print numeric values using a template (for example, print 9458763450 as **945-876-3450**).

How Are Formats and Informats Associated with a Variable?

The following figure summarizes what occurs when you associate an informat and format with a variable. The COMMA*w.d* informat and the DOLLAR*w.d* format are provided by SAS.

Display 25.1 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 538.

Syntax: FORMAT Procedure

Restriction: You cannot use a SELECT statement and an EXCLUDE statement within the same PROC FORMAT step.

Tip: You can also use appropriate global statements with this procedure. See "Global Statements" on page 20 for a list.

See: FORMAT Procedure under z/OS 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);

Task	Statement
Define formats and informats for variables.	"PROC FORMAT Statement" on page 514
Exclude catalog entries from processing by the FMTLIB and CNTLOUT= options.	"EXCLUDE Statement" on page 516
Create an informat for reading and converting raw data values.	"INVALUE Statement" on page 517
Create a template for printing numbers.	"PICTURE Statement" on page 520
Select catalog entries for processing by the FMTLIB and CNTLOUT= options.	"SELECT Statement" on page 530
Create a format that specifies character strings to use to print variable values.	"VALUE Statement" on page 531

PROC FORMAT Statement

Tip: 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)*>;

Task	Option
Specify a SAS data set from which PROC FORMAT builds informats or formats.	CNTLIN= on page 515
Create a SAS data set that stores information about informats or formats.	CNTLOUT= on page 515
Print information about informats or formats.	
Specify a SAS library or catalog that will contain the informats or formats that you are creating in the PROC FORMAT step.	LIBRARY= on page 515
Specify the number of characters of the informatted or formatted value that appear in PROC FORMAT output.	MAXLABLEN= on page 516
Specify the number of characters of the start and end values that appear in the PROC FORMAT output.	MAXSELEN= on page 516
Prevent a new informat or format from replacing an existing one of the same name.	NOREPLACE on page 516
Print information about each format and informat on a separate page[1]	PAGE on page 516

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 library (either the WORK library or USER library) for the data set, regardless of whether you specify the LIBRARY= option.

Note: LIBRARY= can point to either a library or a catalog. If only a libref is specified, a catalog name of FORMATS is assumed. △

Tip: A common source for an input control data set is the output from the CNTLOUT= option of another PROC FORMAT step.

See also: "Input Control Data Set" on page 543

Featured in: Example 5 on page 557

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 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 library (either the WORK library or the USER 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 541

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 561

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 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 539

Featured in: Example 1 on page 547

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 the FMTLIB option and you receive FMTLIB option output.

INVALUE Statement

Creates an informat for reading and converting raw data values.

See also: The section on informats in *SAS Language Reference: Dictionary* for documentation on informats supplied by SAS.

Featured in: Example 4 on page 554.

INVALUE <$>*name* <(*informat-option(s)*)>
<*value-range-set(s)*>;

Task	Option
Specify the default length of the informat.	DEFAULT= on page 534
Specify a fuzz factor for matching values to a range.	FUZZ= on page 535
Specify a maximum length for the informat.	MAX= on page 535
Specify a minimum length for the informat.	MIN= on page 535
Store values or ranges in the order that you define them.	NOTSORTED on page 535

Task	Option
Left-justify all input strings before they are compared to ranges.	JUST on page 518
Uppercase all input strings before they are compared to ranges.	UPCASE on page 518

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. Adding the dollar sign to the name 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. The addition of the at sign to the name is why the name is limited to 31 or 30 characters. You need to use the at sign *only* when you are using the name in an EXCLUDE or SELECT statement; do not prefix the name with an at sign when you are associating the informat with a variable.

Options

The following options are common to the INVALUE, PICTURE, and VALUE statements and are described in “Informat and Format Options” on page 534:

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 two hexadecimal characters per represented character.

number
is a number that becomes the informatted value. Typically, *number* becomes the value of a numeric variable when you use the informat to convert raw data. Use *number* for *informatted-value* when you are creating a numeric informat. The maximum for *number* depends on the host operating environment.

ERROR
treats data values in the designated range as invalid data. SAS assigns a missing value to the variable, prints the data line in the SAS log, and issues a warning message.

SAME
prevents the informat from converting the raw data as any other value. For example, the following GROUP. informat converts values 01 through 20 and assigns the numbers 1 through 20 as the result. All other values are assigned a missing value.

```
invalue group 01-20= _same_
              other= .;
```

existing-informat
is an informat that is supplied by SAS or 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 equal sign. If you use an existing informat, then enclose the informat name in square brackets (for example, [date9.]) or with parentheses and vertical bars, for example (|date9.|). *Do not enclose the name of the existing informat in single quotation marks.*

value-or-range
See "Specifying Values or Ranges" on page 536.
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;
```

- The CHECK. informat uses _ERROR_ and _SAME_ to convert values of 1 through 4 and 99. All other values are invalid:

```
invalue check 1-4=_same_
               99=.
            other=_error_;
```

If you use a numeric informat to convert character strings that do not correspond to any values or ranges, then you receive an error message.

PICTURE Statement

Creates a template for printing numbers.

See also: The section on formats in *SAS Language Reference: Dictionary* for documentation about formats that are supplied by SAS.

Featured in:

- Example 1 on page 547
- Example 9 on page 568

PICTURE *name* <(*format-option(s)*)>
<*value-range-set-1* <(*picture-1-option(s)*)>
<…*value-range-set-n* <(*picture-n-option(s)*)>>>;

Task	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= on page 521
Specify the default length of the format.	DEFAULT= on page 534
Specify the separator character for the fractional part of a number.	DECSEP= on page 522
Specify the three-digit separator character for a number.	DIG3SEP= on page 522
Specify a fuzz factor for matching values to a range.	FUZZ= on page 535
Specify the language that is used for the days of the week and months of the year that you can substitute in a date, time, or datetime specification.	LANGUAGE= on page 522
Specify a maximum length for the format.	MAX= on page 535
Specify a minimum length for the format.	MIN= on page 535

Task	Option
Specify multiple pictures for a given value or range and for overlapping ranges.	MULTILABEL on page 522
Store values or ranges in the order that you define them.	NOTSORTED on page 535
Round the value to the nearest integer before formatting.	ROUND on page 524
Control the attributes of each picture in the format.	
Specify a character that completes the formatted value.	FILL= on page 522
Specify a number to multiply the variable's value by before it is formatted.	MULTIPLIER= on page 523
Specify that numbers are message characters rather than digit selectors.	NOEDIT on page 523
Specify a character prefix for the formatted value.	PREFIX= on page 523

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, you use a dollar sign ($) as the first character, which is why a character informat is limited to 31 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 534:

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 525.

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 568

LANGUAGE=
specifies the language that is used for the days of the week and months of the year that can be substituted in a date, time, or datetime specification.

Default: English

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 can 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, SAS stores the variables in sorted order. This order can 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. The value of the MULTIPLIER= option depends both on the result of the multiplication and on the digit selectors in the label portion of the format. For example, the following PICTURE statement creates the MILLION. format, which formats the variable value 1600000 as **`$1.6M`**:

```
picture million low-high='09.9M'
       (prefix='$' mult=.00001);
```

Note that there is a digit selector after the decimal. The value 16 is placed into the "template" beginning on the right. The value 16 overlays 09.9, and results in 01.6. Leading zeroes are dropped, and the final result is 1.6M.

If the value of low-high is equal to '000M', then the result would be 16M.

Alias: MULT=

Default: 10^n, where n is the number of digits after the first decimal point in the picture. For example, suppose your data contains a value 123.456 and you want to print it using a picture of '999.999'. The format multiplies 123.456 by 10^3 to obtain a value of 123456, which results in a formatted value of **`123.456`**.

Featured in: Example 1 on page 547

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 547
- Example 9 on page 568

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 section uses 9's as nonzero digit selectors. △

Message characters are nonnumeric characters that print as specified in the picture. The following PICTURE statement contains both digit selectors (99) and message characters (**illegal day value**). Because the DAYS. format has nonzero digit selectors, values are printed with leading zeros. The special range OTHER prints the message characters for any values that do not fall into the specified range (1 through 31).

```
picture days 01-31='99'
        other='99-illegal day value';
```

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 use only directives when you specify the DATATYPE= option in the PICTURE statement.

The permitted directives are as follows:

%a
abbreviated weekday name, for example, Wed.

%A
full weekday name, for example, Wednesday.

%b
abbreviated month name, for example, Jan.

%B
full month name, for example, January.

%C
long month name with blank padding (January through December), for example, December.

%d
day of the month as a two-digit decimal number (01–31), for example, 02.

%e
day of the month as a two-character decimal number with leading spaces (" 1"-"31"), for example, “ 2”.

%F
full weekday name with blank padding.

%g
year as a two-digit decimal number (00 - 99), for example, 02. If the week that contains January 1 has four or more days in the new year, then it is considered week 1 in the new year. Otherwise, it is the last week of the previous year and the year is considered the previouse year.

%G
year as a four-digit decimal number, for example, 2008. If the week that contains January 1 has four or more days in the new year, then it is considered week 1 in the new year. Otherwise, it is the last week of the previous year and the year is considered the previouse year.

%H
hour (24-hour clock) as a two-digit decimal number (00–23), for example, 19.

%I
hour (12-hour clock) as a two-digit decimal number (01–12), for example, 05.

%j
day of the year as a decimal number (1–366), with leading zero.

%m
month as a two-digit decimal number (01–12), for example, 01.

%M
minute as a two-digit decimal number (00–59), for example, 45.

%o
month (1-12) with blank padding, for example, " 2".

%p

equivalent to either a.m. or p.m.

%q
abbreviated quarter of the year string such as Qtr1, Qtr2, Qtr3, or Qtr4.

%Q
quarter of the year string, such as Quarter1, Quarter2, Quarter3, or Quarter4.

%S
second as a two-digit decimal number (00–61) and allowing for possible leap seconds, for example, 58.

%u
weekday as a one-digit decimal number (1–7 (Monday - Sunday)), for example, Sunday=7.

%U
week number of the year as a decimal number (0,53) with leading 0. Sunday is considered the first day of the week.

%V
week number (01–53) with the first Monday as the start day of the first week. Minimum days of the first week is 4.

%w
weekday as a one-digit decimal number (0–6 (Sunday through Saturday)), for example Sunday=0.

%W
week number (00–53) with the first Monday as the start day of the first week.

%y
year without century as a two-digit decimal number (00–99), for example, 93.

%Y
year with century as a four-digit decimal number (1970–2069), for example, 1994.

%%
the % character.

Any directive that generates numbers can produce a leading zero, if desired, by adding a 0 before the directive. Adding a leading zero 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 536.

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-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 25.1 on page 529 provides an illustration of each step. The circled numbers in the figure correspond to the step numbers in the table.

Table 25.1 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.
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	Add a fuzz factor of 10e-8 to the number. If the ROUND option is not in effect, truncate the number after the decimal. If the ROUND option is in effect, then the format rounds the number after the decimal to the next highest integer if the number after the decimal is greater than or equal to .5. Note: The fuzzing factor is not related to the FUZZ= option. The FUZZ= option is used in matching ranges to values.	Because the example uses MULT= values that ensured that all of the significant digits were moved to the left of the decimal, no significant digits are lost. The zeros are truncated.
5	Turn the number into a character string. If the number is shorter than the picture, then the length of the character string is equal to the number of digit selectors in the picture. Pad the character string with leading zeros. (The results are equivalent to using the Z*w.* format. Z*w.* is explained in the section on SAS formats in *SAS 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 25.1 Formatting One Value in Each Range

		-2.051	-.05	.556	6.6
❶	range	low - -1	-1 <-< 0	0 -< 1	1 - high
❶	picture	00.00	99	99	00.00
❷	absolute value	2.051	.05	.556	6.6
❸	MULT=	2.051 X 10^2= 205.1	.05 X 100= 5.000	.556 X 100= 55.600	6.6 X 10^2= 660.000
❹	round	205	5	56	660
❺	character string	0205	05	56	0660
❻	template	2.05	05	56	6.60
❼	prefix	prefix = '-'	prefix = '-.'	prefix = '.'	none
	formatted result	-2.05	-.05	.56	6.60

The following PROC PRINT step associates the NOZEROS. format with the AMOUNT variable in SAMPLE. The output shows the result of rounding.

```
proc print data=sample noobs;
   format amount nozeros.;
   title 'Formatting the Variable Amount';
   title2 'with the NOZEROS. Format';
run;
```

```
                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 for 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 561.

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;
```

How the FMTLIB and CNTLOUT= Options Affect Whether a Catalog Is Opened in Read or Update Mode

Using the FMTLIB and CNTLOUT= options on the SELECT statement indicates whether a catalog is opened for read or update mode. The following rules apply:

- If you use the SELECT statement and do not specify the FMTLIB or the CNTLOUT= option, PROC FORMAT assumes that the catalog is opened in update mode.
- If you use the SELECT statement and specify the FMTLIB or the CNTLOUT= option, the catalog is opened for read access.
- If you use the SELECT statement without the FMTLIB or the CNTLOUT= option, and the SAS program does not have write access to the catalog, the following error is written to the SAS log:

```
ERROR: User does not have appropriate authorization level
       for file libref.FORMATS.CATALOG.
```

VALUE Statement

Creates a format that specifies character strings to use to print variable values.

See also: The section about formats in *SAS Language Reference: Dictionary* for documentation about SAS formats.

Featured in: Example 2 on page 549.

VALUE <$>*name* <(*format-option(s)*)>
<*value-range-set(s)*>;

Task	Option
Specify the default length of the format.	DEFAULT= on page 534
Specify a fuzz factor for matching values to a range.	FUZZ= on page 535
Specify a maximum length for the format.	MAX= on page 535
Specify a minimum length for the format.	MIN= on page 535
Specify multiple values for a given range, or for overlapping ranges.	MULTILABEL on page 532
Store values or ranges in the order that you define them.	NOTSORTED on page 535

Required Arguments

name
names the format that you are creating.

Restriction: The name of a user-defined format cannot be the same as the name of a format that is supplied by SAS.

Requirement: The name must be a valid SAS name. A numeric format name can be up to 32 characters in length. A character format name can be up to 31 characters in length. If you are creating a character format, then use a dollar sign ($) as the first character.

Restriction: Format names cannot end in a number.

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 534:

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 can also be assigned to a single range of internal values. The second VALUE statement assigns labels to overlapping ranges of internal values. The MULTILABEL option allows the assignment of multiple labels to the overlapped internal values.

```
value one (multilabel)
   1='ONE'
```

```
   1='UNO'
   1='UN';

value agefmt (multilabel)
   15-29='below 30 years'
   30-50='between 30 and 50'
   51-high='over 50 years'
   15-19='15 to 19'
   20-25='20 to 25'
   25-39='25 to 39'
   40-55='40 to 55'
   56-high='56 and above';
```

Only multilabel-enabled procedures such as PROC MEANS, PROC SUMMARY, and PROC TABULATE can use multiple labels. All other procedures and the data step recognize only the primary label. The *primary label* for a given entry is the external value that is assigned to the first internal value or range of internal values that matches or contains the entry when all internal values are ordered sequentially. 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 equal sign print as the character string on the right side of the equal sign.

formatted-value

specifies a character string that becomes the printed value of the variable value that appears on the left side of the equal sign. Formatted values are always character strings, regardless of whether you are creating a character or numeric format.

Formatted values can be up to 32,767 characters. For hexadecimal literals, you can use up to 32,767 typed characters, or up to 16,382 represented characters at 2 hexadecimal characters per represented character. Some procedures, however, use only the first 8 or 16 characters of a formatted value.

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 equal 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 536.

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 equal sign. If you have a quoted string to the left of the equal sign, then the default length is the length of the longest string.

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 one of the following is true:

- you know the likelihood of certain ranges occurring, and you want your informat or format to search those ranges first to save processing time.
- you want to preserve the order that you define ranges when you print a description of the informat or format using the FMTLIB option.
- you want to preserve the order that you define ranges when you use the ORDER=DATA option and the PRELOADFMT option to analyze class variables in PROC MEANS, PROC SUMMARY, or PROC TABULATE.

Do not use NOTSORTED if the distribution of values is uniform or unknown, or if the number of values is relatively small. The binary searching algorithm that

SAS uses when NOTSORTED is not specified optimizes the performance of the search under these conditions.

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 equal sign you specify the values that you want to convert to other values. On the right side of the equal sign, you specify the values that you want the values on the left side to become. This section discusses the different forms that you can use for *value-or-range*, which represents the values on the left side of the equal sign. For details about how to specify values for the right side of the equal sign, see the "Required Arguments" section for the appropriate statement.

The INVALUE, PICTURE, and VALUE statements accept numeric values on the left side of the equal sign. In character informats, numeric ranges are treated as character strings. INVALUE and VALUE also accept character strings on the left side of the equal sign.

As the syntax shows, you can have multiple occurrences of *value-or-range* in each *value-range-set*, with commas separating the occurrences. Each occurrence of *value-or-range* is either one of the following:

value
a single value, such as 12 or **'CA'**. For character formats and informats, enclose the character values in single quotation marks. If you omit the quotation marks around *value*, then PROC FORMAT assumes the quotation marks to be there.

You can use the keyword OTHER as a single value. OTHER matches all values that do not match any other value or range.

range
a list of values, for example, 12–68 or **'A'-'Z'**. For ranges with character strings, be sure to enclose each string in single quotation marks. For example, if you want a range that includes character strings from A to Z, then specify the range as **'A'-'Z'**, with single quotation marks around the **A** and around the **Z**

If you specify **'A-Z'**, then the procedure interprets it as a three-character string with **A** as the first character, a hyphen (-) as the second character, and a **Z** as the third character.

If you omit the quotation marks, then the procedure assumes quotation marks around each string. For example, if you specify the range **abc-zzz**, then the procedure interprets it as **'abc'-'zzz'**.

In numeric user-defined informats, the procedure interprets an unquoted numeric range on the left side of a *value-range-set* as a numeric range. In a character user-defined informat, the procedure interprets an unquoted numeric range on the left side of a **value-range-set** as a character string. For example, in a character informat, the range **12--86** is interpreted as **'12'--'86'**.

You can use LOW or HIGH as one value in a range, and you can use the range LOW-HIGH to encompass all values. For example, the following are valid ranges:

```
low-'ZZ'
35-high
low-high
```

You can use the less than (<) symbol to exclude values from ranges. If you are excluding the first value in a range, then put the < after the value. If you are excluding the last value in a range, then put the < before the value. For example, the following range does not include 0:

```
0<-100
```

Likewise, the following range does not include 100:

```
0-<100
```

If a value at the high end of one range also appears at the low end of another range, and you do not use the < noninclusion notation, then PROC FORMAT assigns the value to the first range. For example, in the following ranges, the value **AJ** is part of the first range:

```
'AA'-'AJ'=1 'AJ'-'AZ'=2
```

In this example, to include the value **AJ** in the second range, use the noninclusive notation on the first range:

```
'AA'-<'AJ'=1 'AJ'-'AZ'=2
```

If you overlap values in ranges, then PROC FORMAT returns an error message unless, for the VALUE statement, the MULTILABEL option is specified. For example, the following ranges will cause an error:

```
'AA'-'AK'=1 'AJ'-'AZ=2
```

Each *value-or-range* can be up to 32,767 characters. If *value-or-range* has more than 32,767 characters, then the procedure truncates the value after it processes the first 32,767 characters.

Note: You do not have to account for every value on the left side of the equal sign. Those values are converted using the default informat or format. For example, the following VALUE statement creates the TEMP. format, which prints all occurrences of 98.6 as **NORMAL**:

```
value temp 98.6='NORMAL';
```

If the value were 96.9, then the printed result would be **96.9**. △

Concepts: FORMAT Procedure

Associating Informats and Formats with Variables

Methods of Associating Informats and Formats with Variables

The following table summarizes the different methods for associating informats and formats with variables.

Table 25.2 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.

Differences between the FORMAT Statement and PROC FORMAT

Do not confuse the FORMAT statement with the FORMAT procedure. The FORMAT and INFORMAT statements associate an existing format or informat (either standard SAS or user-defined) with one or more variables. PROC FORMAT creates user-defined formats or informats.

Assigning Formats and Informats to a Variable

Assigning your own format or informat to a variable is a two-step process: creating the format or informat with the FORMAT procedure, and then assigning the format or informat with the FORMAT, INFORMAT, or ATTRIB statement.

It is often useful to assign informats in the FSEDIT procedure in SAS/FSP software and in the BUILD procedure in SAS/AF software.

For complete documentation on the ATTRIB, INFORMAT, and FORMAT statements, see the section about 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 26 for more information and examples of using formats in Base SAS procedures.

Storing Informats and Formats

Format Catalogs

PROC FORMAT stores user-defined informats and formats as entries in SAS catalogs.* You use the LIBRARY= option in the PROC FORMAT statement to specify the catalog. If you omit the LIBRARY= option, then formats and informats are stored in the WORK.FORMATS catalog. If you specify LIBRARY=*libref* but do not specify a catalog name, then formats and informats are stored in the *libref*.FORMATS catalog. Note that this use of a one-level name differs from the use of a one-level name elsewhere in SAS. With the LIBRARY= option, a one-level name indicates a library; elsewhere in SAS, a one-level name indicates a file in the WORK library.

The name of the catalog entry is the name of the format or informat. The entry types are as follows:

- FORMAT for numeric formats
- FORMATC for character formats
- INFMT for numeric informats
- INFMTC for character informats

Temporary Informats and Formats

Informats and formats are temporary when they are stored in a catalog in the WORK library. If you omit the LIBRARY= option, then PROC FORMAT stores the informats and formats in the temporary catalog WORK.FORMATS. You can retrieve temporary informats and formats only in the same SAS session or job in which they are created. To retrieve a temporary format or informat, simply include the name of the format or informat in the appropriate SAS statement. SAS automatically looks for the format or informat in the WORK.FORMATS catalog.

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 514.

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:

* Catalogs are a type of SAS file and reside in a SAS library. If you are unfamiliar with the types of SAS files or the SAS library structure, then see the section on SAS files in *SAS Language Reference: Concepts*.

- 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 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 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 566.

Missing Informats and Formats

If you reference an informat or format that SAS cannot find, then you receive an error message and processing stops unless the SAS system option NOFMTERR is in effect. When NOFMTERR is in effect, SAS uses the *w.* or $*w.* default format to print values for variables with formats that it cannot find. For example, to use NOFMTERR, use this OPTIONS statement:

```
options nofmterr;
```

Refer to the section on SAS system options in *SAS Language Reference: Dictionary* for more information on NOFMTERR.

Printing Informats and Formats

The output that is provided when you use the FMTLIB option is intended to present a brief view of the informat and format values.

Instead of using the FMTLIB option, you can use the CNTLOUT= option to create an output data set that stores information about informats and formats. You can then use

PROC PRINT or PROC REPORT to print the data set. In this case, labels are not truncated.

Note: You can use data set options to keep or drop references to additional variables that were added by using the CNTLOUT= option. △

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 as follows:

DEFAULT
specifies a numeric variable that indicates the default length for format or informat.

END
specifies a character variable that gives the range's ending value.

EEXCL
specifies a character variable that indicates whether the range's ending value is excluded. Valid values are as follows:

Y specifies that the range's ending value is excluded.

N specifies that the range's ending value is not excluded.

FILL
for picture formats, specifies a numeric variable whose value is the value of the FILL= option.

FMTNAME
specifies a character variable whose value is the format or informat name.

FUZZ
specifies a numeric variable whose value is the value of the FUZZ= option.

HLO

specifies a character variable that contains range information about the format or informat. The following valid values can appear in any combination:

F specifies a standard SAS format or informat that is used with a value.

H specifies that a range's ending value is HIGH.

I specifies a numeric informat range.

J specifies justification for an informat.

L specifies that a range's starting value is LOW.

M specifies that the MULTILABEL option is in effect.

N specifies that the format or informat has no ranges, including no OTHER= range.

O specifies that the range is OTHER.

R specifies that the ROUND option is in effect.

S specifies that the NOTSORTED option is in effect.

U specifies that the UPCASE option for an informat be used.

LABEL
specifies a character variable whose value is associated with a format or an informat.

LENGTH
specifies a numeric variable whose value is the value of the LENGTH= option.

MAX
specifies a numeric variable whose value is the value of the MAX= option.

MIN
specifies a numeric variable whose value is the value of the MIN= option.

MULT
specifies a numeric variable whose value is the value of the MULT= option.

NOEDIT
for picture formats, specifies a numeric variable whose value indicates whether the NOEDIT option is in effect. Valid values are as follows:

1 specifies that the NOEDIT option is in effect.

0 specifies that the NOEDIT option is not in effect.

PREFIX
for picture formats, specifies a character variable whose value is the value of the PREFIX= option.

SEXCL
specifies a character variable that indicates whether the range's starting value is excluded. Valid values are as follows:

Y specifies that the range's starting value is excluded.

N specifies that the range's starting value is not excluded.

START
specifies a character variable that gives the range's starting value.

TYPE
specifies a character variable that indicates the type of format. Possible values are as follows:

C	specifies a character format.
I	specifies a numeric informat.
J	specifies a character informat.
N	specifies a numeric format (excluding pictures).
P	specifies a picture format.

The following output shows an output control data set that contains information about all the informats and formats created in "Examples: FORMAT Procedure" on page 546.

Output 25.1 Output Control Data Set for PROC FORMAT Examples

An Output Control Data Set 1

Obs	FMTNAME	START	END	LABEL	MIN	MAX	DEFAULT	LENGTH	FUZZ	MULT	FILL	LABEL	DECSEP	DIG3SEP	DATAWIDTH	MULTILABEL	NOEDIT
1	BENEFIT	LOW	7304	WORDDATE20.	1	40	20	20	1E-12		0.00	0	N	N	N	L	F
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 530 and "EXCLUDE Statement" on page 516.

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 541. 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 557.

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 530 and "EXCLUDE Statement" on page 516. For an example, see Example 6 on page 561.

The FMTLIB output shown in the following output contains a description of the NOZEROS. format, which is created in "Building a Picture Format: Step by Step" on page 526, and the EVAL. informat, which is created in Example 4 on page 554.

Output 25.2 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 equal sign.
- For numeric informats
  - LENGTH is 12 if all values on the left side of the equal sign are numeric.
  - LENGTH is the same as the longest raw data value on the left side of the equal sign.
- For formats, the value for LENGTH is the length of the longest value on the right side of the equal sign.

In the output for @EVAL., the length is 1 because 1 is the length of the longest raw data value on the left side of the equal 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
appears only for informats and contains the values that have informats.

LABEL
LABEL appears only for formats and contains either the formatted value or picture. The SAS version number and the date on which the format or informat was created are in parentheses after INVALUE or LABEL.

Note: If SAS displays version numbers V7|V8, then the format is compatible with those versions. If it is not compatible with those earlier releases, the release that created the format is shown. Version V9 supports long format and informat names (over eight characters), and V7|V8 do not. △

For picture formats, such as NOZEROS., the LABEL section contains the PREFIX=, FILL=, and MULT= values. To note these values, FMTLIB prints the letters **P**, **F**, and **M** to represent each option, followed by the value. For example, in the LABEL section, **P-.** indicates that the prefix value is a 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 541 contains a description of these informats and the formats.

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

Create the data set PROCLIB.STAFF. The INPUT statement assigns the names Name, IdNumber, Salary, Site, and HireDate to the variables that appear after the DATALINES statement. The FORMAT statement assigns the standard SAS format DATE7. to the variable HireDate.

```
data proclib.staff;
   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 546.

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-library-1 ';
libname library 'SAS-library-2';
```

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=40;
```

Specify that user-defined formats will be stored in the catalog LIBRARY.FORMATS. The LIBRARY= option specifies a SAS catalog that will contain the formats or informats that you create with PROC FORMAT. When you create the library named LIBRARY, SAS automatically creates a catalog named FORMATS inside LIBRARY.

```
proc format library=library;
```

Define the USCurrency. picture format. The PICTURE statement creates a template for printing numbers. LOW-HIGH ensures that all values are included in the range. The MULT= statement option specifies that each value is multiplied by 1.61. The PREFIX= statement adds a US dollar sign to any number that you format. The picture contains six digit selectors, five for the salary and one for the dollar sign prefix.

```
   picture uscurrency low-high='000,000' (mult=1.61 prefix='$');
run;
```

Print the PROCLIB.STAFF data set. The NOOBS option suppresses the printing of observation numbers. The LABEL option uses variable labels instead of variable names for column headings.

```
proc print data=proclib.staff noobs label;
```

Specify a label and format for the Salary variable. The LABEL statement substitutes the specific label for the variable in the report. In this case, "Salary in US Dollars" is substituted for the variable Salary for this print job only. The FORMAT statement associates the USCurrency. format with the variable name Salary for the duration of this procedure step.

```
label salary='Salary in U.S. Dollars';
format salary uscurrency.;
```

Specify the title.

```
   title 'PROCLIB.STAFF with a Format for the Variable Salary';
run;
```

Output: Listing

```
          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
```

Output: HTML

PROCLIB.STAFF with a Format for the Variable Salary

Name	IdNumber	Salary in U.S. Dollars	Site	HireDate
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 546.

Format: USCurrency. on page 548

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-library-1';
libname library 'SAS-library-2';
```

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=40;
```

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 547) 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: Listing

```
                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
```

Output: HTML

PROCLIB.STAFF with a Format for the Variables
Salary and Site

Name	IdNumber	Salary in U.S. Dollars	Site	HireDate
Capalleti, Jimmy	2355	$34,072	BIRMINGHAM UK	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 546.

Formats:
USCurrency. on page 548 and $CITY. on page 550.

This example uses an existing format that is supplied by SAS as a formatted value. Tasks include the following:

- creating a numeric format
- nesting formats
- writing a format using a standard SAS format
- formatting dates

Program

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 before 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-library-1';
libname library 'SAS-library-2';
```

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=40;
```

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;
```

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 547) with SALARY, the $CITY. format (created in Example 2 on page 549) 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;
```

* 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*.

Output: Listing

```
                  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 **
```

Output: HTML

PROCLIB.STAFF with a Format for the Variables
Salary, Site, and HireDate

Name	IdNumber	Salary in U.S. Dollars	Site	HireDate
Capalleti, Jimmy	2355	$34,072	BIRMINGHAM UK	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-library-1';
libname library 'SAS-library-2';
```

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=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.

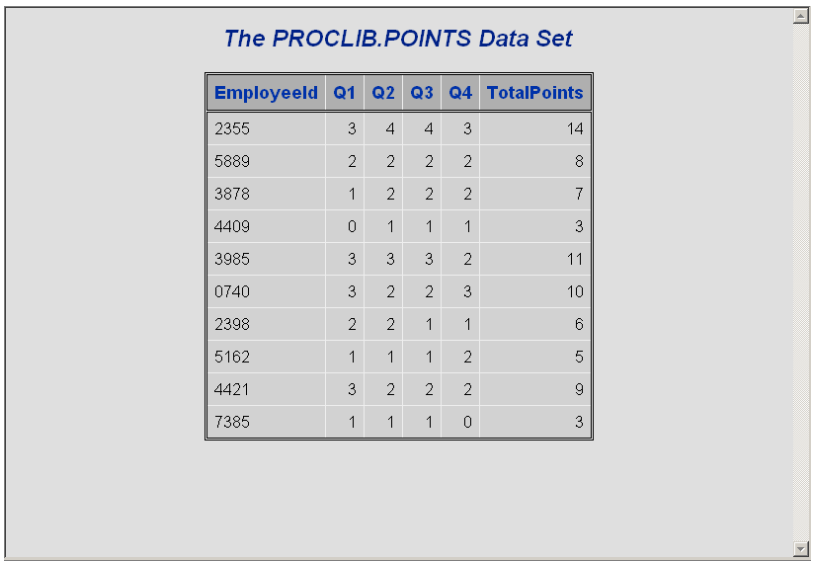
```
   title 'The PROCLIB.POINTS Data Set';
run;
```

Output: Listing

The PROCLIB.POINTS Data Set 1

Employee Id	Q1	Q2	Q3	Q4	Total 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

Output: HTML



The PROCLIB.POINTS Data Set

EmployeeId	Q1	Q2	Q3	Q4	TotalPoints
2355	3	4	4	3	14
5889	2	2	2	2	8
3878	1	2	2	2	7
4409	0	1	1	1	3
3985	3	3	3	2	11
0740	3	2	2	3	10
2398	2	2	1	1	6
5162	1	1	1	2	5
4421	3	2	2	2	9
7385	1	1	1	0	3

Example 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 the following:

- 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 51, "The REPORT Procedure," on page 979 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: Listing

```
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%
```

Output: HTML

The Percentage of Salary for Calculating Bonus

EmployeeId	Total Points	Percentage
2355	14	10%
5889	.	***ERROR***
3878	7	6%
4409	3	0%
3985	11	10%
0740	10	8%
2398	.	***ERROR***
5162	5	3%
4421	9	8%
7385	3	0%

Example 6: Printing the Description of Informats and Formats

Procedure features:

PROC FORMAT statement option:

FMTLIB

SELECT statement

Format:

NOZEROS on page 527.

Informat:

Evaluation. on page 555

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-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.

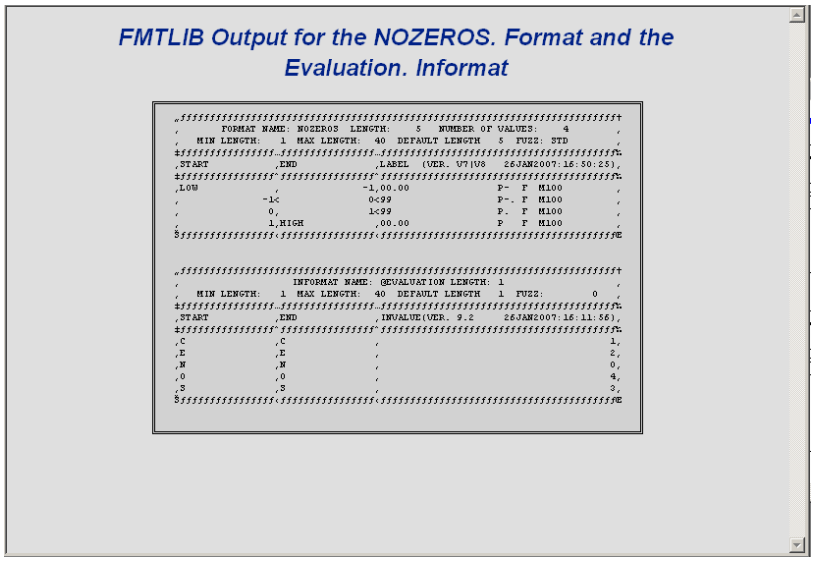
```
    title 'FMTLIB Output for the NOZEROS. Format and the';
    title2 'Evaluation. Informat';
run;
```

Output: Listing

The output is described in "Procedure Output" on page 544.

```
                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. 9.2     26JAN2007:16:11:56)|
|----------------+----------------+-----------------------------------------|
|C               |C               |                                        1|
|E               |E               |                                        2|
|N               |N               |                                        0|
|O               |O               |                                        4|
|S               |S               |                                        3|
-------------------------------------------------------------------------------
```

Output: HTML



Example 7: Retrieving a Permanent Format

Procedure features:

PROC FORMAT statement options:

LIBRARY=

Other features:

FMTSEARCH= system option

Data sets:

SAMPLE on page 526.

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-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 526.

```
   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);
```

* For complete documentation on the FMTSEARCH= system option, see the section on SAS system options in *SAS Language Reference: Dictionary*.

Print the SAMPLE data set. The FORMAT statement associates the NOZEROS. format with the Amount variable.

```
proc print data=sample;
   format amount nozeros.;
```

Specify the titles.

```
   title1 'Retrieving the NOZEROS. Format from PROCLIB.FORMATS';
   title2 'The SAMPLE Data Set';
run;
```

Output: Listing

```
     Retrieving the NOZEROS. Format from PROCLIB.FORMATS      1
                     The SAMPLE Data Set

                        Obs     Amount

                         1      -2.05
                         2       -.05
                         3       -.02
                         4        .00
                         5        .09
                         6        .54
                         7        .56
                         8       6.60
                         9      14.63
```

Output: HTML

Retrieving the NOZEROS. Format from PROCLIB.FORMATS
The SAMPLE Data Set

Obs	Amount
1	-2.05
2	-.05
3	-.02
4	.00
5	.09
6	.54
7	.56
8	6.60
9	14.63

Example 8: Writing Ranges for Character Strings

Data sets:
PROCLIB.STAFF on page 546.

This example creates a format and shows how to use ranges with character strings.

Program

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

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=40;
```

Create the TRAIN data set from the PROCLIB.STAFF data set. PROCLIB.STAFF was created in "Examples: FORMAT Procedure" on page 546.

```
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;
```

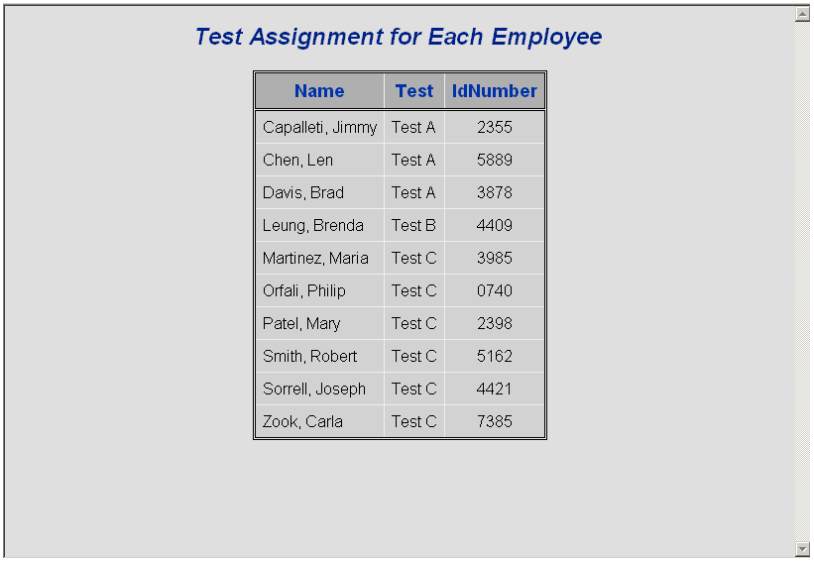
Generate a report of the TRAIN data set. The FORMAT= option in the DEFINE statement associates $SkillTest. with the NAME variable. The column that contains the formatted values of NAME is using the alias Test. Using an alias enables you to print a variable twice, once with a format and once with the default format. See Chapter 51, "The REPORT Procedure," on page 979for 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: Listing

```
                    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
```

Output: HTML



Test Assignment for Each Employee

Name	Test	IdNumber
Capalleti, Jimmy	Test A	2355
Chen, Len	Test A	5889
Davis, Brad	Test A	3878
Leung, Brenda	Test B	4409
Martinez, Maria	Test C	3985
Orfali, Philip	Test C	0740
Patel, Mary	Test C	2398
Smith, Robert	Test C	5162
Sorrell, Joseph	Test C	4421
Zook, Carla	Test C	7385

Example 9: Filling a Picture Format

Procedure features:

PICTURE statement options:

FILL=
PREFIX=

This example does the following:

- prefixes the formatted value with a specified character
- fills the leading blanks with a specified character

- shows the interaction between the FILL= and PREFIX= options

Program

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.;
```

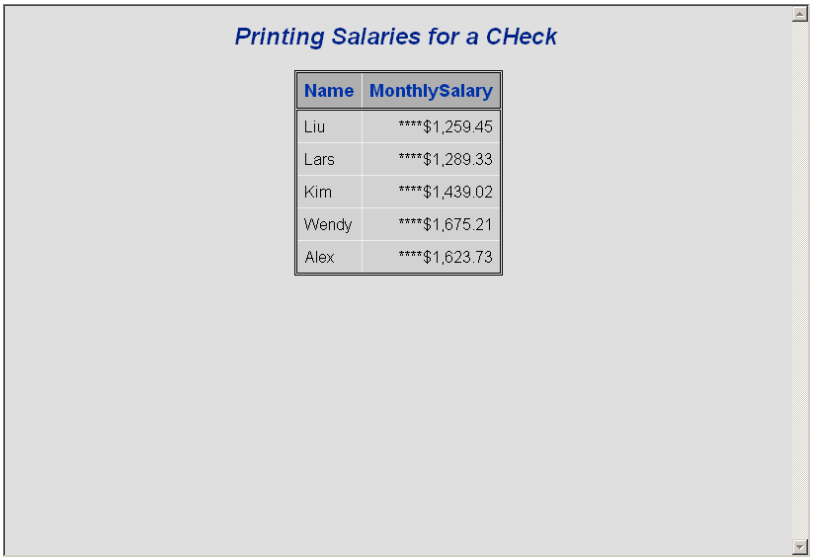
Specify the title.

```
   title 'Printing Salaries for a Check';
run;
```

Output: Listing

```
                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
```

Output: HTML



Printing Salaries for a CHeck

Name	MonthlySalary
Liu	****$1,259.45
Lars	****$1,289.33
Kim	****$1,439.02
Wendy	****$1,675.21
Alex	****$1,623.73

Example 10: Creating a Format in a non-English Language

Procedure features:

PICTURE statement options:

DATATYPE=
LANGUAGE=

This example does the following:

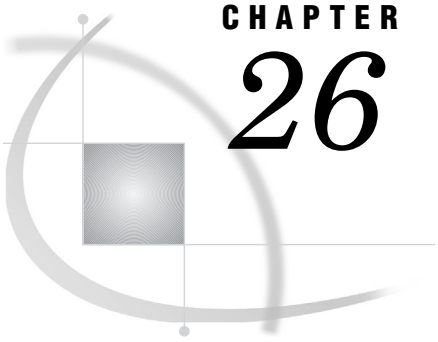
- specifies directives for formatting date, time, and datetime values by using the DATATYPE= statement option
- specifies a language for the format by using the LANGUAGE= statement option
- creates the ALLTEST format

Program

```
proc format;
   picture mdy(default=8) other='%0m%0d%Y' (datatype=date);
   picture langtsda (default=50) other='%A, %B,%d, %Y' (datatype=date);
   picture langtsdt (default=50) other='%A, %B,%d, %Y' (datatype=datetime);
   picture langtsfr (default=50) other='%A, %B,%d, %Y' (datatype=datetime

   picture alltest (default=100)
               other='%a %A %b %B %d %H %I %j %m %M %p %S
                           %w %U %y %%' (datatype=datetime);
run;
```

See Also

FMTSEARCH= System option
VALIDFMTNAME= System option
FORMAT Statement



CHAPTER

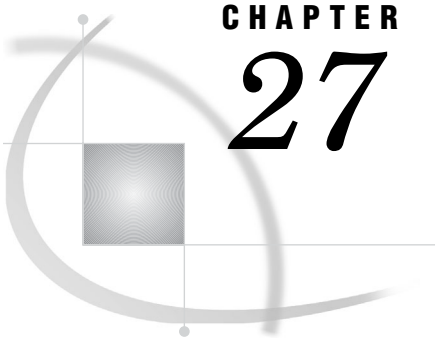
26

The FORMS Procedure

Information about the FORMS Procedure **573**

Information about the FORMS Procedure

See: For documentation about the FORMS procedure, go to **http://support.sas.com/documentation/onlinedoc/base/91/forms.pdf**.



CHAPTER
27

The FREQ Procedure

Information about the FREQ Procedure **575**

Information about the FREQ Procedure

See: The documentation for the FREQ procedure has moved to the *Base SAS Procedures Guide: Statistical Procedures*.

CHAPTER 28

The FSLIST Procedure

Overview: FSLIST Procedure **577**
Syntax: FSLIST Procedure **577**
Statement Descriptions **578**
PROC FSLIST Statement **578**
FSLIST Command **580**
Using the FSLIST Window **582**
General Information about the FSLIST Window **582**
FSLIST Window Commands **582**
Global Commands **582**
Scrolling Commands **583**
Searching Commands **585**
Display Commands **587**
Other Commands **588**

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)*>;

Task	Statement
Initiate the FSLIST procedure and specify the external file to browse.	"PROC FSLIST Statement" on page 578

- 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 the following:

```
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 in effect:

OVP	causes the procedure to honor the overprint code and print the current line over the previous line when the code is encountered.
NOOVP	causes the procedure to ignore the overprint code and print each line from the file on a separate line of the display.

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 box in which you can specify the name of the file to browse, along with various FSLIST procedure options. In the dialog box, you can specify either a physical filename, a fileref, or a directory name. If you specify a directory name, then a selection list of the files in the directory appears, from which you can choose the desired file.

?

opens a selection window from which you can choose the external file to browse. The selection list in the window includes all external files that are identified in the current SAS session (all files with defined filerefs).

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 579 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 582 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

29

The HTTP Procedure

Overview: HTTP Procedure **589**
What Does the Hypertext Transfer Protocol (HTTP) Procedure Do? **589**
Syntax: HTTP Procedure **589**
PROC HTTP Statement **590**
Using Hypertext Transfer Protocol Secure (HTTPS) **591**
HTTP Security: SSL and Data Encryption **591**
Making PROC HTTP Calls by Using the HTTPS Protocol **592**
Examples: HTTP Procedure **592**
Example 1: A Simple POST Request **592**
Example 2: A POST Through a Proxy **593**
Example 3: A POST Through a Proxy That Requires Authentication **593**
Example 4: A POST That Captures the Response Headers **594**

Overview: HTTP Procedure

What Does the Hypertext Transfer Protocol (HTTP) Procedure Do?

PROC HTTP issues HTTP requests. PROC HTTP reads as input the entire body from a fileref and writes output to a fileref. PROC HTTP can also read custom request headers from a fileref and write response headers to a fileref.

Syntax: HTTP Procedure

PROC HTTP *options*;

Task	Statement
Issues HTTP requests.	“PROC HTTP Statement” on page 590

PROC HTTP Statement

Invokes a Web service that issues requests.

PROC HTTP *options*;

Task	Option
Specify the HTTP content-type to be set in the request headers.	CT on page 591
Specify a fileref to a text file that contains one line per request header in the format *key:value*. (In the z/OS operating environment, headerin files must be created with a variable record length.)	HEADERIN on page 591
Specify a fileref to a text file to which the response headers are in the format *key:value*.	HEADEROUT on page 591
Specify a fileref to input data.	IN on page 591
Specify the HTTP method.	METHOD on page 591
Specify a fileref where output is written.	OUT on page 591
Specify an HTTP proxy sever host name.	PROXYHOST on page 591
Specify an HTTP proxy server password. The password is required only if your proxy server requires credentials. Encodings that are produced by PROC PWENCODE are supported.	PROXYPASSWORD on page 591
Specify an HTTP proxy server port.	PROXYPORT on page 591
Specify an HTTP proxy server user name. The user name is required only if your proxy server requires credentials.	PROXYUSERNAME on page 591
Specify the endpoint for the HTTP request.	URL on page 591
Specify the Web authentication domain. If specified, a user name and password is retrieved from metadata for the specified authentication domain.	WEBAUTHDOMAIN on page 591
Specify a password for basic authentication. Encodings that are produced by PROC PWENCODE are supported.	WEBPASSWORD on page 591
Specify a user name for basic authentication.	WEBUSERNAME on page 591

Options

CT
specifies the HTTP content-type to be set in the request headers.

HEADERIN
Specify a fileref to a text file that contains one line per request header in the format *key:value*. (In the z/OS operating environment, headerin files must be created with a variable record length.)

HEADEROUT
specifies a fileref to a text file to which the response headers are in the format *key:value*.

IN
specifies a fileref to input data.

METHOD
specifies the HTTP method.

OUT
specifies a fileref where output is written.

PROXYHOST
specifies an HTTP proxy server host name.

PROXYPASSWORD
specifies an HTTP proxy server password. The password is required only if your proxy server requires credentials. Encodings that are produced by PROC PWENCODE are supported.

PROXYPORT
specifies an HTTP proxy server port.

PROXYUSERNAME
specifies an HTTP proxy server user name. The user name is required only if your proxy server requires credentials.

URL
specifies the endpoint for the HTTP request.

WEBAUTHDOMAIN
Specify the Web authentication domain. If specified, a user name and password is retrieved from metadata for the specified authentication domain.

WEBPASSWORD
Specify a password for basic authentication. Encodings that are produced by PROC PWENCODE are supported.

WEBUSERNAME
specifies a user name for basic authentication.

Using Hypertext Transfer Protocol Secure (HTTPS)

HTTP Security: SSL and Data Encryption

SSL enables Web browsers and Web servers to communicate over a secured connection by encrypting data. Both browsers and servers encrypt data before the data

is transmitted. The receiving browser or server then decrypts the data before it is processed.

Making PROC HTTP Calls by Using the HTTPS Protocol

In order to make PROC HTTP calls by using the HTTPS protocol, you must configure a trust source that contains the certificate of the service to be trusted. This trust source and its password must be provided for the SAS session by setting Java system options using jreoptions. You can provide this information on the SAS command line or in a SAS configuration file. Use the following syntax. Be sure to enter the following entry on one line:

```
-jreoptions (-Djavax.net.ssl.trustStore=full-path-to-the-trust-store -
Djavax.net.ssl.trustStorePassword=trustStorePassword)
```

The following example shows how to use the entry on the SAS command line. This example uses the Windows operating environment. Be sure to enter the following entry on one line:

```
"C:\Program Files\SAS\SASFoundation\9.2\sas.exe" -CONFIG
"C:\Program Files\SAS\SASFoundation\9.2\nls\en\SASV9.CFG"
-jreoptions (- Djavax.net.ssl.trustStore=C:\Documents and Settings\mydir\.keystore
-Djavax.net.ssl.trustStorePassword=trustpassword)
```

Examples: HTTP Procedure

Example 1: A Simple POST Request

This example makes a POST method call to a server on the local network. Parameters to the POST are read from IN and the response is written to OUT.TXT.

```
filename in "u:\prochttp\Testware\in";
filename out "u:\prochttp\Testware\out.txt";
data _null_;
   file in;
   input;
   put _infile_;
   datalines4;



server=localhost&action=list&type=store&user=your-user-name&password
=your-password&details=true&x=1


;;;;

 proc http in=in out=out url="http://localhost.com/rsm/rstool"
     method="post"
```

```
      ct="application/x-www-form-urlencoded";
run;
```

Example 2: A POST Through a Proxy

This example makes a POST method call to an external server and, therefore, requires the use of a proxy server. Parameters to the POST are read from ProxyTest_in and the response is written to ProxyTest_out.txt.

```
filename in "u:\prochttp\Testware\ProxyTest_in";
filename out "u:\prochttp\Testware\ProxyTest_out.txt";
data _null_;
   file in;
   input;
   put _infile_;
   datalines4;
appid=restbook&query=jellyfish
;;;;

proc http
   in=in
   out=out
   url="http://api.search.yahoo.com/WebSearchService/V1/webSearch?"
   method="post"
   ct="application/x-www-form-urlencoded"
   proxyhost="proxygw.abc.sas.com"
   proxyport=80;
run;
```

Example 3: A POST Through a Proxy That Requires Authentication

This example makes the same POST request as in Example 2 on page 593 but uses a proxy server that requires authentication.

```
filename in "u:\prochttp\Testware\ProxyAuthTest_in";
filename out "u:\prochttp\Testware\ProxyAuthTest_out.txt";
data _null_;
   file in;
   input;
   put _infile_;
   datalines4;
appid=restbook&query=jellyfish
;;;;

proc http
   in=in
   out=out
   url="http://api.search.yahoo.com/WebSearchService/V1/webSearch?"
   method="post"
   ct="application/x-www-form-urlencoded"
```

```
   proxyhost="proxyhost.company.com"
   proxyusername="your-user-name"
   proxypassword="your-password"
   proxyport=80;
run;
```

Example 4: A POST That Captures the Response Headers

This example makes the same POST request as in Example 2 on page 593 but captures the response headers in a file called headerOut.txt.

```
filename in "u:\prochttp\Testware\ProxyTest_in";
filename out "u:\prochttp\Testware\ProxyTest_out.txt";
filename hdrout "u:\prochttp\Testware\headerOut.txt";
data _null_;
   file in;
   input;
   put _infile_;
   datalines4;
appid=restbook&query=jellyfish
;;;;

 proc http
    in=in
    out=out
    headerout=hdrout
    url="http://api.search.yahoo.com/WebSearchService/V1/webSearch?"
    method="post"
    ct="application/x-www-form-urlencoded"
    proxyhost="inetgw.unx.sas.com"
    proxyport=80;
run;
```

CHAPTER
30

The IMPORT Procedure

Overview: IMPORT Procedure **595**
Syntax: IMPORT Procedure **596**
PROC IMPORT Statement **596**
Data Source Statements **597**
Examples: IMPORT Procedure **599**
Example 1: Importing a Delimited External File **599**
Example 2: Importing a Specific Delimited File Using a Fileref **602**
Example 3: Importing a Tab-Delimited File **603**
Example 4: Importing a Comma-Delimited File with a CSV Extension **604**

Overview: IMPORT Procedure

PROC IMPORT reads data from an external data source and writes it to a SAS data set. Base SAS can import delimited files. In delimited files, a delimiter—such as a blank, comma, or tab—separates columns of data values. If you license SAS/ACCESS Interface to PC Files, additional external data sources can include such files as Microsoft Access Database, Excel files, and Lotus spreadsheets. See the SAS/ACCESS Interface to PC Files for more information.

When you run the IMPORT procedure, it reads the input file and writes the data to a SAS data set. The SAS variable definitions are based on the input records. When the IMPORT procedure reads a delimited file, it generates a DATA step code to import the data.

You control the results with options and statements that are specific to the input data source. The IMPORT procedure generates the specified output SAS data set and writes information about the import to the SAS log. The log displays the DATA step code that the IMPORT procedure generates.

To import data, you can also use the Import Wizard or the External File Interface (EFI) to guide you through the steps to import an external data source. You can use the Import Wizard to generate IMPORT procedure statements, which you can save to a file for subsequent use. To open the Import Wizard or EFI from the SAS windowing environment, select **File ▸ Import Data**. For more information about the Import Wizard or EFI, see the Base SAS online Help and documentation.

Syntax: IMPORT Procedure

Restriction: PROC IMPORT is available for the following operating environments:

- Windows
- OpenVMS for Integrity servers
- UNIX

Table of Contents:

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: "Examples: IMPORT Procedure" on page 599

PROC IMPORT
DATAFILE="*filename*"
OUT=<*libref.*>*SAS data-set* <(*SAS data-set-options*)>
<DBMS=*identifier*><REPLACE> ;

Required Arguments

DATAFILE="*filename*"
specifies the complete path and filename or fileref for the input PC file, spreadsheet, or delimited external file. A fileref is a SAS name that is associated with the physical location of the output file. To assign a fileref, use the FILENAME statement. If you specify a fileref or if the complete path and filename does not include special characters such as the backslash in a path, lowercase characters, or spaces, then you can omit the quotation marks. For more information about the FILENAME statement, see *SAS Language Reference: Dictionary*. For more information about PC file formats, see *SAS/ACCESS Interface to PC Files: Reference*.

Restriction: The IMPORT procedure does not support device types or access methods for the FILENAME statement except for DISK. For example, the IMPORT procedure does not support the TEMP device type, which creates a temporary external file.

Restriction: When you use a fileref to specify a delimited file to import, the logical record length (LRECL) defaults to 256 unless you specify the LRECL in the filename statement. The maximum LRECL width that the IMPORT procedure supports is 32767.

Restriction: The IMPORT procedure can import data only if SAS supports the data type. SAS supports numeric and character types of data but not (for example,

binary objects). If the data that you want to import is a type that SAS does not support, the IMPORT procedure might not be able to import it correctly. In many cases, the procedure attempts to convert the data to the best of its ability. However, conversion is not possible for some types.

Interaction: For delimited files, the first 20 rows are scanned to determine the variable attributes. You can increase the number of rows scanned by using the GUESSINGROWS data source statement. For more information, see "Data Source Statements" on page 597. All values are read in as character strings. If a Date and Time format or numeric informat can be applied to the data value, the type will be declared as numeric. Otherwise, the type remains character.

Featured in: Example 1 on page 599, Example 2 on page 602, Example 3 on page 603, and Example 4 on page 604

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

identifies the output SAS data set with either a one or two-level SAS name (library and member name). If the specified SAS data set does not exist, the IMPORT procedure creates it. If you specify a one-level name, by default the IMPORT procedure uses either the USER library (if assigned) or the WORK library (if USER is not assigned).

Featured in: "Examples: IMPORT Procedure" on page 599

(*SAS-data-set-options*)

specifies SAS data set options. For example, to assign a password to the resulting SAS data set, you can use the ALTER=, PW=, READ=, or WRITE= data set option. To import only data that meets a specified condition, you can use the WHERE= data set option. For information about SAS data set options, see "Data Set Options" in SAS Language Reference: Dictionary *SAS Language Reference: Dictionary*.

Restriction: You cannot specify data set options when importing delimited, comma-separated, or tab-delimited external files.

Options

DBMS=*identifier*

specifies the type of data to import. Valid identifiers for delimited files are CSV, DLM, JMP, and CSV. To import a tab-delimited file, specify TAB as the identifier. To import any other delimited file that does not end in .CSV, specify DLM as the identifier. For a comma-separated file with a .CSV extension, DBMS= is optional. The IMPORT procedure recognizes .CSV as an extension for a comma-separated file.

Featured in: Example 1 on page 599 and Example 3 on page 603.

REPLACE

overwrites an existing SAS data set. If you do not specify the REPLACE option, the IMPORT procedure does not overwrite an existing data set.

Featured in: Example 1 on page 599.

Data Source Statements

Featured in: "Examples: IMPORT Procedure" on page 599

Delimited files have these valid data source statements:

- DATAROW=
- DLM=
- GETNAMES=
- GUESSINGGROWS=

When you license SAS/ACCESS Interface to PC Files, you have access to a greater number of statements and data types.

For more information about optional statements for delimited files, see the IMPORT and EXPORT Procedure Statements for Delimited Files in *SAS/ACCESS for Relational Databases: Reference*.

Statements for PC Files, Spreadsheets, or Delimited Files

Table 30.1 Statements for Importing Delimited Files

Data Source	Supported Syntax	Valid Values	Default Value
CSV	DATAROW=	1 to 32767	2 *
	GETNAMES=	YES|NO	YES
	GUESSINGROWS=	1 to 32767	20
DLM	DATAROW=	1 to 32767	2 *
	DELIMITER=	'Char'|'nn'x	space
	GETNAMES=	YES|NO	YES
	GUESSINGROWS=	1 to 32767	20
TAB	DATAROWS=	1 to 32767	2 *

[1] The default value of the DATAROW option is dependent upon the specification of the GETNAMES option. For more information, see the DATAROW statement below.

DATAROW
starts reading data from the specified row number in the delimited text file.

Valid values: 1 to 32767

Default: when GETNAMES=NO: 1 when GETNAMES=YES: 2

Restrictions: When GETNAMES=NO, DATAROW must be equal to or greater than 1.
When GETNAMES=YES, DATAROW must be equal to or greater than 2.

See also: GETNAMES

DELIMITER
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=, the IMPORT procedure assumes that the delimiter is a space.

Valid values: *char*|*nn*|*x*|*space*

Default: *space*

GETNAMES
specifies whether the IMPORT procedure generate SAS variable names from the data values in the first record in the input file.

Valid values: YES|NO

Restriction: If the column names in the first record in the input file are not valid SAS names, then the IMPORT procedure uses default variable names.

Note: If a data value in the first record in the input file is read and it contains special characters that are not valid in a SAS name, such as a blank, then SAS converts the character to an underscore. For example, the column name **`Occupancy Code`** would become the SAS variable name **`Occupancy_Code`**.

GUESSINGROWS
specifies the number of rows of the file to scan to determine the appropriate data type and length for the columns. The scan data process scans from row 1 to the number that is specified by the GUESSINGROWS option.

Valid values: 1 to 32767

Default: 20

Note: you can change the default value in the SAS Registry under **SAS Registry ▶ Products ▶ Base ▶ EFI ▶ GuessingRows**.

Restriction: the values should be greater than the value specified for DATAROW.

Examples: IMPORT Procedure

Example 1: Importing a Delimited External File

Procedure features:
The IMPORT procedure 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

```
proc import datafile="C:\My Documents\myfiles\delimiter.txt" out=mydata   dbms=dlm    replace
delimiter='&';
 getnames=yes;
run;
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, the IMPORT procedure 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 DATA step 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:
      Filename=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 the IMPORT procedure from the delimited external file.

The SAS System

Obs	Region	State	Month	Expenses	Revenue
1	Southern	GA	JAN2001	2000	8000
2	Southern	GA	FEB2001	1200	6000
3	Southern	FL	FEB2001	8500	11000
4	Northern	NY	FEB2001	3000	4000
5	Northern	NY	MAR2001	6000	5000
6	Southern	FL	MAR2001	9800	13500
7	Northern	MA	MAR2001	1500	1000

Example 2: Importing a Specific Delimited File Using a Fileref

Procedure features:

The IMPORT procedure statement arguments:

DATAFILE=
DBMS=
OUT=
REPLACE

Data source statements:

GETNAMES=

Other features:

FILENAME statement
PRINT procedure

This example imports the following space-delimited file and creates a temporary SAS data set named WORK.STATES.

```
Region State Capital Bird
South Georgia Atlanta ''Brown Thrasher''
South ''North Carolina'' Raleigh Cardinal
North Connecticut Hartford Robin
West Washington Olympia ''American Goldfinch''
Midwest Illinois Springfield Cardinal
```

Program

```
Filename stdata ''c:\temp\state_data.txt'' lrecl=100;

proc import datafile=stdata
     out=stateinfo
     dbms=dlm
     replace;
     getnames=yes;
run;
```

```
proc print;
run;


ods html file=''statedata.html'';
proc print;
run;
ods html close;
```

Example 3: Importing a Tab-Delimited File

Procedure features:

The IMPORT procedure statement arguments:

DATAFILE=
DBMS=
OUT=
REPLACE

Data source statements:

DATAROW=
DELIMITER=

Other features:

PRINT procedure

This example imports the following tab-delimited file and creates a temporary SAS data set named WORK.CLASS. On an ASCII platform, the hexadecimal representation of a tab is '09'x. On an EBCDIC platform, the hexadecimal representation of a tab is a '05'x. The first observation read will be observation 5 due to the DATAROW= specification. GETNAMES= defaults to 'yes'.

```
Name     Gender   Age
Joyce    F        11
Thomas   M        11
Jane     F        12
Louise   F        12
James    M        12
John     M        12
Robert   M        12
Alice    F        13
Barbara  F        13
Jeffery  M        13
Carol    F        14
Judy     F        14
Alfred   M        14
Henry    M        14
Jenet    F        15
Mary     F        15
Ronald   M        15
William  M        15
Philip   M        16
```

Program

```
proc import datafile='c:\temp\tab.txt'
     out=class
     dbms=dlm
     replace;
     delimiter='09'x;
     datarow=5;
run;
```

Example 4: Importing a Comma-Delimited File with a CSV Extension

Procedure features:

The IMPORT procedure statement arguments:

DATAFILE=
DBMS=
OUT=
REPLACE

Data source statements:

GETNAMES=

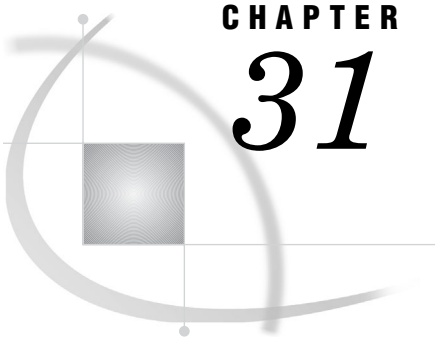
Other features: PRINT procedure

This example imports the following comma-delimited file and creates a temporary SAS data set named WORK.SHOES. GETNAME= is set to 'no', so the variable names in record 1 are not used. DATAROW=2 begins reading data from record 2.

```
''Africa'',''Boot'',''Addis Ababa'',''12'',''$29,761'',''$191,821'',''$769''
''Asia'',''Boot'',''Bangkok'',''1'',''$1,996'',''$9,576'',''$80''
''Canada'',''Boot'',''Calgary'',''8'',''$17,720'',''$63,280'',''$472''
''Central America/Caribbean'',''Boot'',''Kingston'',''33'',''$102,372'',''$393,376'',''$4,454
''Eastern Europe'',''Boot'',''Budapest'',''22'',''$74,102'',''$317,515'',''$3,341''
''Middle East'',''Boot'',''Al-Khobar'',''10'',''$15,062'',''$44,658'',''$765''
''Pacific'',''Boot'',''Auckland'',''12'',''$20,141'',''$97,919'',''$962''
''South America'',''Boot'',''Bogota'',''19'',''$15,312'',''$35,805'',''$1,229''
''United States'',''Boot'',''Chicago'',''16'',''$82,483'',''$305,061'',''$3,735''
''Western Europe'',''Boot'',''Copenhagen'',''2'',''$1,663'',''$4,657'',''$129''
```

Program

```
proc import datafile=''C:\temp\test.csv''
     out=shoes
     dbms=csv
     replace;
     getnames=no;
run;

proc print;
run;
```



CHAPTER 31

The INFOMAPS Procedure

Information about the INFOMAPS Procedure **605**

Information about the INFOMAPS Procedure

See: For complete documentation about the INFOMAPS procedure, see *Base SAS Guide to Information Maps*.

CHAPTER
32

The JAVAINFO Procedure

Overview: JAVAINFO Procedure **607**
Syntax: JAVAINFO Procedure **607**
PROC JAVAINFO Statement **607**

Overview: JAVAINFO Procedure

The JAVAINFO procedure conveys diagnostic information to the user about the Java environment that SAS is using. The diagnostic information can be used to confirm that the SAS Java environment has been configured correctly, and can be helpful when reporting problems to SAS technical support. Also, PROC JAVAINFO is often used to verify that the SAS Java environment is working correctly because PROC JAVAINFO uses Java to report its diagnostics.

Syntax: JAVAINFO Procedure

PROC JAVAINFO <*options*>;

PROC JAVAINFO Statement

PROC JAVAINFO <*options*>;

Required Arguments

There are no required arguments.

Options

ALL

specifies current information about the SAS Java environment.

CLASSPATHS
specifies information about the classpaths that Java is using.

HELP
specifies usage assistance in using the JAVAINFO procedure.

JREOPTIONS
specifies the Java properties that are set when the JREOPTIONS configuration option is specified.

- When used in PROC JAVAINFO, JREOPTIONS specifies the SAS-JREOPTIONS-Java properties that are set when Java is started.
- When used in PROC OPTIONS, JREOPTIONS specifies the Java options that are in the configuration file when SAS is started.

Note: SAS.cfg is the configuration file specified during installation, but other configuration files can be specified. △

OS
specifies information about the operating system that SAS is running under.

version
specifies the Java Runtime Environment (JRE) that SAS is using.

CHAPTER

33

The MEANS Procedure

Overview: MEANS Procedure **610**
What Does the MEANS Procedure Do? **610**
What Types of Output Does PROC MEANS Produce? **610**
Syntax: MEANS Procedure **612**
PROC MEANS Statement **613**
BY Statement **621**
CLASS Statement **622**
FREQ Statement **626**
ID Statement **626**
OUTPUT Statement **627**
TYPES Statement **633**
VAR Statement **635**
WAYS Statement **636**
WEIGHT Statement **636**
Concepts: MEANS Procedure **637**
Using Class Variables **637**
Using TYPES and WAYS Statements **638**
Ordering the Class Values **638**
Computational Resources **639**
In-Database Processing for PROC MEANS **640**
Statistical Computations: MEANS Procedure **641**
Computation of Moment Statistics **641**
Confidence Limits **642**
Student's t Test **643**
Quantiles **643**
Results: MEANS Procedure **644**
Missing Values **644**
Column Width for the Output **644**
The N Obs Statistic **644**
Output Data Set **645**
Examples: MEANS Procedure **646**
Example 1: Computing Specific Descriptive Statistics **646**
Example 2: Computing Descriptive Statistics with Class Variables **648**
Example 3: Using the BY Statement with Class Variables **650**
Example 4: Using a CLASSDATA= Data Set with Class Variables **652**
Example 5: Using Multilabel Value Formats with Class Variables **655**
Example 6: Using Preloaded Formats with Class Variables **659**
Example 7: Computing a Confidence Limit for the Mean **662**
Example 8: Computing Output Statistics **663**
Example 9: Computing Different Output Statistics for Several Variables **665**
Example 10: Computing Output Statistics with Missing Class Variable Values **667**

Example 11: Identifying an Extreme Value with the Output Statistics **669**
Example 12: Identifying the Top Three Extreme Values with the Output Statistics **672**
References **675**

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 57, "The SUMMARY Procedure," on page 1351 for an explanation of the differences.

What Types of Output Does PROC MEANS Produce?

PROC MEANS Default Output

The following output shows the default output that PROC MEANS displays. The data set that PROC MEANS analyzes contains the integers 1 through 10. The output reports the number of observations, the mean, the standard deviation, the minimum value, and the maximum value. The statements that produce the output follow:

```
proc means data=OnetoTen;
run;
```

Output 33.1 The Default Descriptive Statistics

```
                              The SAS System                                1

                          The MEANS Procedure

                       Analysis Variable : Integer

        N            Mean         Std Dev         Minimum         Maximum
-------------------------------------------------------------------------------
       10       5.5000000       3.0276504       1.0000000      10.0000000
-------------------------------------------------------------------------------
```

PROC MEANS Customized Output

The following output shows the results of a more extensive analysis of two variables, MoneyRaised and HoursVolunteered. The analysis data set contains information about the amount of money raised and the number of hours volunteered by high-school students for a local charity. PROC MEANS uses six combinations of two categorical variables to compute the number of observations, the mean, and the range. The first variable, School, has two values and the other variable, Year, has three values. For an explanation of the program that produces the output, see Example 11 on page 669.

Output 33.2 Specified Statistics for Class Levels and Identification of Maximum Values

Summary of Volunteer Work by School and Year 1

The MEANS Procedure

School	Year	N Obs	Variable	N	Mean	Range
Kennedy	1992	15	MoneyRaised	15	29.0800000	39.7500000
			HoursVolunteered	15	22.1333333	30.0000000
	1993	20	MoneyRaised	20	28.5660000	23.5600000
			HoursVolunteered	20	19.2000000	20.0000000
	1994	18	MoneyRaised	18	31.5794444	65.4400000
			HoursVolunteered	18	24.2777778	15.0000000
Monroe	1992	16	MoneyRaised	16	28.5450000	48.2700000
			HoursVolunteered	16	18.8125000	38.0000000
	1993	12	MoneyRaised	12	28.0500000	52.4600000
			HoursVolunteered	12	15.8333333	21.0000000
	1994	28	MoneyRaised	28	29.4100000	73.5300000
			HoursVolunteered	28	19.1428571	26.0000000

Best Results: Most Money Raised and Most Hours Worked 2

Obs	School	Year	_TYPE_	_FREQ_	Most Cash	Most Time	Money Raised	Hours Volunteered
1		.	0	109	Willard	Tonya	78.65	40
2		1992	1	31	Tonya	Tonya	55.16	40
3		1993	1	32	Cameron	Amy	65.44	31
4		1994	1	46	Willard	L.T.	78.65	33
5	Kennedy	.	2	53	Luther	Jay	72.22	35
6	Monroe	.	2	56	Willard	Tonya	78.65	40
7	Kennedy	1992	3	15	Thelma	Jay	52.63	35
8	Kennedy	1993	3	20	Bill	Amy	42.23	31
9	Kennedy	1994	3	18	Luther	Che-Min	72.22	33
10	Monroe	1992	3	16	Tonya	Tonya	55.16	40
11	Monroe	1993	3	12	Cameron	Myrtle	65.44	26
12	Monroe	1994	3	28	Willard	L.T.	78.65	33

In addition to the report, the program also creates an output data set (located on page 2 of the output) that identifies the students who raised the most money and who volunteered the most time over all the combinations of School and Year and within the combinations of School and Year:

- The first observation in the data set shows the students with the maximum values overall for MoneyRaised and HoursVolunteered.
- Observations 2 through 4 show the students with the maximum values for each year, regardless of school.
- Observations 5 and 6 show the students with the maximum values for each school, regardless of year.
- Observations 7 through 12 show the students with the maximum values for each school-year combination.

Syntax: MEANS Procedure

Tip: Supports the Output Delivery System. See "Output Delivery System: Basic Concepts in *SAS Output Delivery System: User's Guide* for details.

Tip: You can use the ATTRIB, FORMAT, LABEL, and WHERE statements. See Chapter 3, "Statements with the Same Function in Multiple Procedures," on page 35 for details. You can also use any global statements. See "Global Statements" on page 20 for a list.

PROC 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*;

Task	Statement
Compute descriptive statistics for variables	"PROC MEANS Statement" on page 613
Calculate separate statistics for each BY group	"BY Statement" on page 621
Identify variables whose values define subgroups for the analysis	"CLASS Statement" on page 622
Identify a variable whose values represent the frequency of each observation	"FREQ Statement" on page 626
Include additional identification variables in the output data set	"ID Statement" on page 626
Create an output data set that contains specified statistics and identification variables	"OUTPUT Statement" on page 627

Task	Statement
Identify specific combinations of class variables to use to subdivide the data	"TYPES Statement" on page 633
Identify the analysis variables and their order in the results	"VAR Statement" on page 635
Specify the number of ways to make unique combinations of class variables	"WAYS Statement" on page 636
Identify a variable whose values weight each observation in the statistical calculations	"WEIGHT Statement" on page 636

PROC MEANS Statement

See also: Chapter 57, "The SUMMARY Procedure," on page 1351

PROC MEANS <*option(s)*> <*statistic-keyword(s)*>;

Task	Option
Specify the input data set	DATA= on page 615
Disable floating point exception recovery	NOTRAP on page 617
Specify the amount of memory to use for data summarization with class variables	SUMSIZE= on page 620
Override the SAS system option THREADS | NOTHREADS	THREADS | NOTHREADS on page 620
Control the classification levels	
Specify a secondary data set that contains the combinations of class variables to analyze	CLASSDATA= on page 615
Create all possible combinations of class variable values	COMPLETETYPES on page 615
Exclude from the analysis all combinations of class variable values that are not in the CLASSDATA= data set	EXCLUSIVE on page 615
Use missing values as valid values to create combinations of class variables	MISSING on page 616
Control the statistical analysis	
Specify the confidence level for the confidence limits	ALPHA= on page 614
Exclude observations with nonpositive weights from the analysis	EXCLNPWGT on page 615
Specify the sample size to use for the P2 quantile estimation method	QMARKERS= on page 618

Task	Option
Specify the quantile estimation method	QMETHOD= on page 618
Specify the mathematical definition used to compute quantiles	QNTLDEF= on page 619
Select the statistics	statistic-keyword on page 619
Specify the variance divisor	VARDEF= on page 620
Control the output	
Specify the field width for the statistics	FW= on page 616
Specify the number of decimal places for the statistics	MAXDEC= on page 616
Suppress reporting the total number of observations for each unique combination of the class variables	NONOBS on page 616
Suppress all displayed output	NOPRINT on page 618
Order the values of the class variables according to the specified order	ORDER= on page 617
Display the output	PRINT on page 618
Display the analysis for all requested combinations of class variables	PRINTALLTYPES on page 618
Display the values of the ID variables	PRINTIDVARS on page 618
Control the output data set	
Specify that the _TYPE_ variable contain character values.	CHARTYPE on page 614
Order the output data set by descending _TYPE_ value	DESCENDTYPES on page 615
Select ID variables based on minimum values	IDMIN on page 616
Limit the output statistics to the observations with the highest _TYPE_ value	NWAY on page 617

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 642

Featured in: Example 7 on page 662

CHARTYPE
specifies that the _TYPE_ variable in the output data set is a character representation of the binary value of _TYPE_. The length of the variable equals the number of class variables.

Interaction: When you specify more than 32 class variables, _TYPE_ automatically becomes a character variable.

Main discussion: "Output Data Set" on page 645

Featured in: Example 10 on page 667

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 652

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 659

DATA=*SAS-data-set*

identifies the input SAS data set.

Main discussion: "Input Data Sets" on page 20

DESCENDTYPES

orders observations in the output data set by descending _TYPE_ value.

Alias: DESCENDING | DESCEND

Interaction: Descending has no effect if you specify NWAY.

Tip: Use DESCENDTYPES to make the overall total (_TYPE_=0) the last observation in each BY group.

See also: "Output Data Set" on page 645

Featured in: Example 9 on page 665

EXCLNPWGT

excludes observations with nonpositive weight values (zero or negative) from the analysis. By default, PROC MEANS treats observations with negative weights like observations with zero weights and counts them in the total number of observations.

Alias: EXCLNPWGTS

See also: WEIGHT= on page 635 and "WEIGHT Statement" on page 636

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 652

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 646

Example 4 on page 652

Example 5 on page 655

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 626

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 648

Example 4 on page 652

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 659

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 644

Featured in:

Example 5 on page 655

Example 6 on page 659

NOPRINT
See PRINT | NOPRINT on page 618.

NOTHREADS
See THREADS | NOTHREADS on page 620.

NOTRAP
disables floating point exception (FPE) recovery during data processing. By default, PROC MEANS traps these errors and sets the statistic to missing.

In operating environments where the overhead of FPE recovery is significant, NOTRAP can improve performance. Note that normal SAS FPE handling is still in effect so that PROC MEANS terminates in the case of math exceptions.

NWAY
specifies that the output data set contain only statistics for the observations with the highest _TYPE_ and _WAY_ values. When you specify class variables, NWAY corresponds to the combination of all class variables.

Interaction: If you specify a TYPES statement or a WAYS statement, then PROC MEANS ignores this option.

See also: "Output Data Set" on page 645

Featured in: Example 10 on page 667

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 638

PCTLDEF=
PCTLDEF is an alias for QNTLDEF=.

See also: QNTLDEF= on page 619

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 663

Example 12 on page 672

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 652

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 626

QMARKERS=*number*
specifies the default number of markers to use for the P^2 quantile estimation method. The number of markers controls the size of fixed memory space.

Default: The default value depends on which quantiles you request. For the median (P50), *number* is 7. For the quantiles (P25 and P50), *number* is 25. For the quantiles P1, P5, P10, P75 P90, P95, or P99, *number* is 105. If you request several quantiles, then PROC MEANS uses the largest value of *number*.

Range: an odd integer greater than 3

Tip: Increase the number of markers above the defaults settings to improve the accuracy of the estimate; reduce the number of markers to conserve memory and computing time.

Main Discussion: "Quantiles" on page 643

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 method 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 the following:

- MODE
- weighted quantiles

Tip: When QMETHOD=P2, reliable estimations of some quantiles (P1,P5,P95,P99) might not be possible for some data sets.

Main Discussion: "Quantiles" on page 643

QNTLDEF=1|2|3|4|5

specifies the mathematical definition that PROC MEANS uses to calculate quantiles when QMETHOD=OS. To use QMETHOD=P2, you must use QNTLDEF=5.

Alias: PCTLDEF=

Default: 5

Main discussion: "Quantile and Related Statistics" on page 1541

statistic-keyword(s)

specifies which statistics to compute and the order to display them in the output. The available keywords in the PROC statement are

Descriptive statistic keywords	
CLM	NMISS
CSS	RANGE
CV	SKEWNESS|SKEW
KURTOSIS|KURT	STDDEV|STD
LCLM	STDERR
MAX	SUM
MEAN	SUMWGT
MIN	UCLM
MODE	USS
N	VAR
Quantile statistic keywords	
MEDIAN|P50	Q3|P75
P1	P90
P5	P95
P10	P99
Q1|P25	QRANGE
Hypothesis testing keywords	
PROBT|PRT	T

Default: N, MEAN, STD, MIN, and MAX

Requirement: To compute standard error, confidence limits for the mean, and the Student's *t*-test, you must use the default value of the VARDEF= option, which is DF. To compute skewness or kurtosis, you must use VARDEF=N or VARDEF=DF.

Tip: Use CLM or both LCLM and UCLM to compute a two-sided confidence limit for the mean. Use only LCLM or UCLM, to compute a one-sided confidence limit.

Main discussion: The definitions of the keywords and the formulas for the associated statistics are listed in "Keywords and Formulas" on page 1536.

Featured in:

Example 1 on page 646

Example 3 on page 650

SUMSIZE=*value*

specifies the amount of memory that is available for data summarization when you use class variables. *value* might be one of the following:

n|*n*K| *n*M| *n*G
specifies the amount of memory available in bytes, kilobytes, megabytes, or gigabytes, respectively. If *n* is 0, then PROC MEANS use the value of the SAS system option SUMSIZE=.

MAXIMUM|MAX
specifies the maximum amount of memory that is available.

Default: The value of the SUMSIZE= system option.

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.

Main discussion: "Computational Resources" on page 639

Note: Specifying SUMSIZE=0 enables proc MEANS to use the preferred global REALMEMSIZE option. △

See also: The SAS system option SUMSIZE= in *SAS Language Reference: Dictionary*.

THREADS | NOTHREADS

enables or disables parallel processing of the input data set. This option overrides the SAS system option THREADS | NOTHREADS unless the system option is restricted (see Restriction). See *SAS Language Reference: Concepts* for more information about parallel processing.

Default: value of SAS system option THREADS | NOTHREADS.

Restriction: Your site administrator can create a restricted options table. A restricted options table specifies SAS system option values that are established at startup and cannot be overridden. If the THREADS | NOTHREADS system option is listed in the restricted options table, any attempt to set these system options is ignored and a warning message is written to the SAS log.

Interaction: PROC MEANS honors the SAS system option THREADS except when a BY statement is specified or the value of the SAS system option CPUCOUNT is less than 2. You can use THREADS in the PROC MEANS statement to force PROC MEANS to use parallel processing in these situations.

Note: If THREADS is specified (either as a SAS system option or on the PROC MEANS statement) and another program has the input data set open for reading, writing, or updating, then PROC MEANS might fail to open the input data set. In this case, PROC MEANS stops processing and writes a message to the SAS log. △

VARDEF=*divisor*

specifies the divisor to use in the calculation of the variance and standard deviation. The following table shows the possible values for *divisor* and associated divisors.

Table 33.1 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 method 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 method yields an asymptotic estimate of the variance of an observation with average weight.

Main discussion: "Keywords and Formulas" on page 1536

See also: "Weighted Statistics Example" on page 43

BY Statement

Produces separate statistics for each BY group.

Main discussion: "BY" on page 36

See also: "Comparison of the BY and CLASS Statements" on page 625

Featured in: Example 3 on page 650

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. 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 21 and “Suppressing the Default BY Line” on page 21.)

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 26.

Featured in:

Example 2 on page 648
Example 4 on page 652
Example 5 on page 655
Example 6 on page 659
Example 10 on page 667

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 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 637

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 667

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 659

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 626

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 667

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 might not produce the order that you expect for the formatted values. You might not get the expected results when you used MLF with CLASSDATA and EXCLUSIVE because MLF processing requires that each TYPE be computed independently. Types other than NWAY might contain more levels than expected.

Tip: If you omit MLF, then PROC MEANS uses the primary format labels. This action corresponds to using the first external format value to determine the subgroup combinations.

See also: The MULTILABEL option in the VALUE statement of the FORMAT procedure on page 522.

Featured in: Example 5 on page 655

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 667

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 655

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 655

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 638

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 659

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 650.

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 639 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 39

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 627.

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 1182 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 663
Example 9 on page 665
Example 10 on page 667
Example 11 on page 669
Example 12 on page 672

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 19 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
MODE	USS
N	VAR
NMISS	
Quantile statistics keyword	
MEDIAN | P50	Q3 | P75
P1	P90
P5	P95
P10	P99
Q1 | P25	QRANGE
Hypothesis testing keyword	
PROBT | PRT	T

By default the statistics in the output data set automatically inherit the analysis variable's format, informat, and label. However, statistics computed for N, NMISS, SUMWGT, USS, CSS, VAR, CV, T, PROBT, PRT, SKEWNESS, and KURTOSIS will not inherit the analysis variable's format because this format might be invalid for these statistics (for example, dollar or datetime formats).

Restriction: If you omit *variable* and *name(s)*, then PROC MEANS allows the *statistic-keyword* only once in a single OUTPUT statement, unless you also use the AUTONAME option.

Featured in:

Example 8 on page 663

Example 9 on page 665
Example 11 on page 669
Example 12 on page 672

variable-list
specifies the names of one or more numeric analysis variables whose statistics you want to store in the output data set.

Default: all numeric analysis variables

name(s)
specifies one or more names for the variables in output data set that will contain the analysis variable statistics. The first name contains the statistic for the first analysis variable; the second name contains the statistic for the second analysis variable; and so on.

Default: the analysis variable name. If you specify AUTONAME, then the default is the combination of the analysis variable name and the *statistic-keyword*. If you use the CLASS statement and an OUTPUT statement without an *output-statistic-specification*, then the output data set contains five observations for each combination of class variables: the value of N, MIN, MAX, MEAN, and STD. If you use the WEIGHT statement or the WEIGHT option in the VAR statement, then the output data set also contains an observation with the sum of weights (SUMWGT) for each combination of class variables.

Interaction: If you specify *variable-list*, then PROC MEANS uses the order in which you specify the analysis variables to store the statistics in the output data set variables.

Tip: Use the AUTONAME option to have PROC MEANS generate unique names for multiple variables and statistics.

Featured in: Example 8 on page 663

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 $\mathrm{O}(T * N \log_2 n)$, where T is the number of types requested and N is the number of observations in the input data set. By comparison, to group the entire data set, the time complexity is $\mathrm{O}(N \log_2 N)$.

Default: 1

Range: an integer between 1 and 100

Featured in: For example, to output two minimum extreme values for each variable, use

```
idgroup(min(x) out[2](x y z)=MinX MinY MinZ);
```

The OUT= data set contains the variables MinX_1, MinX_2, MinY_1, MinY_2, MinZ_1, and MinZ_2.

(*id-variable-list*)
identifies one or more input data set variables whose values PROC MEANS includes in the OUT= data set. PROC MEANS determines which observations to output by the selection criteria that you specify (MIN, MAX, and LAST).

Alias: IDGRP

Requirement: You must specify the MIN|MAX selection criteria first and OUT(*id-variable-list*)= after the suboptions MISSING, OBS, and LAST.

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 663 and Example 12 on page 672

name(s)
specifies one or more names for variables in the OUT= data set.

Default: If you omit *name*, then PROC MEANS uses the names of variables in the *id-variable-list*.

Tip: Use the AUTONAME option to automatically resolve naming conflicts.

CAUTION:
The IDGROUP syntax allows you to create output variables with the same name. When this action happens, only the first variable appears in the output data set. Use the AUTONAME option to automatically resolve these naming conflicts. △

Note: If you specify fewer new variable names than the combination of analysis variables and identification variables, then the remaining output variables use the corresponding names of the ID variables as soon as PROC MEANS exhausts the list of new variable names. △

maximum-id-specification(s)
specifies that one or more identification variables be associated with the maximum values of the analysis variables. The form of the *maximum-id-specification* is

MAXID <(*variable-1* <(*id-variable-list-1*)> <…*variable-n* <(*id-variable-list-n*)>>)> = *name(s)*

variable
identifies the numeric analysis variable whose maximum values PROC MEANS determines. PROC MEANS can determine several maximum values for a variable because, in addition to the overall maximum value, subgroup levels, which are defined by combinations of class variables values, also have maximum values.

Tip: If you use an ID statement and omit *variable*, then PROC MEANS uses all analysis variables.

id-variable-list
identifies one or more variables whose values identify the observations with the maximum values of the analysis variable.

Default: the ID statement variables

name(s)
specifies the names for new variables that contain the values of the identification variable associated with the maximum value of each analysis variable.

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.

Note: If multiple observations contain the maximum value within a class level, then PROC MEANS saves the value of the ID variable for only the first of those observations in the output data set. △

Featured in: Example 11 on page 669

CAUTION:
The MAXID syntax allows you to create output variables with the same name. When this action happens, only the first variable appears in the output data set. Use the AUTONAME option to automatically resolve these naming conflicts. △

Note: If you specify fewer new variable names than the combination of analysis variables and identification variables, then the remaining output variables use the corresponding names of the ID variables as soon as PROC MEANS exhausts the list of new variable names. △

minid-specification
See the description of maximum-id-specification on page 631. This option behaves in exactly the same way, except that PROC MEANS determines the minimum values instead of the maximum values. The form of the *minid-specification* is

MINID<(*variable-1* <(*id-variable-list-1*)> <…*variable-n* <(*id-variable-list-n*)>>)> = *name(s)*

When MINID is used without an explicit variable list, it is similar to the following more advanced IDGROUP syntax example:

```
IDGRP( min(x) missing out(id_variable)=idminx) idgrp( min(y) missing
    out(id_variable)=idminy)
```

If one or more of the analysis variables has a missing value, the id_variable value will correspond to the observation with the missing value not the observation with the value for the MIN statistic.

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 672

AUTONAME
specifies that PROC MEANS creates a unique variable name for an output statistic when you do not assign the variable name in the OUTPUT statement. This action is accomplished by appending 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 672

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 645

Featured in: Example 8 on page 663

NOINHERIT
specifies that the variables in the output data set that contain statistics do not inherit the attributes (label and format) of the analysis variables which are used to derive them.

Interaction When no option is used (implied INHERIT) then the statistics inherit the attributes, label and format, of the input analysis variable(s). If the INHERIT option is used on the OUTPUT statement, then the statistics inherit the length of the input analysis variable(s), the label and format.

Tip: By default, the output data set includes an output variable for each analysis variable and for five observations that contain N, MIN, MAX, MEAN, and STDDEV. Unless you specify NOINHERIT, this variable inherits the format of the analysis variable, which can be invalid for the N statistic (for example, datetime formats).

WAYS
includes a variable named _WAY_ in the output data set. This variable contains a value from 1 to the maximum number of class variables that indicates how many class variables PROC MEANS combines to create the TYPE value.

Main discussion: "Output Data Set" on page 645

See also: "WAYS Statement" on page 636

Featured in: Example 8 on page 663

TYPES Statement

Identifies which of the possible combinations of class variables to generate.

Requirement: CLASS statement

Main discussion: "Output Data Set" on page 645

Featured in:
Example 2 on page 648
Example 5 on page 655
Example 12 on page 672

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 645.

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 57, "The SUMMARY Procedure," on page 1351

Featured in: Example 1 on page 646

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 620.

Tip: Use the WEIGHT option in multiple VAR statements to specify different weights for the analysis variables.

Note: Before 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 659

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

See also: WAYS option on page 633

The following code is an example of creating two-way types for the classification variables A, B, and C. This WAYS statement is equivalent to specifying a*b, a*c, and b*c in the TYPES statement.

```
class A B C ;
ways 2;
```

WEIGHT Statement

Specifies weights for observations in the statistical calculations.

See also: For information on how to calculate weighted statistics and for an example that uses the WEIGHT statement, see “WEIGHT” on page 41

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.

Restriction: PROC MEANS will not compute MODE when a weight variable is active. Instead, try using Chapter 63, "The UNIVARIATE Procedure," on page 1527 when MODE needs to be computed and a weight variable is active.

Interaction: If you use the WEIGHT= option in a VAR statement to specify a weight variable, then PROC MEANS uses this variable instead to weight those VAR statement variables.

Tip: When you use the WEIGHT statement, consider which value of the VARDEF= option is appropriate. See the discussion of VARDEF= on page 620 and the calculation of weighted statistics in "Keywords and Formulas" on page 1536 for more information.

Note: Before 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 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 a nonsubdivided view of the entire data set. For example, consider the following statements:

```
data pets;
 input Pet $ Gender $;
 datalines;
dog  m
dog  f
dog  f
dog  f
cat  m
cat  m
cat  f
;

proc means data=pets order=freq;
   class pet gender;
run;
```

The statements produce this output.

```
                          The SAS System                                1

                      The MEANS Procedure

                                          N
Pet          Gender                   Obs
----------------------------------------
dog          f                          3

             m                          1

cat          f                          1

             m                          2
----------------------------------------
```

In the example, PROC MEANS does not list male cats before female cats. Instead, it determines the order of gender for all types over the entire data set. PROC MEANS found more observations for female pets (f=4, m=3).

Computational Resources

PROC MEANS uses the same memory allocation scheme across all operating environments. When class variables are involved, PROC MEANS must keep a copy of each unique value of each class variable in memory. You can estimate the memory requirements to group the class variable by calculating

$$Nc_1\left(Lc_1 + K\right) + Nc_2\left(Lc_2 + K\right) + ... + Nc_n\left(Lc_n + K\right)$$

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 633). 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 the levels of the class variables. For this reason, PROC MEANS can open one or more utility files and write the levels of one or more types to disk. These types are either the primary types that PROC MEANS built during the input data scan or the derived types.

If PROC MEANS must write partially complete primary types to disk while it processes input data, then one or more merge passes can be required to combine type levels in memory with the levels on disk. In addition, if you use an order other than DATA for any class variable, then PROC MEANS groups the completed types on disk. For this reason, the peak disk space requirements can be more than twice the memory requirements for a given type.

When PROC MEANS uses a temporary work file, you will receive the following note in the SAS log:

```
Processing on disk occurred during summarization.
Peak disk usage was approximately nnn Mbytes.
Adjusting 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 can 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.

In-Database Processing for PROC MEANS

When large data sets are stored in an external database, like Teradata, the transfer of the data sets to computers that run SAS can be impacted by performance, security,

and resource management issues. SAS in-database processing can greatly reduce data transfer by having the database perform the initial data aggregation.

Under the correct conditions, PROC MEANS generates an SQL query based on the statements that are used and the output statistics that are specified in the PROC step. If class variables are specified, the procedure creates an SQL GROUP BY clause that represents the n-way type. The result set that is created when the aggregation query executes in the database is read by SAS into the internal PROC MEANS data structure, and all subsequent types are derived from the original n-way type to form the final analysis results. When SAS format definitions have been deployed in the database, formatting of class variables occurs in the database. If the SAS format definitions have not been deployed in the database, the in-database aggregation occurs on the raw values, and the relevant formats are applied by SAS as the results' set is merged into the PROC MEANS internal structures. Multi-label formatting is always done by SAS using the initially aggregated result set that is returned by the database. The CLASS, TYPES, WAYS, VAR, BY, FORMAT, and WHERE statements are supported when PROC MEANS is processed inside the database. FREQ, ID, IDMIN, IDMAX, and IDGROUPS are not supported. The following statistics are supported for in-database processing: N, NMISS, MIN, MAX, RANGE, SUM, SUMWGT, MEAN, CSS, USS, VAR, STD, STDERR, PRET, UCLM, LCLM, CLM and CV.

Weighting for in-database processing is supported only for N, NMISS, MIN, MAX, RANGE, SUM, SUMWGT, and MEAN.

The following statistics are currently not supported for in-database processing: SKEW, KURT, P1, P5, P10, P25/Q1, P50/MEDIAN, P75/Q3, P90, P95, P99, and MODE.

The SQLGENERATION system option or LIBNAME statement option controls whether and how in-database procedures are run inside the database. By default, the in-database procedures are run inside the database when possible. There are many data set options that will prevent in-database processing: OBS=, FIRSTOBS=, RENAME=, and DBCONDITION=. For a complete listing, refer to "Overview of In-Database Procedures" in *SAS/ACCESS for Relational Databases: Reference*.

In-database processing can greatly reduce the volume of data transferred to the procedure if there are no class variables (one row is returned) or if the selected class variables have a small number of unique values. However, because PROC MEANS loads the result set into its internal structures, the memory requirements for the SAS process will be equivalent to what would have been required without in-database processing. The CPU requirements for the SAS process should be significantly reduced if the bulk of the data summarization occurs inside the database. The real time required for summarization should be significantly reduced because many database-process queries are in parallel.

For more information on database processing, see *SAS/ACCESS for Relational Databases: Reference*.

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 1536 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.

If you use the WEIGHT statement or WEIGHT= in a VAR statement and the default value of VARDEF=, which is DF, the $100\,(1-\alpha)\%$ confidence interval for the weighted mean has upper and lower limits

$$\overline{y}_w \pm t_{(1-\alpha/2)} \frac{s_w}{\sqrt{\sum_{i=1}^{n} w_i}}$$

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 1541. QMETHOD= deals with the mechanics of how PROC MEANS handles the input data. The two methods are

OS
: reads all data into memory and sorts it by unique value.

P2
: accumulates all data into a fixed sample size that is used to approximate the quantile.

If data set A has 100 unique values for a numeric variable X and data set B has 1000 unique values for numeric variable X, then QMETHOD=OS for data set B will take 10 times as much memory as it does for data set A. If QMETHOD=P2, then both data sets A and B will require the same memory space to generate quantiles.

The QMETHOD=P2 technique is based on the piecewise-parabolic (P^2) algorithm invented by Jain and Chlamtac (1985). P^2 is a one-pass algorithm to determine quantiles for a large data set. It requires a fixed amount of memory for each variable for each level within the type. However, using simulation studies, reliable estimations of some quantiles (P1, P5, P95, P99) cannot be possible for some data sets such as data sets with heavily tailed or skewed distributions.

If the number of observations is less than the QMARKERS= value, then QMETHOD=P2 produces the same results as QMETHOD=OS when QNTLDEF=5. To compute weighted quantiles, you must use QMETHOD=OS.

Results: MEANS Procedure

Missing Values

PROC MEANS excludes missing values for the analysis variables before calculating statistics. Each analysis variable is treated individually; a missing value for an observation in one variable does not affect the calculations for other variables. The statements handle missing values as follows:

- If a class variable has a missing value for an observation, then PROC MEANS excludes that observation from the analysis unless you use the MISSING option in the PROC statement or CLASS statement.
- If a BY or ID variable value is missing, then PROC MEANS treats it like any other BY or ID variable value. The missing values form a separate BY group.
- If a FREQ variable value is missing or nonpositive, then PROC MEANS excludes the observation from the analysis.
- If a WEIGHT variable value is missing, then PROC MEANS excludes the observation from the analysis.

PROC MEANS tabulates the number of the missing values. Before the number of missing values are tabulated, PROC MEANS excludes observations with frequencies that are nonpositive when you use the FREQ statement and observations with weights that are missing or nonpositive (when you use the EXCLNPWGT option) when you use the WEIGHT statement. To report this information in the procedure output use the NMISS statistical keyword in the PROC statement.

Column Width for the Output

You control the column width for the displayed statistics with the FW= option in the PROC statement. Unless you assign a format to a numeric class or an ID variable, PROC MEANS uses the value of the FW= option. When you assign a format to a numeric class or an ID variable, PROC MEANS determines the column width directly from the format. If you use the PRELOADFMT option in the CLASS statement, then PROC MEANS determines the column width for a class variable from the assigned format.

The N Obs Statistic

By default when you use a CLASS statement, PROC MEANS displays an additional statistic called N Obs. This statistic reports the total number of observations or the sum of the observations of the FREQ variable that PROC MEANS processes for each class level. PROC MEANS might omit observations from this total because of missing values in one or more class variables or because of the effect of the EXCLUSIVE option when you use it with the PRELOADFMT option or the CLASSDATA= option. Because of this

action and the exclusion of observations when the WEIGHT variable contains missing values, there is not always a direct relationship between N Obs, N, and NMISS.

In the output data set, the value of N Obs is stored in the _FREQ_ variable. Use the NONOBS option in the PROC statement to suppress this information in the displayed output.

Output Data Set

PROC MEANS can create one or more output data sets. The procedure does not print the output data set. Use PROC PRINT, PROC REPORT, or another SAS reporting tool to display the output data set.

Note: By default the statistics in the output data set automatically inherit the analysis variable's format and label. However, statistics computed for N, NMISS, SUMWGT, USS, CSS, VAR, CV, T, PROBT, PRT,SKEWNESS, and KURTOSIS do not inherit the analysis variable's format because this format can be invalid for these statistics. Use the NOINHERIT option in the OUTPUT statement to prevent the other statistics from inheriting the format and label attributes. △

The output data set can contain these variables:

- the variables specified in the BY statement.
- the variables specified in the ID statement.
- the variables specified in the CLASS statement.
- the variable _TYPE_ that contains information about the class variables. By default _TYPE_ is a numeric variable. If you specify CHARTYPE in the PROC statement, then _TYPE_ is a character variable. When you use more than 32 class variables, _TYPE_ is automatically a character variable.
- the variable _FREQ_ that contains the number of observations that a given output level represents.
- the variables requested in the OUTPUT statement that contain the output statistics and extreme values.
- the variable _STAT_ that contains the names of the default statistics if you omit statistic keywords.
- the variable _LEVEL_ if you specify the LEVEL option.
- the variable _WAY_ if you specify the WAYS option.

The value of _TYPE_ indicates which combination of the class variables PROC MEANS uses to compute the statistics. The character value of _TYPE_ is a series of zeros and ones, where each value of one indicates an active class variable in the type. For example, with three class variables, PROC MEANS represents type 1 as 001, type 5 as 101, and so on.

Usually, the output data set contains one observation per level per type. However, if you omit statistical keywords in the OUTPUT statement, then the output data set contains five observations per level (six if you specify a WEIGHT variable). Therefore, the total number of observations in the output data set is equal to the sum of the levels for all the types you request multiplied by 1, 5, or 6, whichever is applicable.

If you omit the CLASS statement (_TYPE_= 0), then there is always exactly one level of output per BY group. If you use a CLASS statement, then the number of levels for each type that you request has an upper bound equal to the number of observations in the input data set. By default, PROC MEANS generates all possible types. In this case the total number of levels for each BY group has an upper bound equal to

$$m \cdot \left(2^k - 1\right) \cdot n + 1$$

where k is the number of class variables and n is the number of observations for the given BY group in the input data set and m is 1, 5, or 6.

PROC MEANS determines the actual number of levels for a given type from the number of unique combinations of each active class variable. A single level consists of all input observations whose formatted class values match.

The following figure shows the values of _TYPE_ and the number of observations in the data set when you specify one, two, and three class variables.

Figure 33.1 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 variable 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 649

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 647

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

Layers	N Obs	Range	Median	Minimum	Maximum
1	8	19	82	75	94
2	5	20	75	72	92
3	0	.	.	.	.

Analysis Variable : TasteScore

Flavor	N Obs	Range	Median	Minimum	Maximum
Chocolate	8	20	81	72	92
Vanilla	5	21	80	73	94

Analysis Variable : TasteScore

Flavor	Layers	N Obs	Range	Median	Minimum	Maximum
Chocolate	1	5	6	83	79	85
	2	3	20	75	72	92
	3	0	.	.	.	.
Vanilla	1	3	19	80	75	94
	2	2	14	80	73	87
	3	0	.	.	.	.

Example 5: Using Multilabel Value Formats with Class Variables

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 647

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. NOTSORTED stores values or ranges for informats or formats in the order in which you define them. 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 notsorted  pageno=1 linesize=80 pagesize=64;
```

Create the $FLVRFMT. and AGEFMT. formats. PROC FORMAT creates user-defined formats to categorize the cake flavors and ages of the participants. MULTILABEL creates a multilabel format for Age. A multilabel format is one in which multiple labels can be assigned to the same value, in this case because of overlapping ranges. Each value is represented in the output for each range in which it occurs.

```
proc format;
   value $flvrfmt
               'Chocolate'='Chocolate'
```

```
                'Vanilla'='Vanilla'
                'Rum','Spice'='Other Flavor';
    value agefmt (multilabel)
                  15 - 29='below 30 years'
                  30 - 50='between 30 and 50'
                  51 - high='over 50 years'
                  15 - 19='15 to 19'
                  20 - 25='20 to 25'
                  25 - 39='25 to 39'
                  40 - 55='40 to 55'
                  56 - high='56 and above';
run;
```

Specify the analyses and the analysis options. FW= uses a field width of six to display the statistics. The statistic keywords specify the statistics and their order in the output. NONOBS suppresses the N Obs column.

```
proc means data=cake fw=6 n min max median nonobs;
```

Specify subgroups for the analysis. The CLASS statements separate the analysis by values of Flavor and Age. ORDER=DATA orders values according to their order in the input data set. ORDER=FMT orders the levels of Age by ascending formatted values. MLF specifies that multilabel value formats be used for Age.

```
    class flavor/order=data;
    class  age /mlf order=fmt;
```

Specify which subgroups to analyze. The TYPES statement requests the analysis for the one-way combination of Flavor and the two-way combination of Flavor and Age.

```
     types flavor flavor*age;
```

Specify the analysis variable. The VAR statement specifies that PROC MEANS calculate statistics on the TasteScore variable.

```
    var TasteScore;
```

Format the output. The FORMAT statement assigns user-defined formats to the Age and Flavor variables for this analysis.

```
    format age agefmt. flavor $flvrfmt.;
```

Specify the title.

```
    title 'Taste Score for Cake Flavors and Participant''s Age';
run;
```

Output

The one-way combination of class variables appears before the two-way combination. A field width of six truncates the statistics to four decimal places. For the two-way combination of Age and Flavor, the total number of observations is greater than the one-way combination of Flavor. This situation arises because of the multilabel format for age, which maps one internal value to more than one formatted value.

The order of the levels of Flavor is based on the frequency count for each level. The order of the levels of Age is based on the order of the user-defined formats.

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	below 30 years	5	75.00	85.00	79.00
	25 to 39	4	75.00	85.00	81.00
	between 30 and 50	2	83.00	92.00	87.50
	40 to 55	2	72.00	92.00	82.00
	20 to 25	1	84.00	84.00	84.00
	over 50 years	2	72.00	84.00	78.00
	15 to 19	1	79.00	79.00	79.00
	56 and above	1	84.00	84.00	84.00
Vanilla	below 30 years	1	80.00	80.00	80.00
	25 to 39	2	73.00	80.00	76.50
	between 30 and 50	2	73.00	75.00	74.00
	40 to 55	1	75.00	75.00	75.00
	over 50 years	3	84.00	94.00	87.00
	56 and above	3	84.00	94.00	87.00
Other Flavor	below 30 years	1	81.00	81.00	81.00
	25 to 39	3	72.00	83.00	81.00
	between 30 and 50	2	72.00	83.00	77.50
	40 to 55	1	91.00	91.00	91.00
	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 647

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 1575 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 649

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 649

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 647

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 chartype nway noprint;
```

Specify subgroups for the analysis. The CLASS statements separate the analysis by Flavor and Layers. ORDER=FREQ and ASCENDING order the levels of Flavor by ascending frequency. MISSING uses missing values of Layers as a valid class level value.

```
   class flavor /order=freq ascending;
   class layers /missing;
```

Specify the analysis variable. The VAR statement specifies that PROC MEANS calculate statistics on the TasteScore variable.

```
   var TasteScore;
```

Specify the output data set options. The OUTPUT statement creates the CAKESTAT data set and outputs the maximum value for the taste score to the new variable HighScore.

```
   output out=cakestat max=HighScore;
run;
```

Print the output data set WORK.CAKESTAT.

```
proc print data=cakestat;
   title 'Maximum Taste Score for Flavor and Cake Layers';
run;
```

Output

The CAKESTAT output data set contains only observations for the combination of the two class variables, Flavor and Layers. Therefore, the value of _TYPE_ is 11 for all observations. The observations are ordered by ascending frequency of Flavor. The missing value in Layers is a valid value for this class variable. PROC MEANS excludes the observation with the missing flavor because it is an invalid value for Flavor.

Maximum Taste Score for Flavor and Cake Layers 1

Obs	Flavor	Layers	_TYPE_	_FREQ_	High Score
1	Rum	2	11	1	72
2	Spice	2	11	2	83
3	Spice	3	11	1	91
4	Vanilla	.	11	1	84
5	Vanilla	1	11	3	94
6	Vanilla	2	11	2	87
7	Chocolate	.	11	1	84
8	Chocolate	1	11	5	85
9	Chocolate	2	11	3	92

Example 11: Identifying an Extreme Value with the Output Statistics

Procedure features:

CLASS statement

OUTPUT statement options:

statistic keyword

MAXID

Other features:

PRINT procedure

Data set: CHARITY on page 662

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 662

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, 2008    Observation Length    144
Last Modified        18:59 Thursday, March 14, 2008    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.

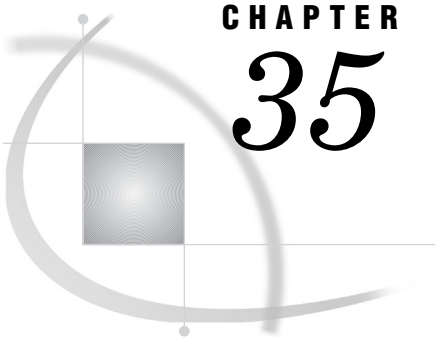
CHAPTER
34

The METADATA Procedure

Information about the METADATA Procedure **677**

Information about the METADATA Procedure

See: For documentation about the METADATA procedure, see *SAS Language Interfaces to Metadata*.



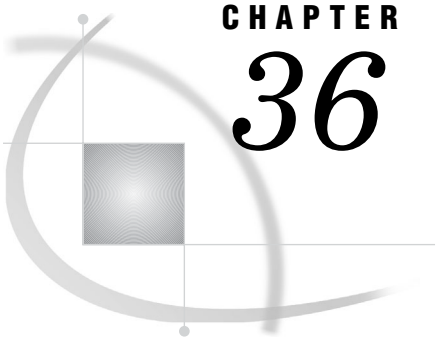
CHAPTER
35

The METALIB Procedure

Information about the METALIB Procedure **679**

Information about the METALIB Procedure

See: For documentation about the METALIB procedure, see *SAS Language Interfaces to Metadata*.



CHAPTER
36

The METAOPERATE Procedure

Information about the METAOPERATE Procedure **681**

Information about the METAOPERATE Procedure

See: For documentation about the METAOPERATE procedure, see *SAS Language Interfaces to Metadata*.

CHAPTER
37

The MIGRATE Procedure

Overview: MIGRATE Procedure **683**
What Does the MIGRATE Procedure Do? **683**
Best Practices **684**
Syntax: MIGRATE Procedure **685**
PROC MIGRATE Statement **685**
Concepts: MIGRATE Procedure **687**
What Are the Specific Considerations for Each Member Type? **687**
Migrating a Data File with Audit Trails, Generations, Indexes, or Integrity Constraints **689**
Migrating a SAS Data Set with NODUPKEY Sort Indicator **689**
Migrating a SAS 6 Library **690**
Migrating SAS Data Sets that Contain Non-English Characters **690**
Migrating Files with Short Extensions on PC Operating Environments **691**
SAS 9 Compatibility with Short-Extension Files **692**
Migrating a Library with Validation Tools **693**
Introduction **693**
Using the MOVE Option with Validation Tools **693**
Using the SLIBREF= Option **693**
When to Use the SLIBREF= Option **694**
When to Not Use the SLIBREF= Option **694**
Requirements for the SAS/CONNECT or SAS/SHARE Server **694**
Restrictions for the SLIBREF= Option **695**
Examples **695**
Example 1: Migrating without SLIBREF= across Computers **695**
Example 2: Migrating with SLIBREF= across Computers **696**
Example 3: Migrating without SLIBREF= on the Same Computer **698**
Example 4: Migrating with SLIBREF= on the Same Computer **698**
Example 5: Additional Steps for Unsupported Catalogs **699**
Example 6: Alternatives to PROC MIGRATE **700**

Overview: MIGRATE Procedure

What Does the MIGRATE Procedure Do?

The MIGRATE procedure migrates members in a SAS library to the current SAS version.

The procedure migrates a library from most SAS 6, SAS 7, SAS 8, and SAS 9 operating environments to the current release of SAS. The migration must occur within

the same engine family; for example, V6, V7, or V8 can migrate to V9, but V6TAPE must migrate to V9TAPE.

The procedure migrates the following library members:

- data sets with alternate collating sequence, audit trails, compression, created and modified datetimes, deleted observations, encryption, generations, indexes, integrity constraints, and passwords
- in many cases, views, catalogs, item stores, and MDDBs (see "What Are the Specific Considerations for Each Member Type?" on page 687)

The procedure does not support stored compiled DATA step programs or stored compiled macros. Instead, move the source code to the target, where you can compile and store it.

The procedure does not support SPD engine (Scalable Performance Data engine) data sets. If you are staying on the same operating environment, the SPD engine data sets are compatible. If you are changing to a different operating environment where the data sets are not compatible, use PROC CPORT and PROC CIMPORT to convert them. See *SAS Scalable Performance Data Engine: Reference*.

Best Practices

First use the Compatibility Calculator on the SAS Web site at **`http://support.sas.com/rnd/migration/planning/files/calculator/`** to determine whether you must migrate your libraries. Then use the PROC MIGRATE Calculator on the SAS Web site at **`http://support.sas.com/rnd/migration/planning/files/migratecalc/`**, which provides the PROC MIGRATE syntax for your migration.

To document and validate the migration of your libraries, use the MIGRATE procedure's validation tools. The validation tools are located on the SAS Web site at **`http://support.sas.com/rnd/migration/resources/procmigrate/validtools.html`**. For more information, see "Migrating a Library with Validation Tools" on page 693.

Syntax: MIGRATE Procedure

Restriction: SAS data set options are not supported.

Restriction: The source and target libraries must be in different physical locations.

Restriction: When PROC MIGRATE creates a member in the target library, the member does not retain its permissions from the source library. It has the permissions associated with the user ID of the person who ran PROC MIGRATE.

Restriction: Source libraries that were created under a SAS 8.2 CMS, OS/2, OpenVMS VAX, or 64–bit AIX operating environment are not supported. See Example 6 on page 700.

Restriction: Source libraries that were created under a 64–bit Windows source environment are supported only for a 64–bit Windows target environment. To migrate to a different target environment, see Example 6 on page 700.

Restriction: Catalogs that were created under a Tru64 UNIX source environment are not supported for a Linux for x64 or a Solaris for x64 target environment. To migrate catalogs under those conditions, see Example 5 on page 699.

Restriction: SAS files that were created before SAS 6.12 (SAS 6.09E for z/OS) must be converted to SAS 6.12 before they can be migrated to the current release of SAS. Some SAS 6 source environments are not supported; see "Migrating a SAS 6 Library" on page 690.

Tip: For encoding and transcoding issues, see the topic about national language support within the migration topics at **`http://support.sas.com/rnd/migration/`** or see the *SAS National Language Support (NLS): Reference Guide*.

PROC MIGRATE IN=*libref-2* OUT=*libref-1* <BUFSIZE=*n*> <MOVE>
<SLIBREF=*libref*> <KEEPNODUPKEY>;

Task	Language Element
Migrate SAS library forward to the current release of SAS	PROC MIGRATE
Name the source library	IN=
Name the target library	OUT=
Specify buffer page size	BUFSIZE=
Migrate and delete the original source files	MOVE
Specify a libref that was assigned through SAS/CONNECT or SAS/SHARE to migrate foreign catalogs	SLIBREF=
Retain the NODUPKEY sort assertion	KEEPNODUPKEY

PROC MIGRATE Statement

PROC MIGRATE IN=*libref-1* OUT=*libref-2* <BUFSIZE=*n*> <MOVE>
<SLIBREF=*libref*> <KEEPNODUPKEY> ;

Required Arguments

IN=*libref-1*
names the source SAS library from which to migrate members.

Requirement: If you use a server, such as SAS/CONNECT or SAS/SHARE, the server must be SAS 9 or later.

Requirement: If cross-environment data access (CEDA) processing is invoked, and if the IN= source library contains catalogs, then you must specify the SLIBREF= option. See "Using the SLIBREF= Option" on page 693.

Requirement: If CATALOG is the only member type in the library and you are using the SLIBREF= option, then omit the IN= argument.

OUT=*libref-2*
names the target SAS library to contain the migrated members.

Restriction: Do not assign the OUT= target library to a server, such as SAS/CONNECT or SAS/SHARE. The REMOTE engine is not supported for the target library. The following example causes an error:

```
libname lib1 'path to source library';
libname lib2 server=server id;
proc migrate in=lib1 out=lib2;run;
```

Requirement: Assign the OUT= target library to a different physical location than the IN= source library.

Tip: Assign the target library to an empty location. If a member already exists in the target library that has the same name and member type as a member in the source library, the member is not migrated. An error message is written to the SAS log, and PROC MIGRATE continues with the next member. Note that members in a sequential library are an exception, because PROC MIGRATE does not read the entire tape to determine existence.

Options

BUFSIZE=*n* | *n*K | *n*M | *n*G
specifies the buffer page size of the members that are written to the target library. For example, a value of **8** specifies a page size of 8 bytes, and a value of **4k** specifies a page size of 4096 bytes.

Default: the original buffer page size that was used to create the source library member

n
specifies the number of bytes.

*n*K
specifies the number of kilobytes.

*n*M
specifies the number of megabytes.

*n*G
specifies the number of gigabytes.

MOVE

deletes the original members from the source library. If a member already exists in the target library, the member is not deleted from the source library and a message is sent to the SAS log. If a catalog already exists in the target library, then no catalogs are deleted from the source library and a message is sent to the SAS log. Specifying MOVE reduces the scope of the validation tools. See "Using the MOVE Option with Validation Tools" on page 693.

Restriction: The engine that is associated with the IN= source library must support the deletion of tables. Sequential engines do not support the deletion of tables.

Tip: Use the MOVE option only if your system is space-constrained. It is preferable to verify the migration of the member before it is deleted.

SLIBREF=*libref*
specifies a libref that is assigned through a SAS/CONNECT or SAS/SHARE server. You must specify the SLIBREF= option when the processing invokes CEDA and catalogs members exist in the library. See "Using the SLIBREF= Option" on page 693.

Requirement: If the catalogs were created in SAS 6 or SAS 8, SLIBREF= must be assigned through a SAS 8 server.

Requirement: The SLIBREF= server must be running on the same kind of operating environment as the source library. For example, if the source session is running under UNIX, then the server must be running under UNIX.

Interaction: If CATALOG is the only member type in the library and you are using the SLIBREF= option, then omit the IN= argument.

Interaction: If you are migrating from a SAS 9 release or later (for example, migrating from SAS 9.1.3 to SAS 9.2), then SLIBREF= is not required. Instead, specify the SAS/CONNECT or SAS/SHARE server libref in the IN= argument.

KEEPNODUPKEY
specifies to retain the NODUPKEY sort order. See "Migrating a SAS Data Set with NODUPKEY Sort Indicator" on page 689.

Concepts: MIGRATE Procedure

What Are the Specific Considerations for Each Member Type?

Here are the considerations for each member type. Remember that a SAS data set can be a data file or a data view. See also several important restrictions at "Syntax: MIGRATE Procedure" on page 685.

Data Files
PROC MIGRATE retains compression, created and modified datetimes, deleted observations, encryption, indexes, integrity constraints, and passwords. The audit trail and generations are also migrated. Indexes and integrity constraints are rebuilt on the member in the target library. Migrated data files take on the data representation and encoding attributes of the target library. See "Migrating a Data File with Audit Trails, Generations, Indexes, or Integrity Constraints" on page 689.

Views
As with data files, migrated data views take on the data representation and encoding attributes of the target library. When you migrate a library that contains

DATA step views to a different operating environment, and the views were created before SAS 9.2, setting the proper encoding might be necessary. In releases before SAS 9.2, DATA step views did not save encoding information. Therefore, if the view has a different encoding than the target session, you must specify the INENCODING= option for the source library's LIBNAME statement. Here is an example:

```
libname srclib 'c:\mysource' inencoding="OPEN_ED-1047";
libname lib1 'c:\mytarget';
proc migrate in=srclib out=lib1;
run;
```

A note is printed to the log that recommends using the validation tools to recompile the DATA step view and determine whether the migration was successful. See "Migrating a Library with Validation Tools" on page 693.

In addition, embedded librefs associated with a view are not updated during migration. The following example illustrates the issue. In this example, LIB1.MYVIEW contains a view of the data file LIB1.MYDATA:

```
data LIB1.MYDATA;x=1;
run;

proc sql;
   create view LIB1.MYVIEW as select * from LIB1.MYDATA;
 quit;
```

After you migrate LIB1 to LIB2, you have LIB2.MYVIEW and LIB2.MYDATA. However, because LIB2.MYVIEW was originally created with an embedded libref of LIB1, it still references the data file LIB1.MYDATA, not LIB2.MYDATA. The following example fails with an error message that LIB1 cannot be found:

```
proc print data=LIB2.MYVIEW;
run;
```

PROC MIGRATE supports three types of views: DATA step views, SQL views, and SAS/ACCESS views:

DATA Step Views
: Beginning with SAS 8, when you create a DATA step view, you can specify the SOURCE= option to store the DATA step code along with the view. PROC MIGRATE supports DATA step views with stored code. The stored code is recompiled the first time the DATA step view is accessed by SAS in the target environment. PROC MIGRATE does not support DATA step views that were created before SAS 8 or DATA step views without stored code. For DATA step views without stored code, use the DESCRIBE statement in the source session to recover the DATA step code. Then submit the DATA step code in the target session and recompile it.

PROC SQL Views
: PROC MIGRATE supports PROC SQL views with no known issues.

SAS/ACCESS Views
: PROC MIGRATE supports SAS/ACCESS views that were written with the Oracle, Sybase, or DB2 engine. PROC MIGRATE automatically uses the CV2VIEW procedure, which converts SAS/ACCESS views into SQL views. Migrating SAS/ACCESS views to a different operating environment is not

supported. For more information about the conversion, see the overview of the CV2VIEW procedure in *SAS/ACCESS for Relational Databases: Reference*.

Catalogs
To migrate catalogs, PROC MIGRATE calls PROC CPORT and PROC CIMPORT. You might notice that CPORT and CIMPORT notes are written to the SAS log during migration. PROC CPORT and CIMPORT restrictions apply. For example, catalogs in sequential libraries are not migrated.

Restriction: PROC MIGRATE is not supported for SAS 6 AIX catalogs. Use PROC CPORT and PROC CIMPORT instead. See Example 5 on page 699.

Requirement: If CEDA processing is invoked, and if the source library contains catalogs, then you must specify the SLIBREF= option. If the catalogs were created in SAS 6 or SAS 8, SLIBREF= must be assigned through a SAS 8 server. See "Using the SLIBREF= Option" on page 693.

MDDBs
PROC MIGRATE supports MDDBs with no known issues.

Program Files
PROC MIGRATE does not support program files. If a program file exists in the library, a message is written to the SAS log.

Item Stores
PROC MIGRATE supports item stores unless you migrate from a 32-bit to a 64-bit environment. Migrations from 32-bit to 64-bit environments use RLS (Remote Library Services), which does not support item stores. In that case, an error message is not written to the SAS log, but item stores might not work correctly in the target library.

Migrating a Data File with Audit Trails, Generations, Indexes, or Integrity Constraints

In all cases, the data file migrates first, and then the audit trails, generations, index, or integrity constraints are applied. Errors are handled in the following ways:

- If an error occurs while an index is created for a migrated data file, the data file migrates without the index and a warning is written to the SAS log. If an index fails to migrate, resolve the error and recreate the index in the target library.
- If an error occurs while integrity constraints are applied to a migrated data file, or while an audit trail or generations are migrated, the data file is removed from the target library. A note is written to the SAS log. If the MOVE option is specified, it does not delete the data file from the source library.
- For a data file with referential integrity constraints, the MOVE option does not delete any members in the source library, even when the migration is successful. You must remove referential integrity constraints before the member can be deleted. An error message is written to the SAS log.

Migrating a SAS Data Set with NODUPKEY Sort Indicator

When you migrate a SAS data set that was sorted with the NODUPKEY option, you can either use the default behavior or specify the KEEPNODUPKEY option.

Under the default behavior (without the KEEPNODUPKEY option), the SAS data set retains its sort indicator in the target library. However, the NODUPKEY attribute is

removed, and a warning message is written to the SAS log. This is the default behavior because SAS data sets that were sorted with the NODUPKEY option in previous releases might retain observations with duplicate keys. You can re-sort the migrated SAS data set by the key variables in PROC SORT so that observations with duplicate keys are eliminated and the correct attributes are recorded.

If you specify the KEEPNODUPKEY option, you must examine your migrated data to determine whether observations with duplicate keys exist. If so, you must re-sort the SAS data set to have the data and NODUPKEY sort attribute match.

Migrating a SAS 6 Library

For SAS 6 source libraries, the following operating environments are supported. Find your source operating environment in the first column and read across the row for information about the supported target operating environments.

If your libraries are not supported, see Example 6 on page 700. If only your catalogs are not supported, see Example 5 on page 699. For important information about the SLIBREF= option, see "Using the SLIBREF= Option" on page 693. See "Examples" on page 695.

Table 37.1

Source Library	Target Library	Instructions for Libraries with Catalogs
SAS 6.12 AIX, HP-UX, or Solaris	AIX, HP-UX, or Solaris	PROC MIGRATE does not support catalogs from SAS 6 AIX. For HP-UX or Solaris libraries that contain catalogs, specify the SLIBREF= option.
SAS 6.12 Windows	32–bit Windows	Catalogs are supported. Do not use the SLIBREF= option.
SAS 6.12 Windows	64–bit Windows	For libraries that contain catalogs, specify the SLIBREF= option.
SAS 6.09E z/OS	z/OS	Catalogs are supported. Do not use the SLIBREF= option.

SAS files that were created before SAS 6.12 (SAS 6.09E for z/OS) must be converted to SAS 6.12 before they can be migrated to the current release of SAS. See also Example 6 on page 700.

Migrating SAS Data Sets that Contain Non-English Characters

For SAS data sets that use the ASCII-OEM character set, PROC MIGRATE does not translate non-English characters. To migrate a SAS data set with ASCII-OEM characters to the current release of SAS, use the CPORT and the CIMPORT procedures with the TRANTAB option. Specify the appropriate TRANTAB values for the source and target data sets. For more information about CPORT and CIMPORT, see *Base SAS Procedures Guide*.

The following example specifies **p850wlt1** and **wlt1wlt1** for the old and new translation tables, respectively. You can check the output from the PRINT procedure to determine whether the non-English characters translate correctly.

```
options trantab =(p850wlt1,wlt1wlt1) ;
libname source v6 "source-library-pathname" ;
libname target base "target-library-pathname" ;

title1 "SAS data set containing non-English characters";
title2 "source library";
proc print data=source.filename;
run;

proc cport lib=source file="souce library-pathname\filename" ;
run ;
proc cimport lib=target infile="target library-pathname\filename" ;
run ;

title2 "compare this output with output from the source library";
proc print data=target.filename ;
run ;
```

Migrating Files with Short Extensions on PC Operating Environments

In SAS 7 and 8, the SHORTFILEEXT option creates a file with a shortened, three-character extension on PC operating environments only. This feature is necessary for operating systems that use a *FAT (file allocation table)* file system. The FAT file system is also referred to as *8.3* because a filename can include up to eight characters and a file extension can include up to three characters. These files are created on PC environments. They are not usable by SAS on other environments.

Note: SAS 6 files all have three-character extensions but are not affected by this issue. You can distinguish SAS 6 files because their extensions do not contain the number 7. △

Below is a table of the short and standard extensions for SAS 7 and 8 files. To determine whether a library contains files with short extensions, look at the filenames in the SAS Explorer or use the file management tools of your operating environment.

Table 37.2 Short and Standard File Extensions for SAS 7 and 8 Files

Memtype	Short Extension	Standard Extension
ACCESS	sa7	sas7bacs
AUDIT	st7	sas7baud
CATALOG	sc7	sas7bcat
DATA	sd7	sas7bdat
DMDB	s7m	sas7bdmd
FDB	sf7	sas7bfdb
INDEX	si7	sas7bndx
ITEMSTOR	sr7	sas7bitm

Memtype	Short Extension	Standard Extension
MDDB	sm7	sas7bmdb
PROGRAM	ss7	sas7bpgm
PUTILITY	sp7	sas7bput
UTILITY	su7	sas7butl
VIEW	sv7	sas7bvew

SAS 9 Compatibility with Short-Extension Files

SAS 9.0
supports the SHORTFILEEXT option, although it is not documented. If you use this option, you have read and write access to your existing short-extension files.

SAS 9.1 and 9.1.2
do not support any access to short-extension files. You can use PROC COPY in SAS 8 to copy the short file extensions to standard file extensions before migrating your files to SAS 9.

As an alternative to PROC COPY, if you are comfortable using operating system commands or file management tools, you can manually change the file's extension to the standard extension. The content of a short-extension file is identical to the content of a standard-extension file. The use of operating system commands is not supported by SAS.

SAS 9.1.3 and later
support read-only access to existing short-extension files. You can migrate your library to the current release of SAS by using PROC MIGRATE. You must specify the SHORTFILEEXT option on the LIBNAME statement for the source library. The files are written to the target library with standard extensions; the files support full access.

For example, a library named MYLIB contains two files with short extensions: a SAS data set named **mydata.sd7** and a catalog named **mycat.sc7**. Use the following code to migrate the library to SAS 9:

```
libname mylib v8 'source-library-pathname' shortfileext;
libname newlib v9 'target-library-pathname';

proc migrate in=mylib out=newlib;
run;
```

After migration, the target library NEWLIB contains two files with standard extensions: a SAS data set named **mydata.sas7bdat** and a catalog named **mycat.sas7bcat**.

If your library also contains standard-extension files, then perform an additional migration without the SHORTFILEEXT option on the LIBNAME statement to migrate those files. Make sure that no short-extension files have the same name as a standard-extension file. In the target library, all files have a standard extension. If a short-extension file and a standard-extension file have the same name and same member type in the target library, the second one fails to migrate.

Migrating a Library with Validation Tools

Introduction

When you run PROC MIGRATE with validation tools, you generate Output Delivery System (ODS) reports that validate a successful migration. The validation tools are located on the SAS Web site at **http://support.sas.com/rnd/migration/resources/procmigrate/validtools.html**. The validation tools support a basic migration or a migration with the SLIBREF option (which uses RLS).

The validation tools consist of a set of macros and a template of example code. Some of the macros run before the migration to record the expected behavior of PROC MIGRATE. Another group of macros runs after the migration to record the actual behavior. The final group of macros compares expected and actual behavior.

Using the MOVE Option with Validation Tools

If you use the MOVE option with PROC MIGRATE, the validation tools can produce validation output only for the members that were migrated. The MOVE option deletes the source library after it has been moved to the target library. For these reasons, the MOVE option significantly limits the validation tools.

Follow these steps to use the MOVE option with validation tools:

1. Submit the migrate_macros.sas code in the interactive SAS session to compile the needed macros.
2. Copy the migrate_template.sas code into the interactive SAS session and edit the three LIBNAME statements for your pathnames.
3. Submit:

```
%before;
```

4. Submit:

```
proc migrate in=lib1 out=lib2 move;
run;
```

5. Then, submit the following code:

```
%mig_in_lib(lib=lib1, after=y);
%mig_in_lib(lib=lib2);
%mig_indexes(lib=LIB2);  /*must be upper case*/
%mig_check_libs;
%mig_check_source(move=yes);
```

Note: If a member other than a catalog already exists in the target library, the member is not deleted from the source library and a message is sent to the SAS log. If a catalog already exists in the target library, then no catalogs are deleted from the source library and a message is sent to the SAS log. △

Using the SLIBREF= Option

When to Use the SLIBREF= Option

The SLIBREF= option is required if the source library contains catalogs and the processing invokes CEDA on the target session. In general, CEDA is invoked when you migrate from a 32-bit to a 64-bit operating environment, when you migrate to a different family of operating environments, or when you migrate to an environment with an incompatible character encoding.

For SAS 8 files, you can run a test to determine whether CEDA processing will be used by PROC MIGRATE. In the target session, set MSGLEVEL=I and try processing a data set that was created under the source session. Submit something simple like the CONTENTS procedure. Check the SAS log for a message. If CEDA processing was used, then you need the SLIBREF= option.

For SAS 6 files, only use the SLIBREF= option for SAS 6 HP-UX or Solaris libraries that contain catalogs. For more information, see “Migrating a SAS 6 Library” on page 690.

If you are uncertain whether you must specify SLIBREF=, use the PROC MIGRATE Calculator on the SAS Web site at **http://support.sas.com/rnd/migration/planning/files/migratecalc/**.

For sample code, see Example 2 on page 696 and Example 4 on page 698.

When to Not Use the SLIBREF= Option

- If you are migrating from a SAS 9 release or later (for example, migrating from SAS 9.1.3 to SAS 9.2), then SLIBREF= is not required. Instead, specify the SAS/CONNECT or SAS/SHARE server libref in the IN= argument.
- Do not use the SLIBREF= option if the library contains no catalogs.
- If the library does contain catalogs, do not use the SLIBREF= option for the following source and target libraries:
  - source is 64–bit HP-UX or Solaris and target is 64–bit AIX, HP-UX, or Solaris
  - source and target are both 32–bit Windows
  - source and target are both z/OS

If you are uncertain whether you must specify SLIBREF=, use the PROC MIGRATE Calculator on the SAS Web site at **http://support.sas.com/rnd/migration/planning/files/migratecalc/**.

Requirements for the SAS/CONNECT or SAS/SHARE Server

To use the SLIBREF= option, you must have access to a SAS/CONNECT or SAS/SHARE server that is running on the same kind of operating environment as the source library. For example, if the source session is running under UNIX, then the server must be running under UNIX.

If the catalogs were created in SAS 6 or SAS 8, SLIBREF= must be assigned through a SAS 8 server. (Note that this is not the same server that you assign through the IN= argument. If you assign a server through the IN= argument, the IN= server must be SAS 9.)

If you cannot meet these requirements, use the alternate method described in Example 5 on page 699.

Restrictions for the SLIBREF= Option

If CATALOG is the only member type in the library and you are using the SLIBREF= option, then omit the IN= argument.

When you use the SLIBREF= option for a SAS 8.2 library, multilabel formats are not supported. If a catalog contains a multilabel format, the format is not created on the target and an error is printed to the log. See SAS Note 20052, which is available from SAS customer support at **http://support.sas.com**.

Examples

In SAS 9.2, migrating from one operating environment to another has been simplified. You can migrate a SAS 8.2 data library from almost every SAS 8.2 operating environment to any SAS 9.2 operating environment. Most SAS 6 operating environments are also supported, but not for cross-environment migration.

Use the PROC MIGRATE Calculator on the SAS web site at **http://support.sas.com/rnd/migration/planning/files/migratecalc/**, which provides the PROC MIGRATE syntax needed for your specific migration.

You are encouraged to run PROC MIGRATE with the validation tools. For more information, see "Migrating a Library with Validation Tools" on page 693.

Example 1: Migrating without SLIBREF= across Computers

Tip: You can use the PROC MIGRATE Calculator on the SAS web site at **http://support.sas.com/rnd/migration/planning/files/migratecalc/**, which provides the PROC MIGRATE syntax needed for your specific migration.

Tip: You are encouraged to run PROC MIGRATE with validation tools. See "Migrating a Library with Validation Tools" on page 693.

Procedure features:

MIGRATE statement options:

IN=
OUT=

In this example, the source and target libraries are on different computers, and the SLIBREF= option is not required. (See "Using the SLIBREF= Option" on page 693 to learn whether SLIBREF= is required.)

You can assign the source library to the IN= argument in one of the following two ways:

- directly via a Network File System (NFS)
- via a SAS 9 SAS/CONNECT or SAS/SHARE server.

The direct method is possible only if you can access the library via an NFS, which is a standard protocol of UNIX operating environments. If you want to use that method, see the documentation for NFS and for your operating environment.

This example uses SAS/CONNECT software. The SAS/CONNECT server that you assign to the IN= argument must be a SAS 9 server.

Program

1 From a session in the current release of SAS, submit the SIGNON command to invoke a SAS/CONNECT server session. Note that because you are working across computers, you might specify a machine name in the server ID:

```
signon serv-ID sascmd='my-sas-invocation-command';
```

2 Within this remote session, assign a libref to the source library that contains the library members to be migrated. Use the RSUBMIT and ENDRSUBMIT commands for SAS/CONNECT:

```
rsubmit;
libname source <engine> 'source-library-pathname';
endrsubmit;
```

3 In the local (client) session in the current release, assign the same source libref as in step 2. But do not assign the libref to a physical location. Instead, specify the SERVER= option with the server ID (in this example, *serv-ID*) that you assigned in the SIGNON command in step 1:

```
libname source <engine> server=serv-ID;
```

4 Assign the target library:

```
libname target <engine> 'target-library-pathname';
```

5 Use PROC MIGRATE:

```
proc migrate in=source out=target <options>;
run;
```

If your library contains catalogs, see Example 2 on page 696.

Example 2: Migrating with SLIBREF= across Computers

Tip: You can use the PROC MIGRATE Calculator on the SAS web site at **http://support.sas.com/rnd/migration/planning/files/migratecalc/**, which provides the PROC MIGRATE syntax needed for your specific migration.

Tip: You are encouraged to run PROC MIGRATE with validation tools. See "Migrating a Library with Validation Tools" on page 693.

Procedure features:

MIGRATE statement options:

IN=
OUT=
SLIBREF=

In this example, the source and target libraries are on different computers. Because the source library contains catalogs that are incompatible with the current release of SAS, the SLIBREF= option is required. (See "Using the SLIBREF= Option" on page 693 to learn whether SLIBREF= is required.)

You can assign the source library (except the catalogs) to the IN= argument in one of the following two ways:

□ directly via a Network File System (NFS)

□ via a SAS/CONNECT or SAS/SHARE server.

This example uses NFS, which is a standard protocol of UNIX operating environments. See the documentation for NFS and for your operating environment. If you are not in a UNIX environment, or if you wish to use a SAS/CONNECT or SAS/SHARE server to access the source library, please see Example 1 for instructions.

The example uses the SLIBREF= argument to access the catalogs in the source library. The SLIBREF= argument must be assigned to a SAS/CONNECT or SAS/SHARE server running in a session of SAS that can access the catalogs. For example, if the source library contains SAS 8.2 catalogs created by 32-bit Solaris, SLIBREF= must be assigned to a SAS 8.2 32-bit Solaris server.

Program

In this example, the source library was created in SAS 8.2. If catalogs were created in SAS 6 or SAS 8.2, SLIBREF= must be assigned through a SAS 8.2 server.

1. From a session in the current release of SAS, submit the SIGNON command to invoke a SAS/CONNECT server session. Note that because you are working across computers, you might specify a machine name in the server ID:

```
signon v8srv sascmd='my-v8--sas-invocation-command';
```

2. Within this remote SAS 8.2 session, assign a libref to the source library that contains the library members to be migrated. Use the RSUBMIT and ENDRSUBMIT commands for SAS/CONNECT:

```
rsubmit;
libname srclib <engine> 'source-library-pathname';
endrsubmit;
```

3. In the local (client) session in the current release, assign to the same source library through NFS:

```
libname source <engine> '/nfs/v8machine-name/source-library-pathname';
```

4. Assign the same libref to the same source libref as in step 2 (in this example, SRCLIB). But do not assign the libref to a physical location. Instead, specify the SERVER= option with the server ID (in this example, V8SRV) that you assigned in the SIGNON command in step 1:

```
libname srclib <engine> server=v8srv;
```

5. Assign the target library:

```
libname target <engine> 'target-library-pathname';
```

6. Use PROC MIGRATE with the SLIBREF= option. For the IN= and OUT= options, specify the usual source and target librefs (in this example, SOURCE and TARGET, respectively). Set SLIBREF= to the libref that uses the SERVER= option (in this example, SRCLIB):

```
proc migrate in=source out=target slibref=srclib <options>;
run;
```

   If CATALOG is the only member type in the library and you are using the SLIBREF= option, then omit the IN= argument:

```
proc migrate out=target slibref=srclib <options>;
run;
```

Example 3: Migrating without SLIBREF= on the Same Computer

Tip: You can use the PROC MIGRATE Calculator on the SAS web site at **http://support.sas.com/rnd/migration/planning/files/migratecalc/**, which provides the PROC MIGRATE syntax needed for your specific migration.

Tip: You are encouraged to run PROC MIGRATE with validation tools. See "Migrating a Library with Validation Tools" on page 693.

Procedure features:

MIGRATE statement options:

IN=
OUT=

In this example, the source and target libraries are on one computer, and the SLIBREF= option is not required. (See "Using the SLIBREF= Option" on page 693 to learn whether SLIBREF= is required.) Because the source and target are on one computer, this example assigns the source library to the IN= argument directly.

Program

```
libname source <engine> 'source-library-pathname';
libname target base 'target-library-pathname-on-same-computer';

proc migrate in=source out=target;
run;
```

Example 4: Migrating with SLIBREF= on the Same Computer

Tip: You can use the PROC MIGRATE Calculator on the SAS web site at **http://support.sas.com/rnd/migration/planning/files/migratecalc/**, which provides the PROC MIGRATE syntax needed for your specific migration.

Tip: You are encouraged to run PROC MIGRATE with validation tools. See "Migrating a Library with Validation Tools" on page 693.

Procedure features:

MIGRATE statement options:

IN=
OUT=
SLIBREF=

In this example, the source and target libraries are on one computer. Because the source library contains catalogs that are incompatible with the current release of SAS, the SLIBREF= option is required. (See "Using the SLIBREF= Option" on page 693 for important information and to learn whether SLIBREF= is required.)

Because the source and target are on one computer, this example assigns the source library (except the catalogs) to the IN= argument directly.

The example uses the SLIBREF= argument to access the catalogs in the source library. The SLIBREF= argument must be assigned to a SAS/CONNECT or SAS/SHARE server running in a session of SAS that can access the catalogs. For example, if the source library contains SAS 8.2 catalogs created by 32-bit Solaris, SLIBREF= must be assigned to a SAS 8.2 32-bit Solaris server.

Program

In this example, the source library was created in SAS 8.2. If catalogs were created in SAS 6 or SAS 8.2, SLIBREF= must be assigned through a SAS 8.2 server.

1. From a session in the current release of SAS, submit the SIGNON command to invoke a SAS/CONNECT server session:

```
signon v8srv sascmd='my-v8--sas-invocation-command';
```

2. Within this remote SAS 8.2 session, assign a libref to the source library that contains the library members to be migrated. Use the RSUBMIT and ENDRSUBMIT commands for SAS/CONNECT:

```
rsubmit;
libname srclib <engine> 'source-library-pathname';
endrsubmit;
```

3. In the local (client) session in the current release, assign to the same source library as in step 2. But do not assign the libref to a physical location. Instead, specify the SERVER= option with the server ID (in this example, V8SRV) that you assigned in the SIGNON command in step 1:

```
libname srclib <engine> server=v8srv;
```

4. Assign two librefs: one to the source library and another to the target library. The source library is the same one that is assigned in step 2, but you must use a different libref:

```
libname source <engine> 'source-library-pathname;
libname target <engine> 'target-library-pathname';
```

5. Use PROC MIGRATE with the SLIBREF= option. For the IN= and OUT= options, specify the librefs that you assigned in step 4 (in this example, SOURCE and TARGET, respectively). Set SLIBREF= to the libref that uses the SERVER= option (in this example, SRCLIB):

```
proc migrate in=source out=target slibref=srclib <options>;
run;
```

   If CATALOG is the only member type in the library and you are using the SLIBREF= option, then omit the IN= argument:

```
proc migrate out=target slibref=srclib <options>;
run;
```

Example 5: Additional Steps for Unsupported Catalogs

PROC MIGRATE is not supported for migrating catalogs under the following circumstances:

- SAS 6 AIX catalogs to any target library.
- Tru64 UNIX catalogs to either Linux for x64 or Solaris for x64 target library.

- any catalogs if you must use SLIBREF but you do not have access to either SAS/CONNECT or SAS/SHARE software. See "Using the SLIBREF= Option" on page 693 to learn whether SLIBREF= is required.

The following process converts catalogs so that they are native with the target environment and with the current release of SAS:

1. In the source session, create a transport file with PROC CPORTChapter 15, "The CPORT Procedure," on page 269.
2. Move the transport file to the target environment. Do not use RLS (a feature of SAS/CONNECT and SAS/SHARE software) to move catalogs, or you will encounter errors. You must use binary FTP, the DOWNLOAD procedure, NFS (Network File System), or another method of directly accessing files.
3. In the target session, use PROC CIMPORTChapter 10, "The CIMPORT Procedure," on page 191 to import the transport file.

You can use this process for just the catalogs or for all members of a library. You might want to limit this method to catalogs for the following reasons:

- FTP can be time-consuming for large libraries.
- By using PROC MIGRATE for other members of the library, you can retain those members' attributes and use validation tools.

For more information, see *Moving and Accessing SAS Files*.

Example 6: Alternatives to PROC MIGRATE

PROC MIGRATE is not supported for libraries that were created under the following source environments:

- SAS 8.2 CMS, OS/2, OpenVMS VAX, or 64–bit AIX.
- any unsupported SAS 6 operating environment. For a list of supported SAS 6 operating environments, see "Migrating a SAS 6 Library" on page 690.

To migrate across operating environments that cannot use PROC MIGRATE, use the following process:

1. Under the source installation of SAS, use one of the conversion procedures, PROC COPY, PROC CPORT or PROC UPLOAD, to create a transport file containing your source library. See *Moving and Accessing SAS Files*.
2. Under the target installation of SAS, use PROC COPY, PROC CIMPORT or PROC DOWNLOAD to convert the transport file to a SAS 9 library.

CHAPTER
38

The OPTIONS Procedure

Overview: OPTIONS Procedure **701**
What Does the OPTIONS Procedure Do? **701**
What Types of Output Does PROC OPTIONS Produce? **702**
Syntax: OPTIONS Procedure **707**
PROC OPTIONS Statement **707**
Results: OPTIONS Procedure **710**
Examples: OPTIONS Procedure **710**
Example 1: Producing the Short Form of the Options Listing **710**
Example 2: Displaying the Setting of a Single Option **711**
Example 3: Displaying Expanded Path Environment Variables **712**

Overview: OPTIONS Procedure

What Does the OPTIONS Procedure Do?

The OPTIONS procedure lists the current settings of SAS system options in the SAS log.

SAS system options control how SAS formats output, handles files, processes data sets, interacts with the operating environment, and does other tasks that are not specific to a single SAS program or data set. You can change the settings of SAS system options by using one of the following methods:

- the SAS command
- the option in a configuration or autoexec file
- the SAS OPTIONS statement
- the OPTLOAD and OPTSAVE procedures
- the SAS System Options window
- the DMOPTSAVE and DMOPTLOAD commands
- 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?

Displaying a List of System Options

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 38.1 Log Showing a Partial Listing of SAS System Options

```
Portable Options:
APPLETLOC=site-specific-llocation
                            Location of Java applets
ARMAGENT=         ARM Agent to use to collect ARM records
 ARMLOC=ARMLOG.LOG Identify location where ARM records are to be written
 ARMSUBSYS=(ARM_NONE)
                   Enable/Disable ARMing of SAS subsystems
 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
 BINDING=DEFAULT   Controls the binding edge for duplexed output
 BOMFILE           Add Byte Order Mark when creating Unicode files
 BOTTOMMARGIN=0.000 IN
                   Bottom margin for printed output
 BUFNO=1           Number of buffers for each SAS data set
 BUFSIZE=0         Size of buffer for page of SAS data set
 BYERR             Set the error flag if a null data set is input to the SORT procedure
 BYLINE            Print the by-line at the beginning of each by-group
 BYSORTED          Require SAS data set observations to be sorted for BY processing
 NOCAPS            Do not translate source input to uppercase
 NOCARDIMAGE       Do not process SAS source and data lines as 80-byte records
 CATCACHE=0        Number of SAS catalogs to keep in cache memory
 CBUFNO=0          Number of buffers to use for each SAS catalog
 CENTER            Center SAS procedure output
 CGOPTIMIZE=3      Control code generation optimization
 NOCHARCODE        Do not use character combinations as substitute for special characters not on the keyboard
 CLEANUP           Attempt recovery from out-of-resources condition
 NOCMDMAC          Do not support command-style macros
 CMPLIB=           Identify previously compiled libraries of CMP subroutines to use  when linking
 CMPMODEL=BOTH     Identify CMP model storage type
 CMPOPT=(NOEXTRAMATH NOMISSCHECK NOPRECISE NOGUARDCHECK NOFUNCDIFFERENCING)
                   Enable SAS compiler performance optimizations
```

Displaying Information about a Single Option

To view the setting of a particular option, you can use the OPTION= and DEFINE options on the PROC OPTIONS statement. The following example shows a log that PROC OPTIONS produces for a single SAS system option.

```
options pagesize=60;
proc options option=pagesize define;
```

```
run;
```

Output 38.2 The Setting of a Single SAS System Option

```
8    options pagesize=60;
9    proc options option=pagesize define;
10   run;

    SAS (r) Proprietary Software Release xxx

 PAGESIZE=60
Option Definition Information for SAS Option PAGESIZE
    Group= LOGCONTROL
    Group Description: LOGCONTROL
    Group= LISTCONTROL
    Group Description: Procedure output and display settings
    Group= LOG_LISTCONTROL
    Group Description: SAS log and procedure output settings
    Description: Number of lines printed per page of output
    Type: The option value is of type LONG
          Range of Values: The minimum is 15 and the maximum is 32767
          Valid Syntax(any casing): MIN|MAX|n|nK|nM|nG|nT|hex
    Numeric Format: Usage of LOGNUMBERFORMAT does not impact the value format
    When Can Set: Startup or anytime during the SAS Session
    Restricted: Your Site Administrator cannot restrict modification of this option
    Optsave: Proc Optsave or command Dmoptsave will save this option
```

Displaying Information About System Option Groups

Each SAS system option belongs to one or more groups, which are based on functionality, such as error handling or sorting. You can display a list of system-option groups and the system options that belong to one or more of the groups.

Use the LISTGROUPS option to display a list of system-option groups.

```
proc options listgroups;
run;
```

Output 38.3 Partial List of SAS System Option Groups

```
11   options pagesize=60;
12   proc options listgroups;
13   run;

    SAS (r) Proprietary Software Release xxx

    Option Groups
       GROUP=COMMUNICATIONS         Networking and encryption

       GROUP=ENVDISPLAY             Display

       GROUP=ERRORHANDLING          Error handling

       GROUP=ENVFILES               Files

       GROUP=LANGUAGECONTROL        Language control

       GROUP=EXECMODES              Initialization and operation

       GROUP=EXTFILES               External files

       GROUP=SASFILES               SAS Files

       GROUP=INPUTCONTROL           Data Processing

       GROUP=GRAPHICS               Driver settings

       GROUP=LOGCONTROL             SAS log

       GROUP=LISTCONTROL            Procedure output

       GROUP=LOG_LISTCONTROL        SAS log and procedure output
```

Use the GROUP= option to display system options that belong to a particular group. You can specify one or more groups.

```
proc options group=(sort memory;)
run;
```

Output 38.4 Sample Output Using the GROUP= Option

```
33   proc options group=(sort memory);
34   run;

    SAS (r) Proprietary Software Release XXX

Group=SORT
 SORTDUP=PHYSICAL  SORT applies NODUP option to physical or logical records?
 SORTEQUALS        Maintain order for the input data set in the output data set, when processing
                   identical BY-variable values with Proc Sort
 SORTSEQ=          Collating sequence for sorting
 SORTSIZE=0        Size parameter for sort
 NOSORTVALIDATE    Do not use automatic sort order validation to determine assertion
 SORTANOM=         Host sort option
 SORTCUT=0         Specifies the number of observations above which the host sort program is
                   used instead of the SAS sort program
 SORTCUTP=0        Specifies the number of bytes above which the host sort program should be
                   used instead of the SAS sort program
 SORTDEV=          Specifies the pathname used for temporary host sort files
 SORTPARM=         Specifies the host sort parameters
 SORTPGM=BEST      Specifies the name of the sort utility
Group=MEMORY
 SORTSIZE=0        Size parameter for sort
 SUMSIZE=0         Upper limit for data-dependent memory usage during summarization
 MAXMEMQUERY=0     Maximum amount of memory returned when inquiring as to available space
 MEMBLKSZ=16777216 Size of memory blocks allocated to support MEMLIB and MEMCACHE options.
 MEMMAXSZ=2147483648
                   Maximum amount of memory allocated to support MEMLIB and MEMCACHE options.
 LOADMEMSIZE=0     Suggested memory limit for loaded SAS executables
 MEMSIZE=100663296 Specifies the limit on the total amount of memory to be used by the SAS System
 REALMEMSIZE=0     Limit on the total amount of real memory to be used by the SAS SystemGroup=SORT
```

The following table lists the values that are available in all operating environments when you use the GROUP= option with PROC OPTIONS.

Values for Use with GROUP=		
CODEGEN	HELP	META
COMMUNICATIONS	INPUTCONTROL	ODSPRINT
DATAQUALITY	INSTALL	PDF
EMAIL	LANGUAGECONTROL	PERFORMANCE
ENVDISPLAY	LISTCONTROL	SASFILES
ENVFILES	LOG_LISTCONTROL	SECURITY
ERRORHANDLING	LOGCONTROL	SORT
EXECMODES	MEMORY	SVG
EXTFILES	MACRO	
GRAPHICS		

The following table lists operating environment–specific values that might be available when you use the GROUP= option with PROC OPTIONS.

Possible Operating Environment–Specific Values for Use with GROUP=			
ADABAS	DB2	ISPF	SMF
CODEGEN	IDMS	ORACLE	SPF
DATACOM	IMS	REXX	

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)*>;

Task	Statement
List the current system option settings to the SAS Log	"PROC OPTIONS Statement" on page 707

PROC OPTIONS Statement

PROC OPTIONS <*option(s)*>;

Task	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
Display system option character values as hexadecimal values	HEXVALUE
When displaying a path, replace an environment variable with its value	EXPAND
When displaying a path, display the environment variable(s) the option was defined with	NOEXPAND
Display numeric system option values using commas	LOGNUMBERFORMAT
Displays numeric system option values without using commas	NOLOGNUMBERFORMAT
Restrict the number of options displayed	
Display options belonging to one or more groups	GROUP=
Display host options only	HOST
Display portable options only	NOHOST | PORT
Display a single option	OPTION=

Task	Option
Display restricted options only	RESTRICT
Display groups and group descriptions	LISTGROUPS

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 is ignored when SHORT is specified.

Featured in: Example 2 on page 711

EXPAND | NOEXPAND
specifies whether to replace an environment variable in a path with the value of the environment variable.

EXPAND
displays the path using the value of the environment variable.

NOEXPAND
displays the path using the environment variable.

Restriction: Variable expansion is valid only in the Windows and UNIX operating environments.

Tip: By default, some option values are displayed with expanded variables. Other options require the EXPAND option on the PROC OPTIONS statement. Use the DEFINE option on the PROC OPTIONS statement to determine whether an option value expands variables by default or if the EXPAND option is required.

Featured in: Example 3 on page 712

GROUP=*group-name*
GROUP=(*group-name–1 ... group-name-n*)
displays the options in one or more groups specified by *group-name*.

Interaction: This option is ignored when OPTION= is specified.

Requirement: When you specify more than one group, enclose the group names in parenthesis and separate the group names by a space.

See also: “Displaying Information About System Option Groups” on page 703

HEXVALUE
displays system option character values as hexadecimal values.

HOST | NOHOST
specifies whether to display only host options or only portable options.

HOST
display only host options.

NOHOST
display only portable options.

Alias: PORTABLE or PORT is an alias for NOHOST.

LISTGROUPS

lists the system option groups for all operating environments and a description of each group.

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 710

LOGNUMBERFORMAT | NOLOGNUMBERFORMAT
specifies whether to display numeric system option values using commas.

LOGNUMBERFORMAT
displays numeric system option values using commas.

NOLOGNUMBERFORMAT
displays numeric system option values without using commas.

Featured in: Example 2 on page 711

NOEXPAND
See EXPAND | NOEXPAND

NOHOST | PORT
See HOST | NOHOST on page 708.

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 equal sign, such as PAGESIZE=, do not include the equal sign when specifying the option to OPTION=.

Featured in: Example 2 on page 711

NOLOGNUMBERFORMAT
See LOGNUMBERFORMAT | NOLOGNUMBERFORMAT

RESTRICT
displays the system options that have been set by your site administrator in a restricted options configuration file. These options cannot be changed by the user. For each option that is restricted, the RESTRICT option displays the option's value, scope, and how it was set.

If your site administrator has not restricted any options, then the following message appears in the SAS log:

```
Your Site Administrator has not restricted any SAS options.
```

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.

Featured in: Example 2 on page 711

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. 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 701.

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)

```
NOTE: PROCEDURE PRINTTO used (Total process time):
      real time           0.10 seconds
      cpu time            0.00 seconds

6    proc options short;
7    run;
    SAS (r) Proprietary Software Release xxx  TS1B0

Portable Options:

APPEND= APPLETLOC=your-directory ARMAGENT= ARMLOC=ARMLOG.LOG
ARMSUBSYS=(ARM_NONE) NOASYNCHIO AUTOSAVELOC= NOAUTOSIGNON BINDING=DEFAULT BOMFILE
BOTTOMMARGIN=0.000 IN BUFNO=1 BUFSIZE=0 BYERR BYLINE BYSORTED NOCAPS NOCARDIMAGE CATCACHE=0
CBUFNO=0 CENTER NOCHARCODE CLEANUP NOCMDMAC CMPLIB= CMPMODEL=BOTH CMPOPT=(NOEXTRAMATH
NOMISSCHECK NOPRECISE NOGUARDCHECK NOFUNCDIFFERENCING) NOCOLLATE COLORPRINTING COMAMID=TCP
COMPRESS=NO CONNECTPERSIST CONNECTREMOTE= CONNECTSTATUS CONNECTWAIT COPIES=1 CPUCOUNT=2 CPUID
DATASTMTCHK=COREKEYWORDS DATE DATESTYLE=MDY NODBIDIRECTEXEC DBSLICEPARM=(THREADED_APPS, 2)
DBSRVTP=NONE DEFLATION=6 NODETAILS DEVICE= DFLANG=ENGLISH DKRICOND=ERROR DKROCOND=WARN
DLDMGACTION=REPAIR NODMR DMS DMSEXP DMSLOGSIZE=99999 DMSOUTSIZE=99999 DMSPGMLINESIZE=136
NODMSSYNCHK DQLOCALE= DQSETUPLOC= DSNFERR NODTRESET NODUPLEX NOECHOAUTO EMAILAUTHPROTOCOL=NONE
NOEMAILFROM EMAILHOST=LOCALHOST EMAILID= EMAILPORT=25 EMAILPW=xxxxxxxx ENGINE=V9 NOERRORABEND
NOERRORBYABEND ERRORCHECK=NORMAL ERRORS=20 NOEXPLORER FILESYNC=SAS FIRSTOBS=1 FMTERR
FMTSEARCH=(WORK LIBRARY) FONTEMBEDDING FONTRENDERING=FREETYPE_POINTS FONTSLOC=C:\V9setup\font
FORMCHAR=            +=|-/\<>* FORMDLIM= FORMS=DEFAULT GSTYLE GWINDOW HELPENCMD
HELPINDEX=(/help/common.hlp/index.txt /help/common.hlp/keywords.htm common.hhk)
HELPTOC=(/help/helpnav.hlp/navigation.xml /help/common.hlp/toc.htm common.hhc) IBUFNO=0
IBUFSIZE=0 NOIMPLMAC INITCMD= INITSTMT= INVALIDDATA=. NOIPADDRESS JPEGQUALITY=75 LABEL
LEFTMARGIN=0.000 IN LINESIZE=97 LOGAPPLNAME= LOGCONFIGLOC= LOGPARM=WRITE=BUFFERED ROLLOVER=NONE
OPEN=REPLACE LRECL=256 MACRO MAPS=( 'your-directory' )
NOMAUTOLOCDISPLAY

Host Options:

 ACCESSIBILITY=STANDARD ALTLOG= ALTPRINT= AUTHPROVIDERDOMAIN= AUTHSERVER=
AUTOEXEC= AWSCONTROL=(SYSTEMMENU MINMAX TITLE) AWSDEF=0 0 100 100 AWSMENU
AWSMENUMERGE AWSTITLE= COMAUX1= COMAUX2= COMDEF=(BOTTOM CENTER)
CONFIG=C:\Program Files\SAS\SASFoundation\9.2\SASv9.cfg NODBCS DBCSLANG=NONE DBCSTYPE=WINDOWS
ECHO= EMAILDLG=NATIVE EMAILSYS=MAPI ENCODING=WLATIN1 ENHANCEDEDITOR
FILELOCKWAITMAX=600 FILTERLIST= FONT= FONTALIAS= NOFULLSTIMER HELPLOC=(
    "!sasuser\classdoc"
"your-directory"           ) HELPREGISTER=
HOSTPRINT NOICON JREOPTIONS=(
-Dsas.jre.libjvm=your-directory
-Djava.security.policy=your-directory
-Djava.security.auth.login.config=C:\SASv9\sas\dev\mva\DNT\misc\sas.login.config
-Djava.class.path=your-directory
-Djava.awt.headless=yes
-Djava.system.class.loader=com.sas.app.AppClassLoader
-Dsas.app.class.path=your-directory
-Dtkj.app.class.dirs=your-directories
-Dsas.ext.config=your-config-file
-DPFS_TEMPLATE=your-xml file
) LOADMEMSIZE=0 LOCALE=ENGLISH_UNITEDSTATES LOG= MAXMEMQUERY=0
MEMBLKSZ=16777216 MEMCACHE=0 NOMEMLIB MEMMAXSZ=2147483648 MEMSIZE=100663296
```

Example 2: Displaying the Setting of a Single Option

Procedure features:

PROC OPTIONS statement option:

OPTION=
DEFINE

LOGNUMBERFORMAT
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 MEMBLKSZ. The DEFINE and VALUE options display additional information. The LOGNUMBERFORMAT displays the value using commas.

Program

Specify the MEMBLKSZ SAS system option. OPTION=MEMBLKSZ displays option value information. DEFINE and VALUE display additional information. LOGNUMBERFORMAT specifies to format the value using commas.

```
proc options option=memblksz define value lognumberformat;
run;
```

Output 38.5 Log Output from Specifying the MEMBLKSZ Option

```
30   proc options option=memblksz lognumberformat define value;
31   run;

    SAS (r) Proprietary Software Release XXX

Option Value Information For SAS Option MEMBLKSZ
    Option Value: 16,777,216
    Option Scope: Default
    How option value set:  Shipped Default
Option Definition Information for SAS Option MEMBLKSZ
    Group= MEMORY
    Group Description: Memory settings
    Description: Size of memory blocks allocated to support MEMLIB and MEMCACHE options.
    Type: The option value is of type INTMAX
          Range of Values: The minimum is 0 and the maximum is 9223372036854775807
          Valid Syntax(any casing): MIN|MAX|n|nK|nM|nG|nT|hex
    Numeric Format: Usage of LOGNUMBERFORMAT does not impact the value format
    When Can Set: Session startup (command line or config) only
    Restricted: Your Site Administrator can restrict modification of this option
    Optsave: Proc Optsave or command Dmoptsave will not save this option
    SAS Language: Can "get" the option value using SAS language
    SAS Language: Can "set" the option value using SAS language
    Print or Display: Special keyword is NOT required
```

Example 3: Displaying Expanded Path Environment Variables

Procedure features:

PROC OPTIONS statement options:

OPTION=
EXPAND

NOEXPAND
HOST

This example shows the value of an environment variable when the path is displayed.

Show the value of envirionment variables: The EXPAND option causes the values of environment variables to display in place of the environment variable. The NOEXPAND option causes the environment variable to display. In this example, the environment variable is !sasroot.

```
proc options option=msg expand host;
run;
proc options option=msg noexpand host;
run;
```

Output 38.6 Displaying an Expanded and Nonexpanded Pathname using the OPTIONS Procedure

```
NOTE: PROCEDURE PRINTTO used (Total process time):
      real time           0.03 seconds
      cpu time            0.00 seconds

6    proc options option=msg expand host;
7    run;
    SAS (r) Proprietary Software Release xxx

 MSG=( 'C:\Program File\SAS\SAS 9.2\sasmsg' )
                  The path to the sasmsg directory
NOTE: PROCEDURE OPTIONS used (Total process time):
      real time           0.01 seconds
      cpu time            0.00 seconds

8    proc options option=msg noexpand host;
9    run;
    SAS (r) Proprietary Software Release 9.2  TS1B0

 MSG=( '!sasroot\sasmsg' )
                  The path to the sasmsg directory
NOTE: PROCEDURE OPTIONS used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds

10   proc printto; run;
```

CHAPTER
39

The OPTLOAD Procedure

Overview: OPTLOAD Procedure **715**
What Does the OPTLOAD Procedure Do? **715**
Syntax: OPTLOAD Procedure **715**
PROC OPTLOAD Statement **716**

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 one of these methods:

- the DMOPTLOAD command from a command line in the SAS windowing environment. For example, DMOPTLOAD key= “core\options”.
- the PROC OPTLOAD statement

When an option is restricted by the site administrator, and the option value that is being set by PROC OPTLOAD differs from the option value that was established by the site administrator, SAS issues a Warning message to the log.

Some SAS options will not be saved with PROC OPTSAVE and therefore cannot be loaded with the PROC OPTLOAD statement. 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*>;

Task	Statement
Enables SAS system options that are stored in the SAS registry or in a SAS data set	"PROC OPTLOAD Statement" on page 716

PROC OPTLOAD Statement

PROC OPTLOAD <*options*>;

Task	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

40

The OPTSAVE Procedure

Overview: OPTSAVE Procedure **717**
What Does the OPTSAVE Procedure Do? **717**
Syntax: OPTSAVE Procedure **717**
PROC OPTSAVE Statement **718**

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 one of these methods:

- the DMOPTSAVE command from a command line in the SAS windowing environment. Use the command like this: DMOPTSAVE <*save-location*>.
- the PROC OPTSAVE statement.

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

PROC OPTSAVE <*options* >;

Task	Statement
Saves the current SAS system option settings to the SAS registry or to a SAS data set	"PROC OPTSAVE Statement" on page 718

PROC OPTSAVE Statement

PROC OPTSAVE <*options* >;

Task	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

41

The PLOT Procedure

Overview: PLOT Procedure **720**
Syntax: PLOT Procedure **722**
PROC PLOT Statement **723**
BY Statement **726**
PLOT Statement **726**
Concepts: PLOT Procedure **738**
RUN Groups **738**
Generating Data with Program Statements **738**
Labeling Plot Points with Values of a Variable **739**
Pointer Symbols **739**
Understanding Penalties **740**
Changing Penalties **741**
Collision States **741**
Reference Lines **742**
Hidden Label Characters **742**
Overlaying Label Plots **742**
Computational Resources Used for Label Plots **742**
Time **742**
Memory **742**
Results: PLOT Procedure **743**
Scale of the Axes **743**
Printed Output **743**
ODS Table Names **743**
Portability of ODS Output with PROC PLOT **743**
Missing Values **744**
Hidden Observations **744**
Examples: PLOT Procedure **744**
Example 1: Specifying a Plotting Symbol **744**
Example 2: Controlling the Horizontal Axis and Adding a Reference Line **746**
Example 3: Overlaying Two Plots **748**
Example 4: Producing Multiple Plots per Page **749**
Example 5: Plotting Data on a Logarithmic Scale **752**
Example 6: Plotting Date Values on an Axis **753**
Example 7: Producing a Contour Plot **755**
Example 8: Plotting BY Groups **758**
Example 9: Adding Labels to a Plot **761**
Example 10: Excluding Observations That Have Missing Values **764**
Example 11: Adjusting Labels on a Plot with the PLACEMENT= Option **766**
Example 12: Adjusting Labeling on a Plot with a Macro **770**
Example 13: Changing a Default Penalty **772**

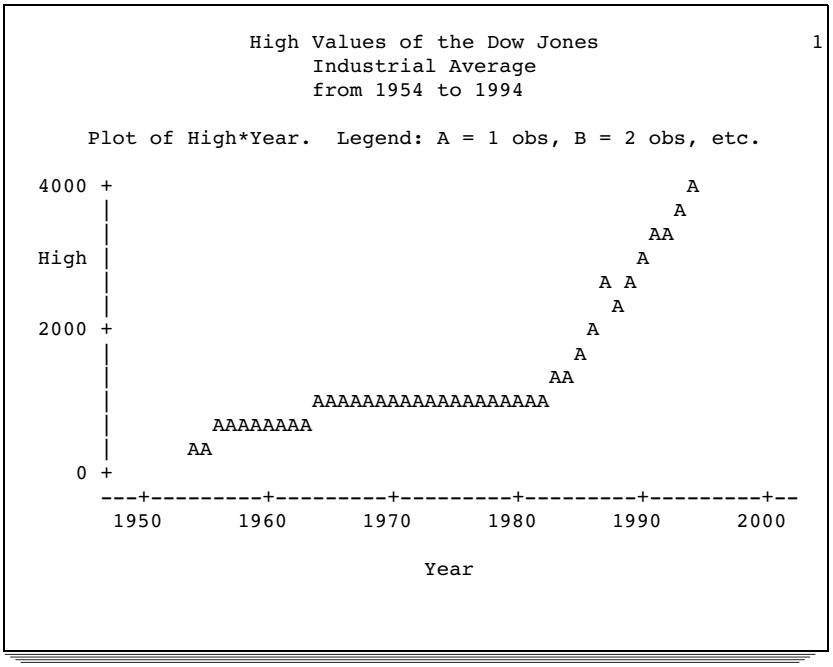
Overview: PLOT Procedure

The PLOT procedure plots the values of two variables for each observation in an input SAS data set. The coordinates of each point on the plot correspond to the two variables' values in one or more observations of the input data set.

The following output is a simple plot of the high values of the Dow Jones Industrial Average (DJIA) between 1954 and 1994. PROC PLOT determines the plotting symbol and the scales for the axes. Here are the statements that produce the output:

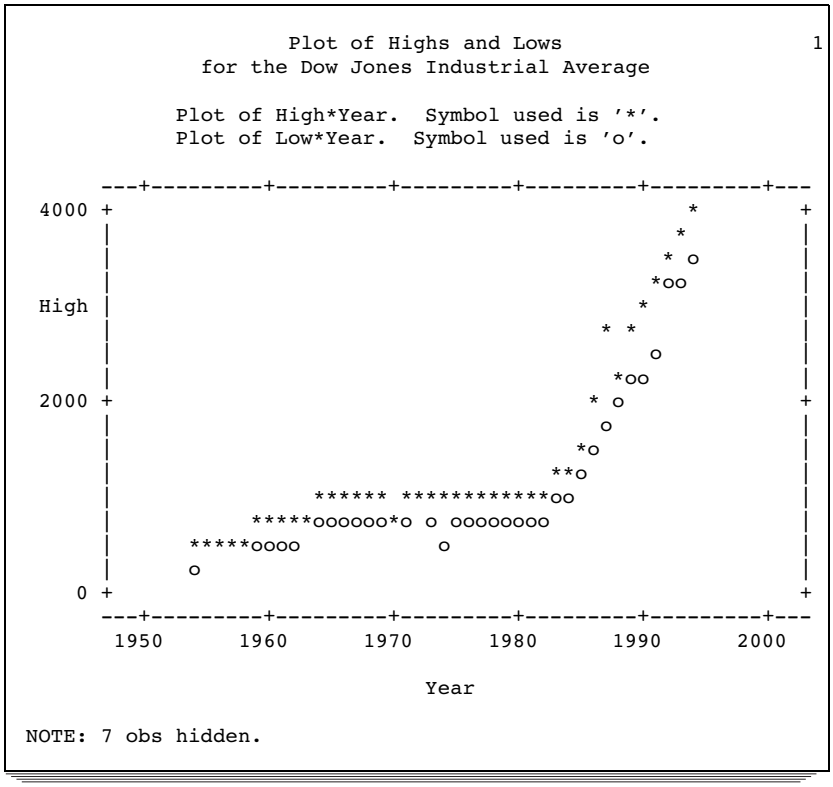
```
options nodate pageno=1 linesize=64
   pagesize=25;

proc plot data=djia;
   plot high*year;
   title 'High Values of the Dow Jones';
   title2 'Industrial Average';
   title3 'from 1954 to 1994';
run;
```

Output 41.1 A Simple Plot



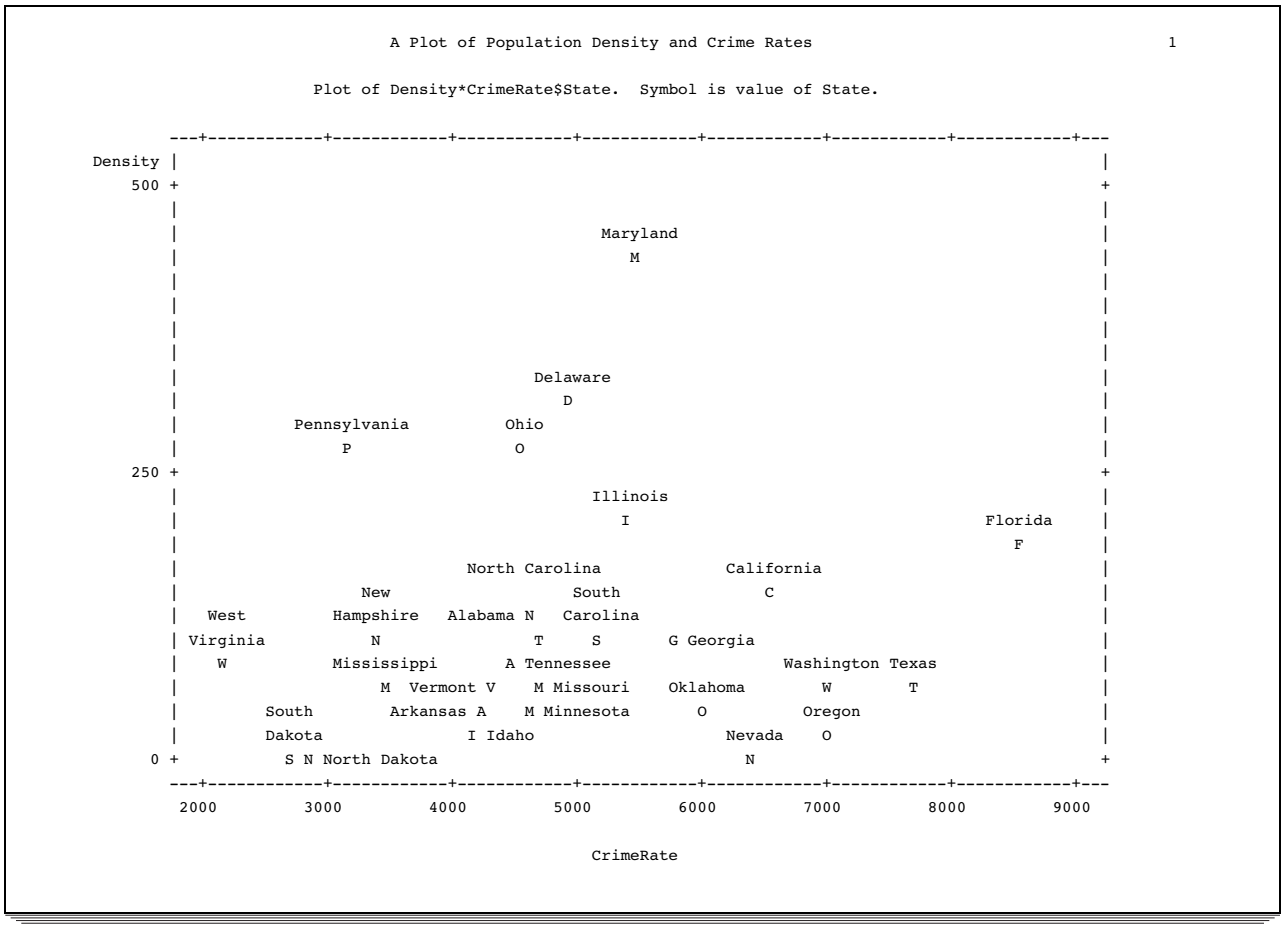
You can also overlay two plots, as shown in the following output. One plot shows the high values of the DJIA; the other plot shows the low values. The plot also shows that you can specify plotting symbols and put a box around a plot. The statements that produce the following output are shown in Example 3 on page 748.

Output 41.2 Plotting Two Sets of Values at Once



PROC PLOT can also label points on a plot with the values of a variable, as shown in the following output. The plotted data represents population density and crime rates for selected U.S. states. The SAS code that produces the following output is shown in Example 11 on page 766.

Output 41.3 Labeling Points on a Plot



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: Basic Concepts in *SAS Output Delivery System: User's Guide* for details.

Tip: You can use the ATTRIB, FORMAT, LABEL, and WHERE statements. See Chapter 3, "Statements with the Same Function in Multiple Procedures," on page 35 for details. You can also use any global statements. See "Global Statements" on page 20 for a list.

PROC PLOT <*option(s)*>;

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

PLOT *plot-request(s)* </ *option(s)*>;

Task	Statement
Requests the plots be produced	"PROC PLOT Statement" on page 723
Produce a separate plot for each BY group	"BY Statement" on page 726
Describe the plots you want	"PLOT Statement" on page 726

PROC PLOT Statement

Reminder: You can use data set options with the DATA= option. See "Data Set Options" on page 19 for a list.

PROC PLOT <*option(s)*>;

Task	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 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.

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

```
formchar(3,7)='2D7C'x
```

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 749

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 764

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 749

See also: HPERCENT= on page 724

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 729 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 36
Featured in: Example 8 on page 758

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)*>;

Task	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=
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 739 for more information. In addition, see Example 9 on page 761 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 748

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 755

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 752 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 746, Example 5 on page 752, and Example 6 on page 753

HEXPAND
expands the horizontal axis to minimize the margins at the sides of the plot and to maximize the distance between tick marks, if possible.

HEXPAND causes PROC PLOT to ignore information about the spacing of the data. Plots produced with this option waste less space but can obscure the nature of the relationship between the variables.

HPOS=*axis-length*

specifies the number of print positions on the horizontal axis. The maximum value of *axis-length* that allows a plot to fit on one page is three positions less than the value of the LINESIZE= system option because there must be space for the procedure to print information next to the vertical axis. The exact maximum depends on the number of characters that are in the vertical variable's values. If *axis-length* is too large to fit on a line, then PROC PLOT ignores the option.

HREF=*value-specification*
draws lines on the plot perpendicular to the specified values on the horizontal axis. PROC PLOT includes the values 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 729.

Featured in: Example 8 on page 758

HREFCHAR='*character*'
specifies the character to use to draw the horizontal reference line.

Default: vertical bar (|)

See also: FORMCHAR= option on page 724 and HREF= on page 732

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 740

Featured in: Example 11 on page 766

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 748

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 740

Featured in: Example 13 on page 772

PLACEMENT=(*expression(s)*)
controls the placement of labels by specifying possible locations of the labels relative to their coordinates. Each *expression* consists of a list of one or more suboptions (H=, L=, S=, or V=) that are joined by an asterisk (*) or a colon (:). PROC PLOT uses the asterisk and colon to expand each expression into combinations of values for the four possible suboptions. The asterisk creates every possible combination of values in the expression list. A colon creates only pairwise combinations. The colon takes precedence over the asterisk. With the colon, if one list is shorter than the other, then the values in the shorter list are reused as necessary.

Use the following suboptions to control the placement:

H=*integer(s)*
specifies the number of horizontal spaces (columns) to shift the label relative to the starting position. Both positive and negative integers are valid. Positive integers shift the label to the right; negative integers shift it to the left. For example, you can use the H= suboption in the following way:

```
place=(h=0 1 -1 2 -2)
```

You can use the keywords BY ALT in this list. BY ALT produces a series of numbers whose signs alternate between positive and negative and whose absolute values change by one after each pair. For example, the following PLACE= specifications are equivalent:

```
place=(h=0 -1 to -3 by alt)

place=(h=0 -1 1 -2 2 -3 3)
```

If the series includes zero, then the zero appears twice. For example, the following PLACE= options are equivalent:

```
place=(h= 0 to 2 by alt)

place=(h=0 0 1 -1 2 -2)
```

Default: H=0

Range: –500 to 500

L=*integer(s)*
specifies the number of lines onto which the label can be split.

Default: L=1

Range: 1-200

S=*start-position(s)*
specifies where to start printing the label. The value for *start-position* can be one or more of the following:

CENTER
the procedure centers the label around the plotting symbol.

RIGHT
the label starts at the plotting symbol location and continues to the right.

LEFT
the label starts to the left of the plotting symbol and ends at the plotting symbol location.

Default: CENTER

V=*integer(s)*
specifies the number of vertical spaces (lines) to shift the label relative to the starting position. V= behaves the same as the H= suboption, described earlier.

A new expression begins when a suboption is not preceded by an operator. Parentheses around each expression are optional. They make it easier to recognize individual expressions in the list. However, the entire expression list must be in parentheses, as shown in the following example. The following table shows how this expression is expanded and describes each placement state.

```
place=((v=1)
       (s=right left : h=2 -2)
       (v=-1)
       (h=0 1 to 2 by alt * v=1 -1)
       (l=1 to 3 * v=1 to 2 by alt *
        h=0 1 to 2 by alt))
```

Each combination of values is a *placement state*. The procedure uses the placement states in the order in which they appear in the placement states list, so specify your most preferred placements first. For each label, the procedure tries all states, then it uses the first state that places the label with minimum penalty. When all labels are initially placed, the procedure cycles through the plot multiple times, systematically refining the placements. The refinement step tries to both minimize the penalties and to use placements nearer to the beginning of the states list. However, PROC PLOT uses a heuristic approach for placements, so the procedure does not always find the best set of placements.

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 739

Featured in: Example 11 on page 766 and Example 12 on page 770

Table 41.1 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.
	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.

Expression	Placement state	Meaning
	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 gray 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 739

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 729.

Featured in: Example 7 on page 755 and Example 12 on page 770

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 731

VPOS=*axis-length*
specifies the number of print positions on the vertical axis. The maximum value for *axis-length* that allows a plot to fit on one page is eight lines less than the value of the SAS system option PAGESIZE= because you must allow room for the procedure to print information under the horizontal axis. The exact maximum depends on the titles that are used, whether plots are overlaid, and whether CONTOUR is specified. If the value of *axis-length* specifies a plot that cannot fit on one page, then the plot spans multiple pages.

See also: HPOS= on page 732

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 729.

Featured in: Example 2 on page 746

VREFCHAR='*character*'

specifies the character to use to draw the vertical reference lines.

Default: horizontal bar (-)

See also: FORMCHAR= option on page 724, HREFCHAR= on page 732, and VREF= on page 737

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 on. Any options submitted in the PROC PLOT statement remain in effect until you submit another PROC PLOT statement.

When you submit a RUN statement, PROC PLOT executes all the statements submitted since the last PROC PLOT or RUN statement. Each group of statements is called a *RUN group*. With each RUN group, PROC PLOT begins a new page and begins with the first item in the VPERCENT= and HPERCENT= lists, if any.

To terminate the procedure, submit a QUIT statement, a DATA statement, or a PROC statement. Like the RUN statement, each of these statements completes a RUN group. If you do not want to execute the statements in the RUN group, then use the RUN CANCEL statement, which terminates the procedure immediately.

You can use the BY statement interactively. The BY statement remains in effect until you submit another BY statement or terminate the procedure.

See Example 11 on page 766 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	^
CENTER	<=0	v

If you are using pointer symbols and multiple points coincide, then PROC PLOT uses the number of points as the plotting symbol if the number of points is between 2 and 9. If the number of points is more than 9, then the procedure uses an asterisk (*).

Note: Because of character set differences among operating environments, the pointer symbol for S=CENTER and V>0 might differ from the one shown here. △

Understanding Penalties

PROC PLOT assesses the quality of placements with penalties. If all labels are plotted with zero penalty, then no labels collide and all labels are near their symbols. When it is not possible to place all labels with zero penalty, PROC PLOT tries to minimize the total penalty. The following table gives a description of the penalty, the default value of the penalty, the index that you use to reference the penalty, and the range of values that you can specify if you change the penalties. Each penalty is described in more detail in Table 41.3 on page 740.

Table 41.2 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

The following table contains the index values from the previous table with a description of the corresponding penalty.

Table 41.3 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: $$[\mathrm{MAX}(|H| - fhs, 0) + vsw \times \mathrm{MAX}(|V| - (L{+}fvs{+}(V > 0))/2, 0)]/vhsd$$ 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 753 illustrates a case in which removing the horizontal shift penalty is useful.
8	a label might collide with its own plotting symbol. If the plotting symbol is blank, then a collision state cannot occur. See "Collision States" on page 741 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 20th penalty of 2 is the penalty for not printing the sixth through 200th character. When the last index i is greater than 18, the last penalty is used for the $(i - 14)$th character and beyond.

You can also extend the penalty list by just specifying the starting index. For example, the following PENALTIES= option is equivalent to the one above:

```
penalties(15)=2 2 11 10 8 2
```

Collision States

Collision states are placement states that can cause a label to collide with its own plotting symbol. PROC PLOT usually avoids using collision states because of the large default penalty of 500 that is associated with them. PROC PLOT does not consider the actual length or splitting of any particular label when determining if a placement state is a collision state. The following are the rules that PROC PLOT uses to determine collision states:

- When S=CENTER, placement states that do not shift the label up or down sufficiently so that all of the label is shifted onto completely different lines from the symbol are collision states.
- When S=RIGHT, placement states that shift the label zero or more positions to the left without first shifting the label up or down onto completely different lines from the symbol are collision states.
- When S=LEFT, placement states that shift the label zero or more positions to the right without first shifting the label up or down onto completely different lines from the symbol are collision states.

Note: A collision state cannot occur if you do not use a plotting symbol. △

Reference Lines

PROC PLOT places labels and computes penalties before placing reference lines on a plot. The procedure does not attempt to avoid rows and columns that contain reference lines.

Hidden Label Characters

In addition to the number of hidden observations and hidden plotting symbols, PROC PLOT prints the number of hidden label characters. Label characters can be hidden by plotting symbols or other label characters.

Overlaying Label Plots

When you overlay a label plot and a nonlabel plot, PROC PLOT tries to avoid collisions between the labels and the characters of the nonlabel plot. When a label character collides with a character in a nonlabel plot, PROC PLOT adds the usual penalty to the penalty sum.

When you overlay two or more label plots, all label plots are treated as a single plot in avoiding collisions and computing hidden character counts. Labels of different plots never overprint, even with the OVP system option in effect.

Computational Resources Used for Label Plots

This section uses the following variables to discuss how much time and memory PROC PLOT uses to construct label plots:

n	number of points with labels
len	constant length of labels
s	number of label pieces, or fragments
p	number of placement states specified in the PLACE= option.

Time

For a given plot size, the time that is required to construct the plot is approximately proportional to $n \times len$. The amount of time required to split the labels is approximately proportional to ns^2. Generally, the more placement states that you specify, the more time that PROC PLOT needs to place the labels. However, increasing the number of horizontal and vertical shifts gives PROC PLOT more flexibility to avoid collisions, often resulting in less time used to place labels.

Memory

PROC PLOT uses 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 723.

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 "ODS Output Object Table Names" in *SAS Output Delivery System: User's Guide*.

Table 41.4 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 might not be able to recognize coinciding values.

If you specify a plotting symbol, then PROC PLOT uses the same symbol regardless of the number of observations whose values coincide. If you use the OVERLAY option and overprinting is not in effect, then PROC PLOT uses the symbol from the first plot request. In both cases, the output includes a message telling you how many observations are hidden.

Examples: PLOT Procedure

Example 1: Specifying a Plotting Symbol

Procedure features:

PLOT statement

plotting symbol in plot request

This example expands on Output 41.1 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 1604 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.

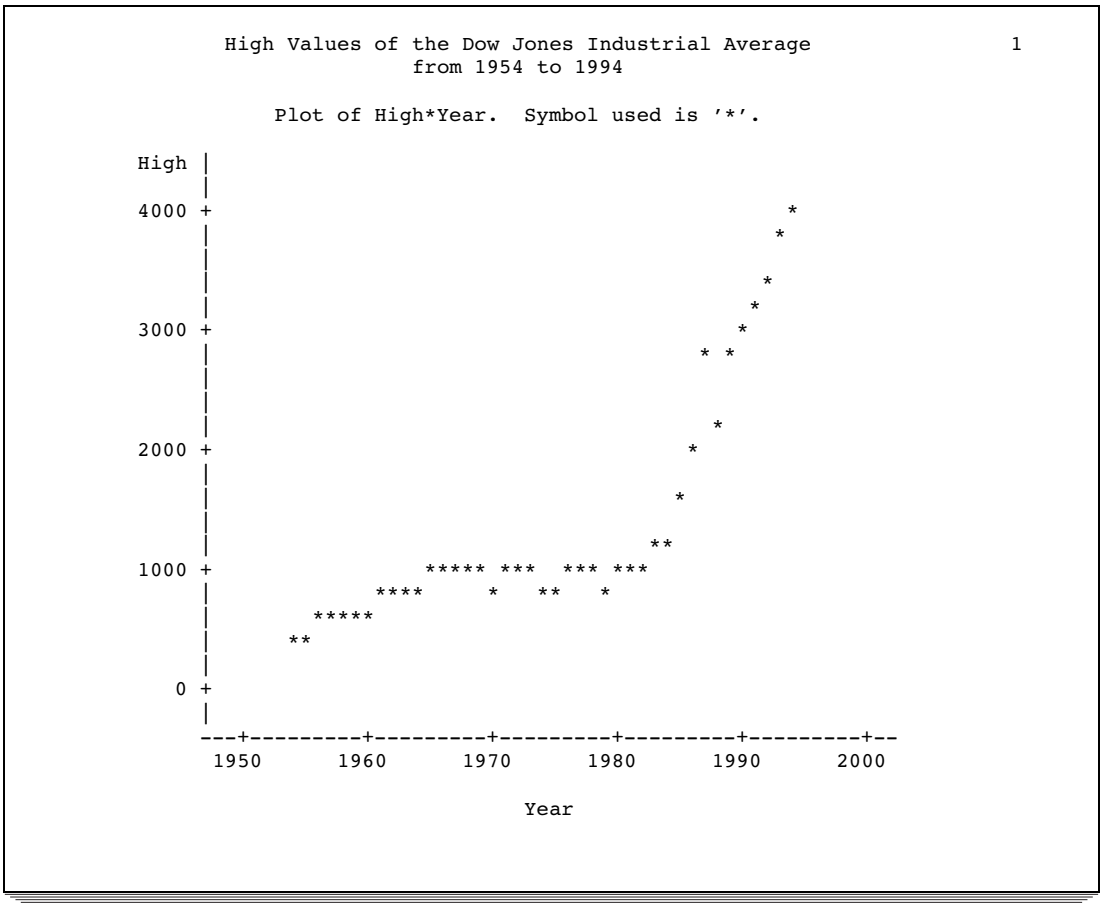
```
proc plot data=djia;
   plot high*year='*';
```

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.



Example 2: Controlling the Horizontal Axis and Adding a Reference Line

Procedure features:

PLOT statement options:

HAXIS=

VREF=

Data set: DJIA on page 745

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='*'
```

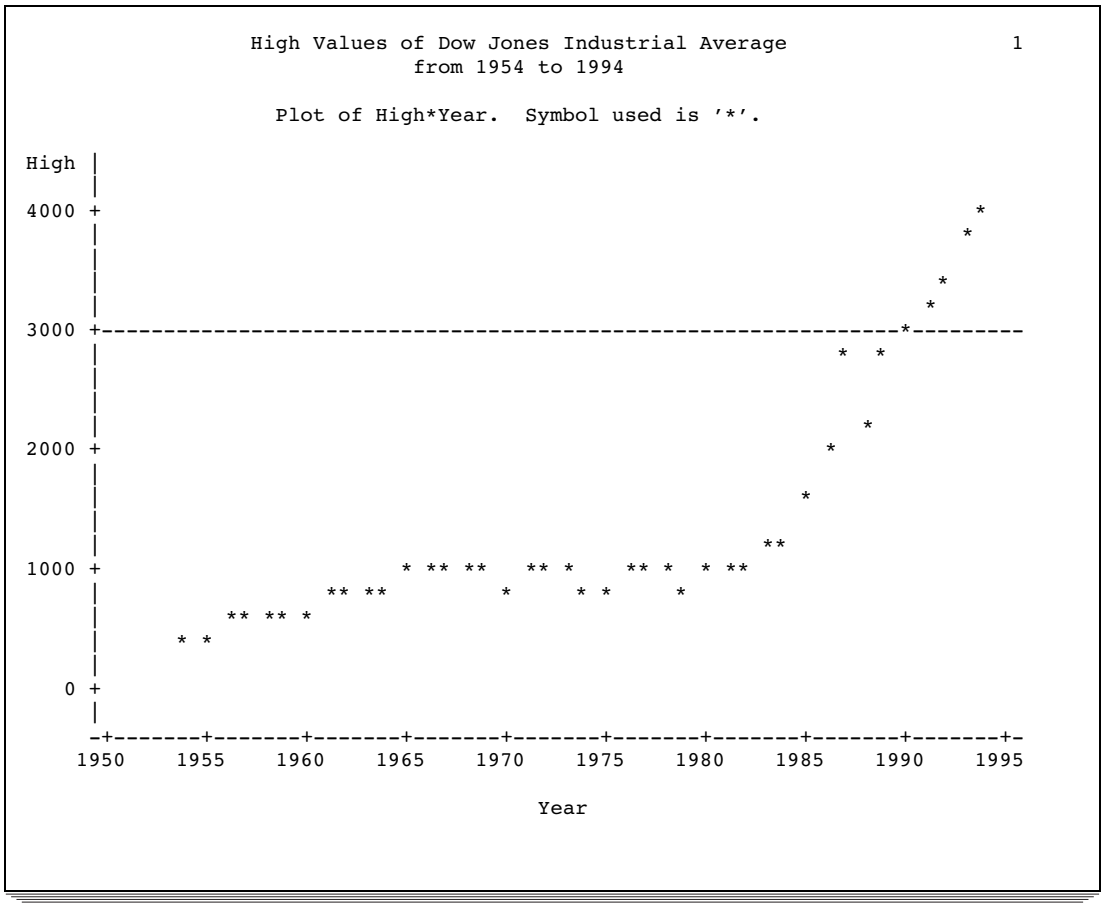
Customize the horizontal axis and draw a reference line. HAXIS= specifies that the horizontal axis will show the values 1950 to 1995 in five-year increments. VREF= draws a reference line that extends from the value 3000 on the vertical axis.

```
                    / 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



Example 3: Overlaying Two Plots

Procedure features:
PLOT statement options
BOX
OVERLAY
Data set: DJIA on page 745

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;
```

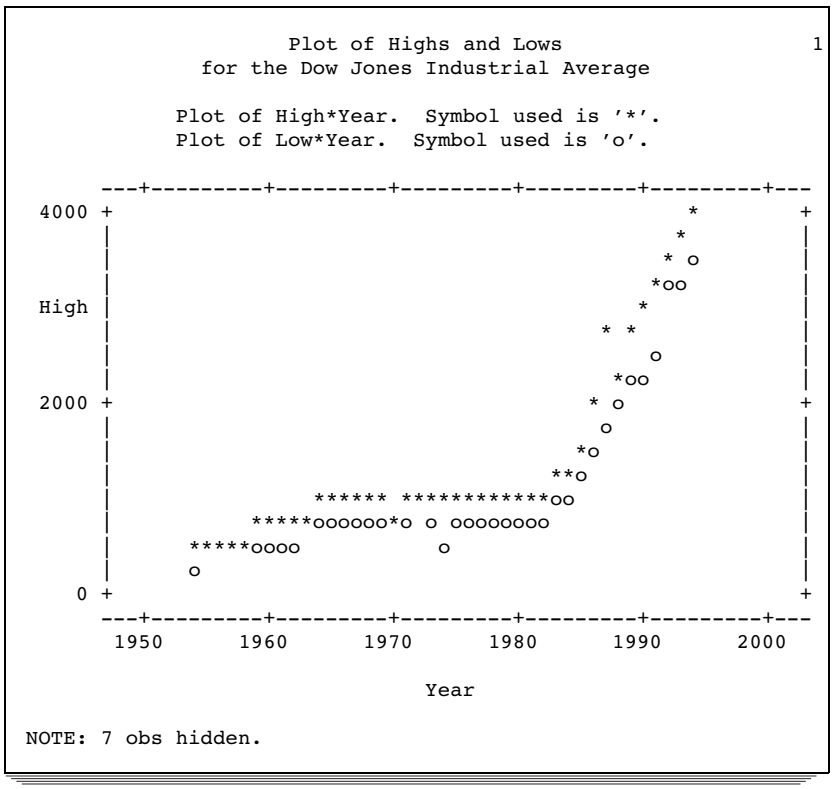
Create the plot. The first plot request plots High on the vertical axis, plots Year on the horizontal axis, and specifies an asterisk as a plotting symbol. The second plot request plots Low on the vertical axis, plots Year on the horizontal axis, and specifies an '**o** ' as a plotting symbol. OVERLAY 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



Example 4: Producing Multiple Plots per Page

Procedure features:

PROC PLOT statement options

HPERCENT=
VPERCENT=

Data set: DJIA on page 745

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';
```

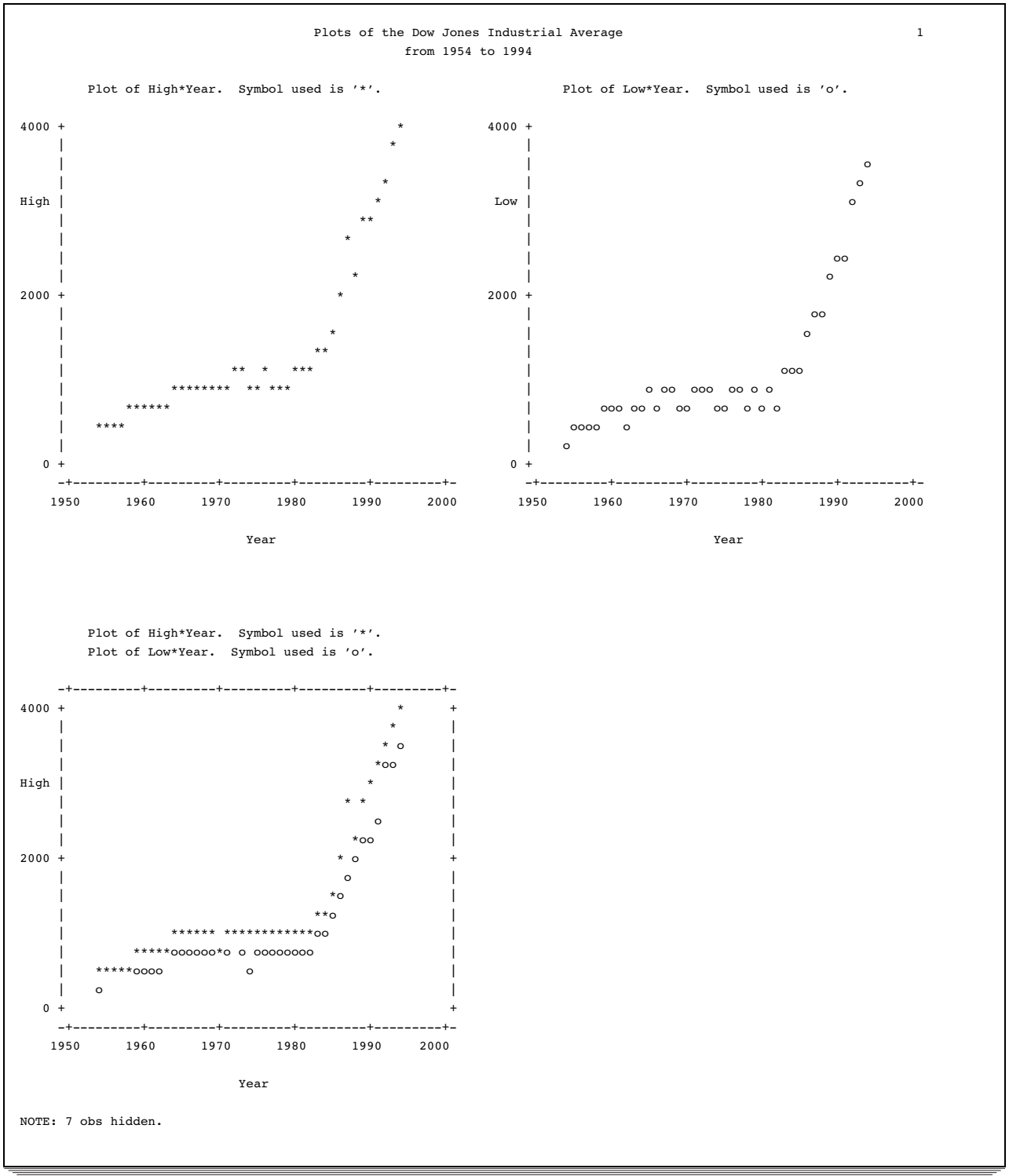
Create the third plot. The first plot request plots High on the vertical axis, plots Year on the horizontal axis, and specifies an asterisk as a plotting symbol. The second plot request plots Low on the vertical axis, plots Year on the horizontal axis, and specifies an '**o** ' as a plotting symbol. 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
2000
0
High
Low
1950
1960
1970
1980
1990
2000
Year
Plot of High*Year. Symbol used is '*'.
Plot of Low*Year. Symbol used is 'o'.
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;
```

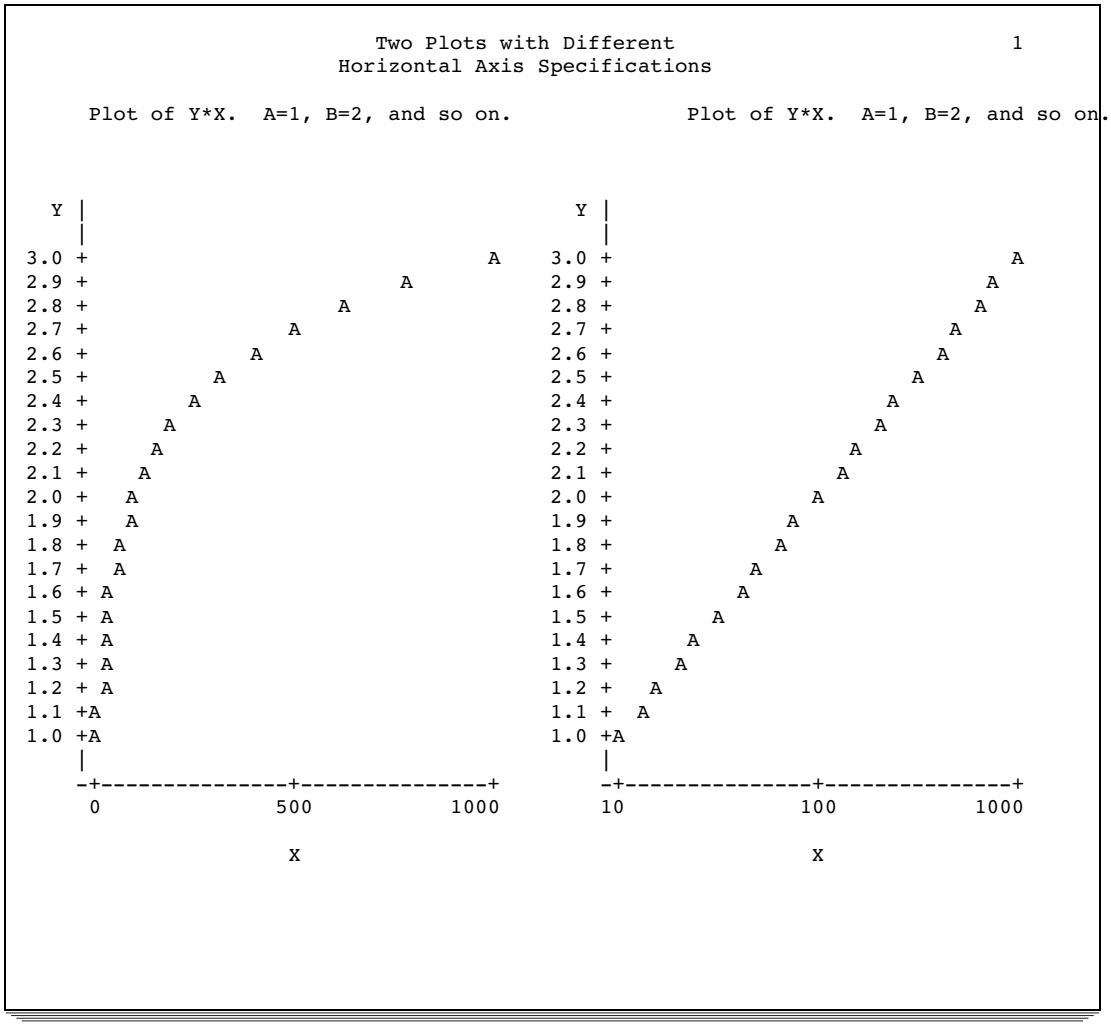
Create the plots. The plot requests plot Y on the vertical axis and X on the horizontal axis. HAXIS= specifies a logarithmic scale for the horizontal axis for the second plot.

```
    plot y*x;
    plot y*x / haxis=10 100 1000;
```

Specify the titles.

```
    title 'Two Plots with Different';
    title2 'Horizontal Axis Specifications';
run;
```

Output



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.

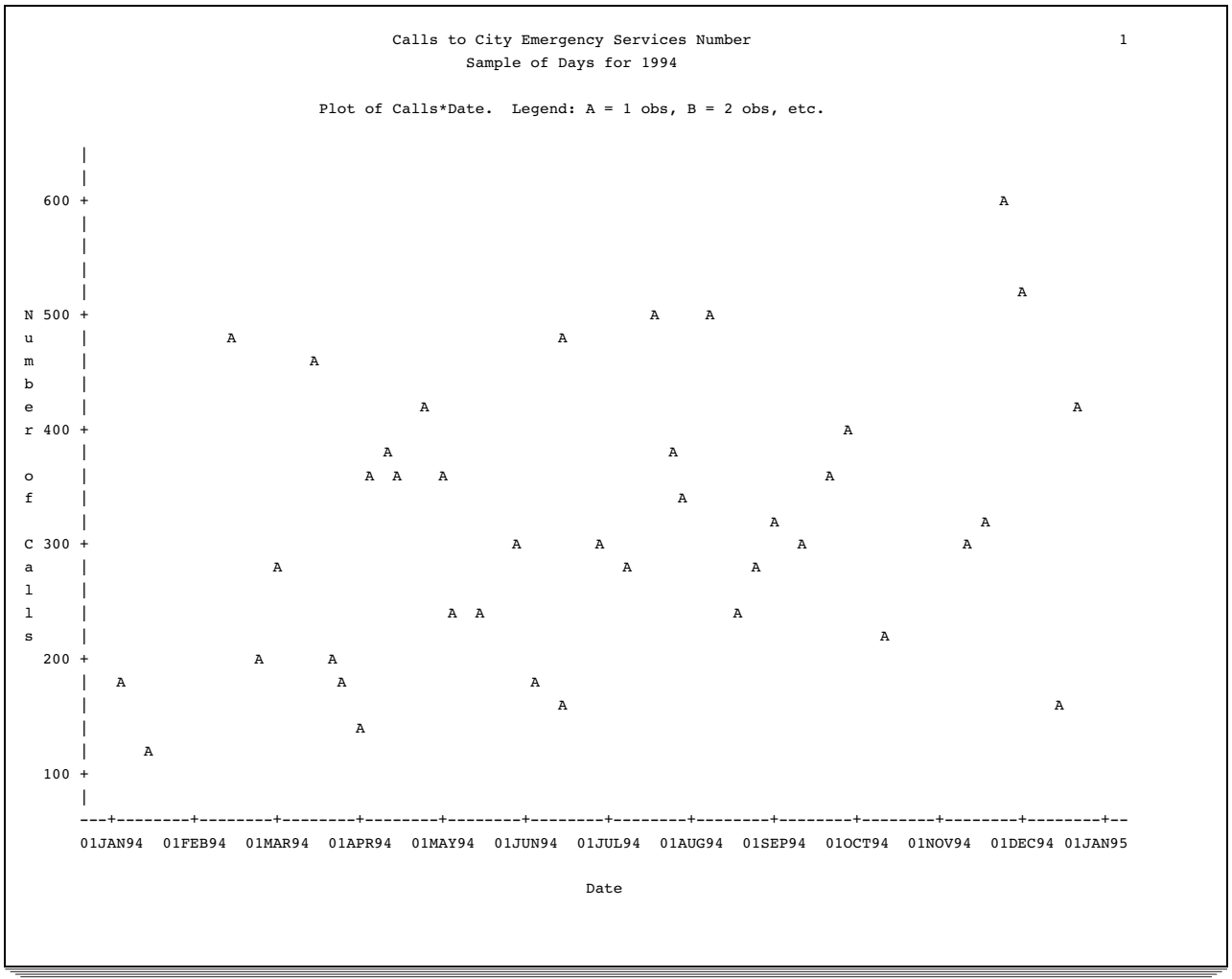
```
   format date date7.;
```

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.



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.

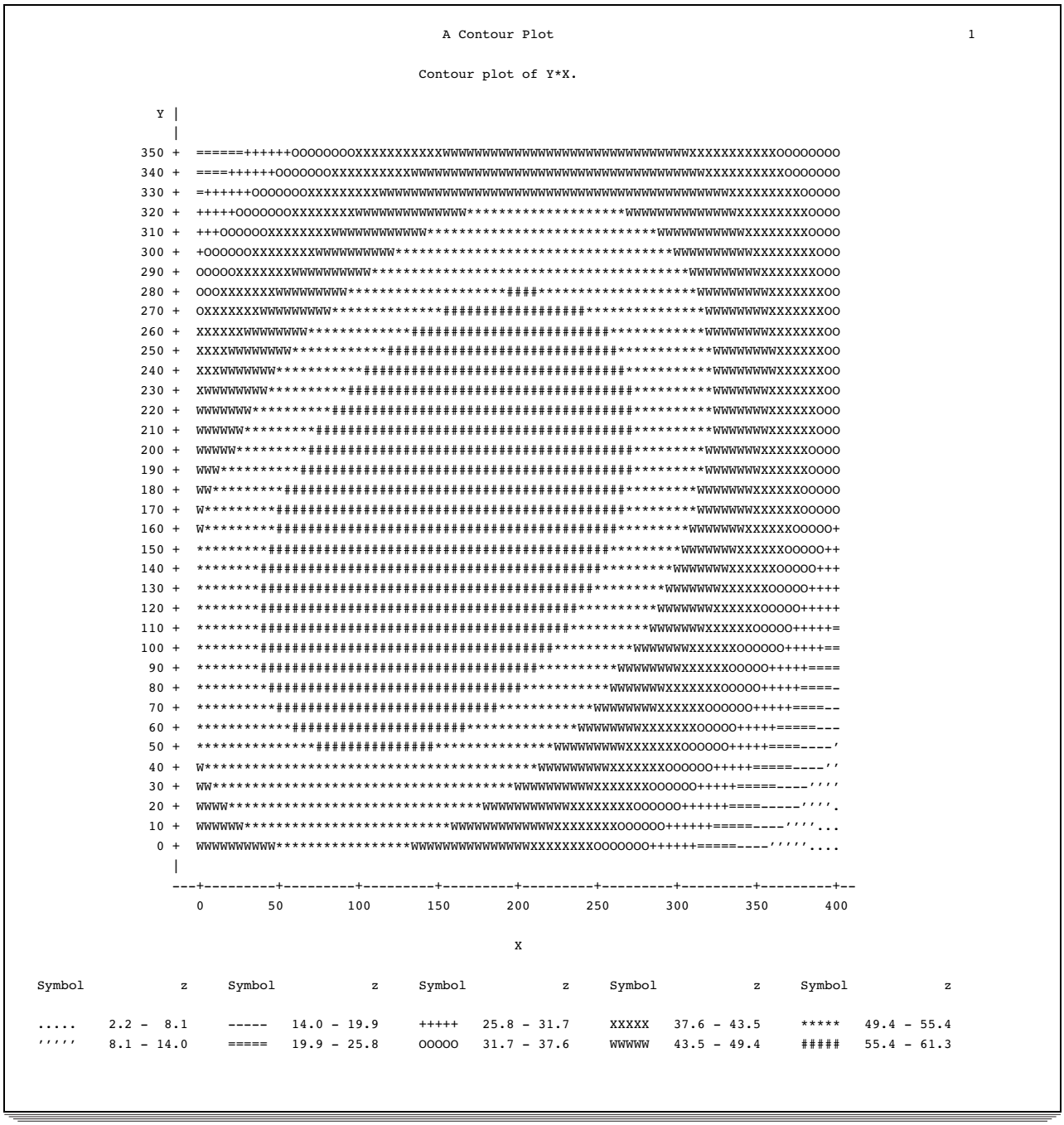
```
proc plot data=contours;
plot y*x=z / contour=10;
```

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.



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 1605 creates this data set.

```
data education;
   input State $14. +1 Code $ DropoutRate Expenditures MathScore
         Region $;
   label dropout='Dropout Percentage - 1989'
          expend='Expenditure Per Pupil - 1989'
            math='8th Grade Math Exam - 1990';
   datalines;
Alabama        AL 22.3 3197 252 SE
Alaska         AK 35.8 7716 .   W
...more data lines...
New York       NY 35.0 .    261 NE
North Carolina NC 31.2 3874 250 SE
North Dakota   ND 12.1 3952 281 MW
Ohio           OH 24.4 4649 264 MW
;
```

Sort the EDUCATION data set. PROC SORT sorts EDUCATION by Region so that Region can be used as the BY variable in PROC PLOT.

```
proc sort data=education;
   by region;
run;
```

* Source: U.S. Department of Education.

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;
```

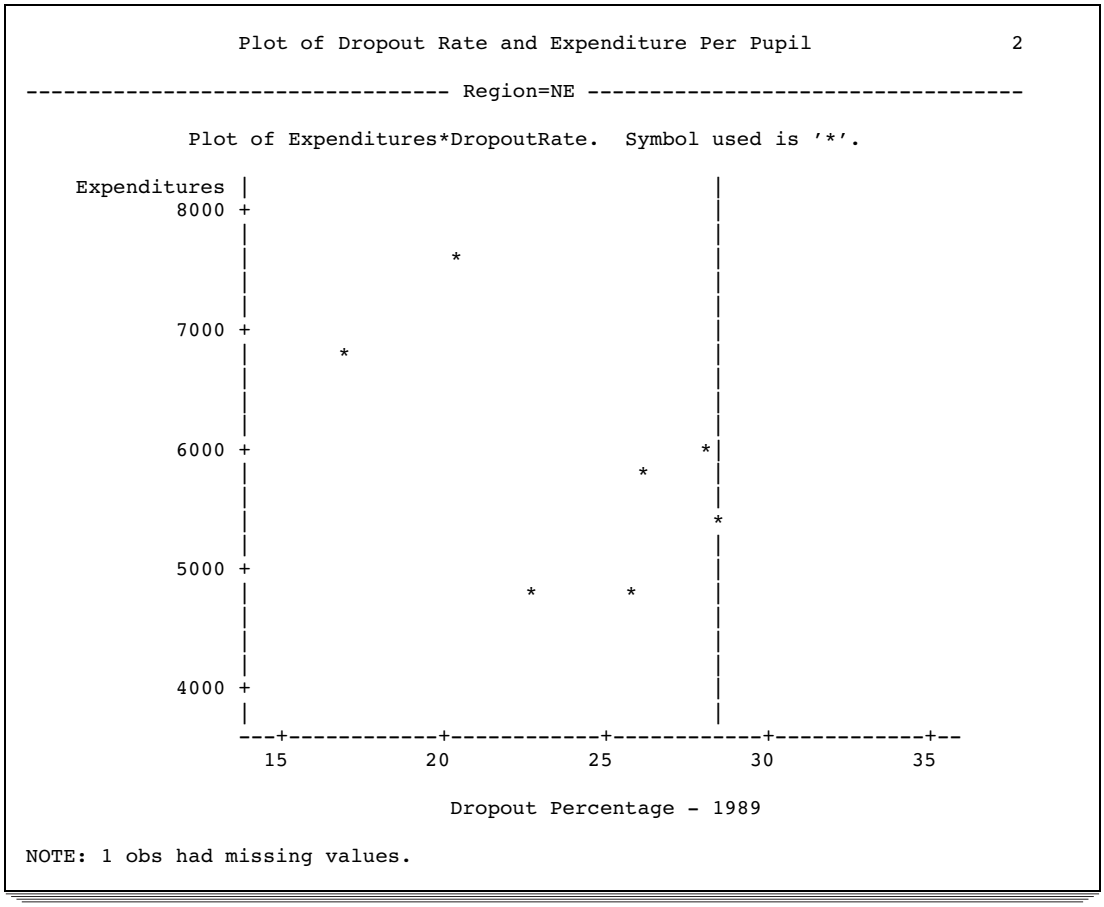
Specify the title.

```
    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
```



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 759

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 758.

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.

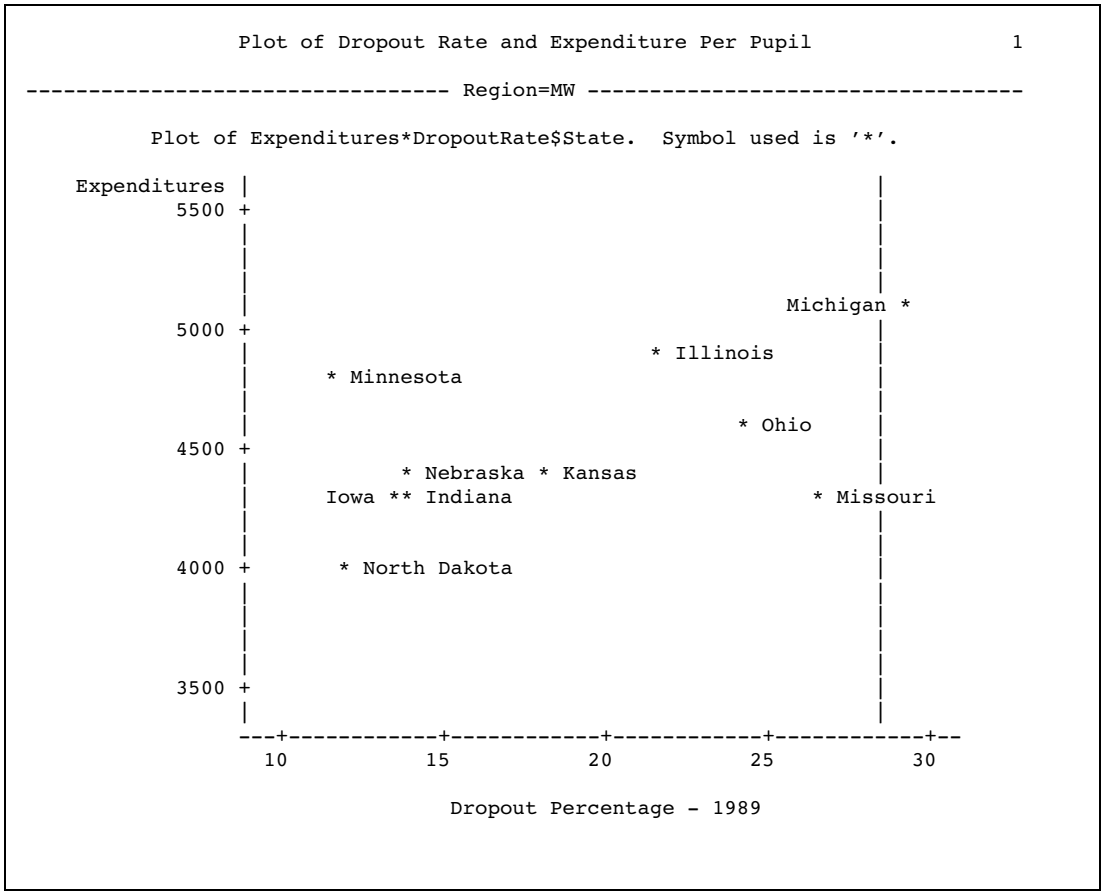
```
   plot expenditures*dropoutrate='*' $ state / href=28.6;
```

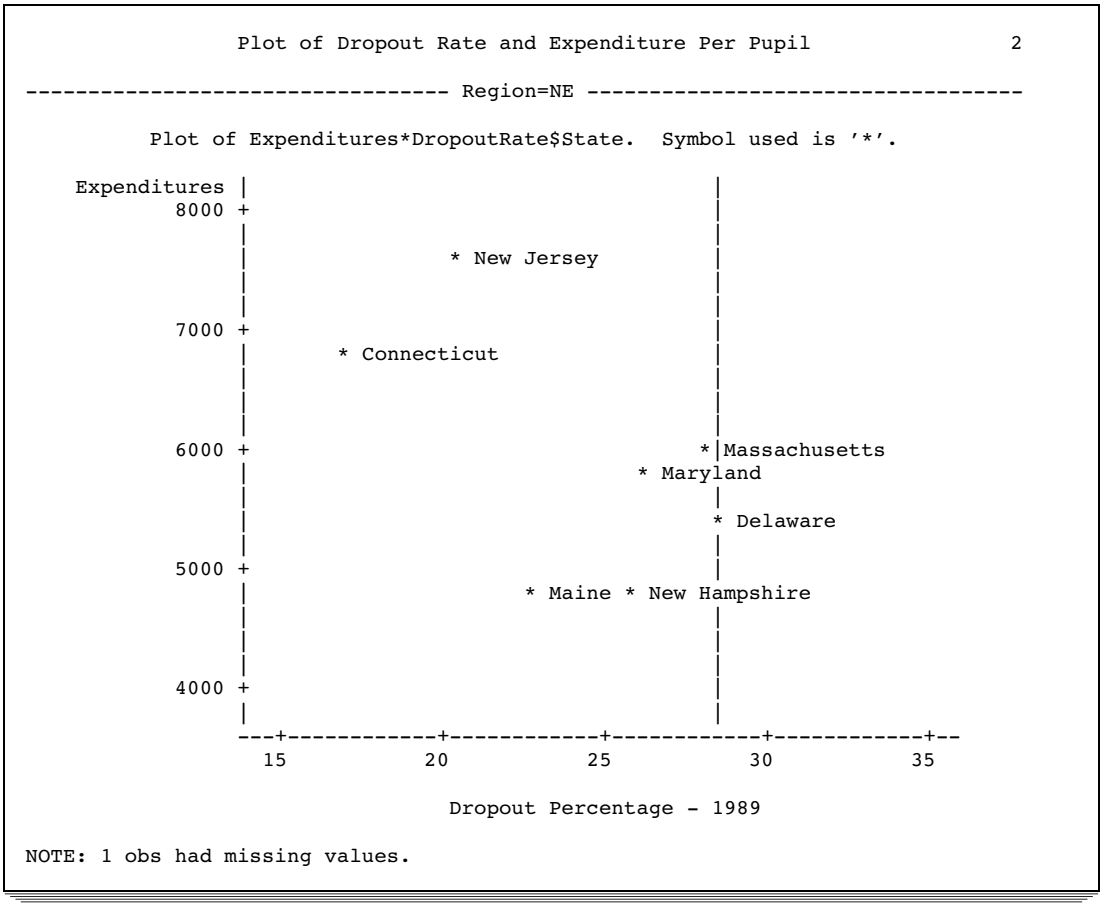
Specify the title.

```
   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.





Example 10: Excluding Observations That Have Missing Values

Procedure features:

PROC PLOT statement option

NOMISS

Data set: EDUCATION on page 759

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 761.

```
                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 1574 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

                        Vertical   Horizontal             Starting           Vertical   Horizontal
      Label                Axis         Axis   Penalty   Position   Lines      Shift        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.

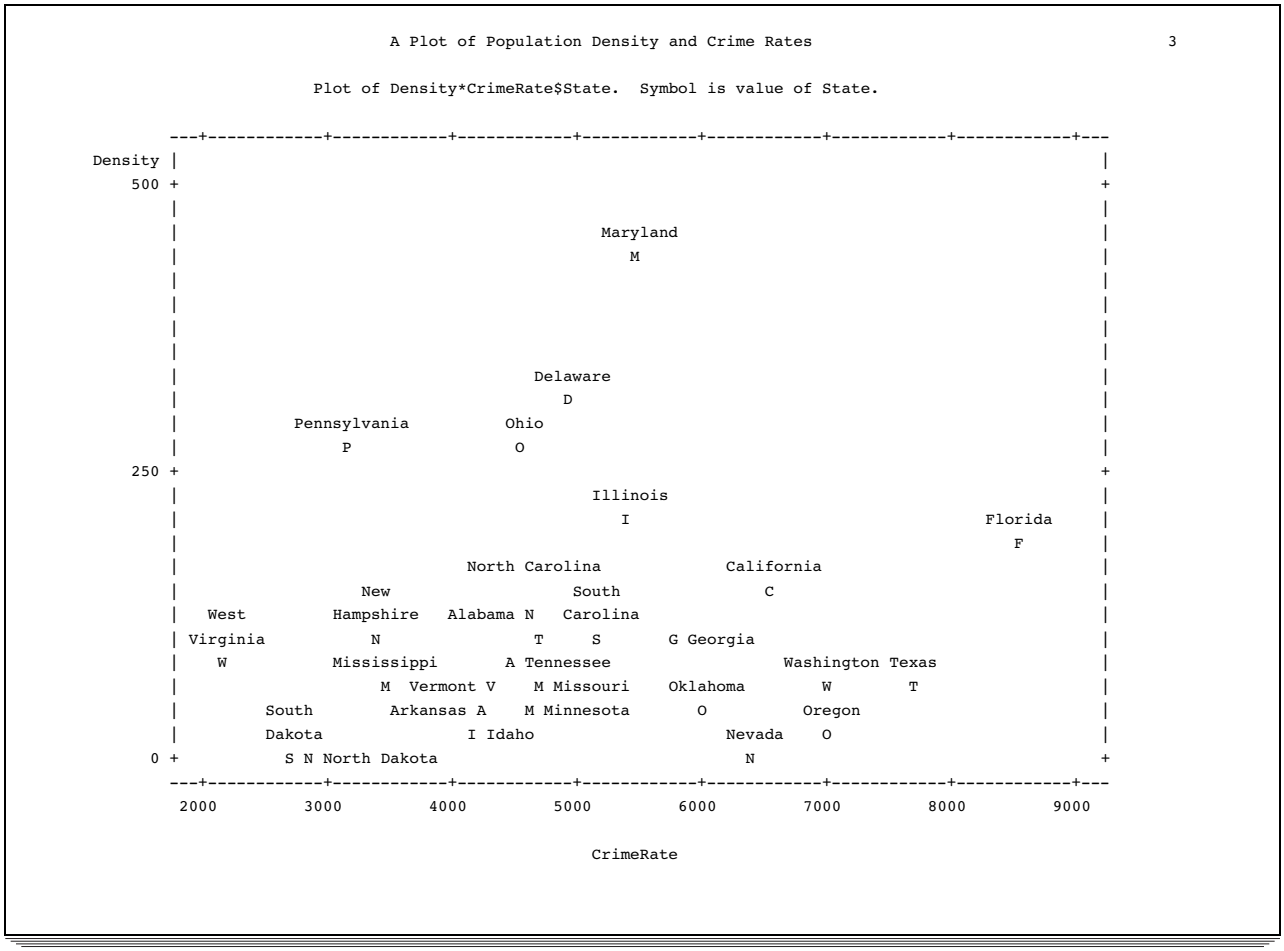
```
     placement=((v=2 1 : l=2 1)
           ((l=2 2 1 : v=0 1 0) * (s=right left : h=2 -2))
           (s=center right left * l=2 1 * v=0 1 -1 2 *
            h=0 1 to 5 by alt));
```

Specify the title.

```
   title 'A Plot of Population Density and Crime Rates';
run;
```

Output

No collisions occur in the plot.



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 767

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.

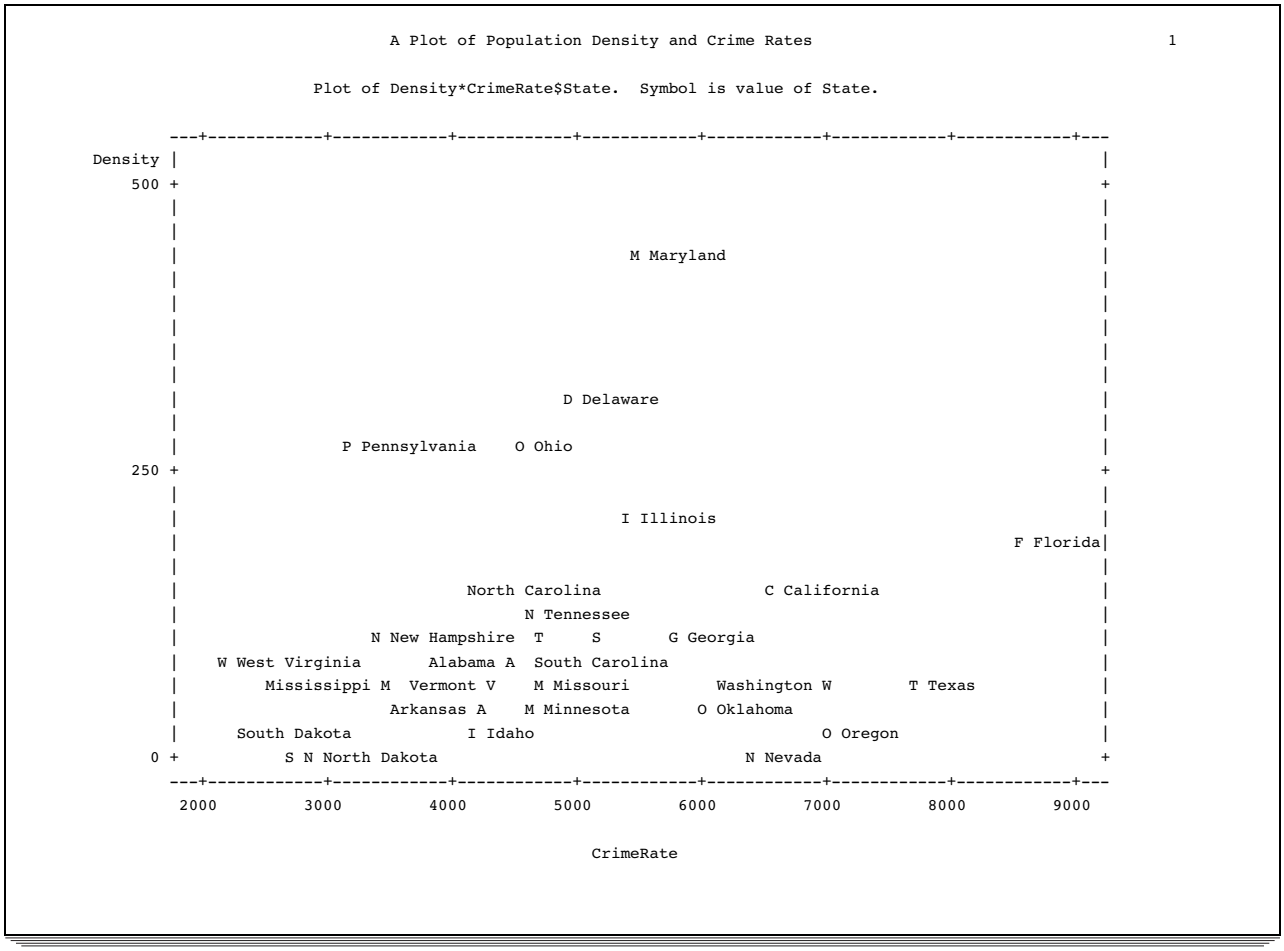
```
          box
          list=1
          haxis=by 1000
          vaxis=by 250
          %place(4);
```

Specify the title.

```
   title 'A Plot of Population Density and Crime Rates';
run;
```

Output

No collisions occur in the plot.



Example 13: Changing a Default Penalty

Procedure features:

PLOT statement option

PENALTIES=

Data set: CENSUS on page 767

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

42

The PMENU Procedure

Overview: PMENU Procedure **777**
Syntax: PMENU Procedure **778**
PROC PMENU Statement **779**
CHECKBOX Statement **780**
DIALOG Statement **781**
ITEM Statement **783**
MENU Statement **786**
RADIOBOX Statement **787**
RBUTTON Statement **788**
SELECTION Statement **789**
SEPARATOR Statement **790**
SUBMENU Statement **790**
TEXT Statement **790**
Concepts: PMENU Procedure **792**
Procedure Execution **792**
Initiating the Procedure **792**
Ending the Procedure **792**
Steps for Building and Using PMENU Catalog Entries **792**
Templates for Coding PROC PMENU Steps **793**
Examples: PMENU Procedure **794**
Example 1: Building a Menu Bar for an FSEDIT Application **794**
Example 2: Collecting User Input in a Dialog Box **797**
Example 3: Creating a Dialog Box to Search Multiple Variables **800**
Example 4: Creating Menus for a DATA Step Window Application **806**
Example 5: Associating Menus with a FRAME Application **812**

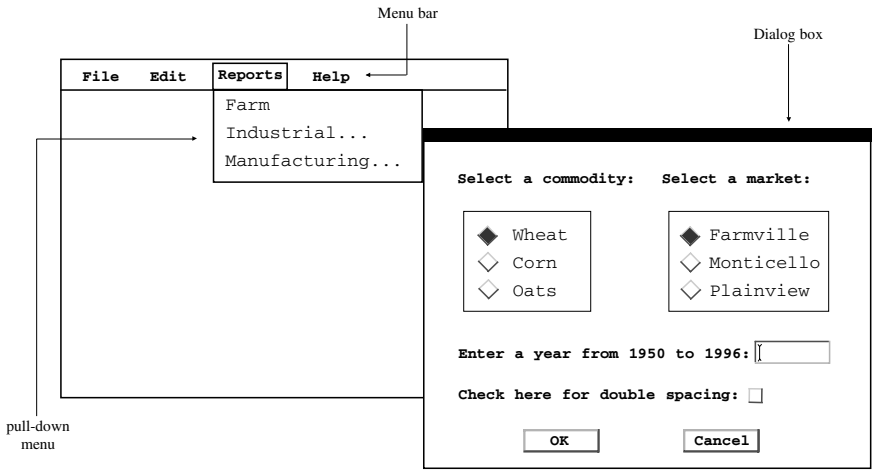
Overview: PMENU Procedure

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 42.1 Menu Bar, Menu, and Dialog Box



Note: A menu bar in some operating environments might appear as a pop-up menu or might appear at the bottom of the window. △

The PMENU procedure produces no immediately visible output. It simply builds a catalog entry of type PMENU that can be used later in an application.

Syntax: PMENU Procedure

Restriction: You must use at least one MENU statement followed by at least one ITEM statement.

Tip: Supports RUN group processing

Tip: You can also use appropriate global statements with this procedure. See Chapter 2, "Fundamental Concepts for Using Base SAS Procedures," on page 17 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*;

Task	Statement
Define customized menus	"PROC PMENU Statement" on page 779
Define choices a user can make in a dialog box	"CHECKBOX Statement" on page 780
Describe a dialog box that is associated with an item in a menu	"DIALOG Statement" on page 781
Identify an item to be listed in a menu bar or in a menu	"ITEM Statement" on page 783
Name the catalog entry or define a menu	"MENU Statement" on page 786
List and define mutually exclusive choices within a dialog box	"RADIOBOX Statement" on page 787 and "RBUTTON Statement" on page 788
Define a command that is submitted when an item is selected	"SELECTION Statement" on page 789
Draw a line between items in a menu	"SEPARATOR Statement" on page 790
Define a common submenu associated with an item	"SUBMENU Statement" on page 790
Specify text and the input fields for a dialog box	"TEXT Statement" on page 790

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 library. If you omit CATALOG=, then the entries are stored in the SASUSER.PROFILE catalog.

Featured in: Example 1 on page 794

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 menu.

Restriction: Must be followed by at least one TEXT statement.

Featured in:

Example 2 on page 797
Example 3 on page 800
Example 4 on page 806

DIALOG *dialog-box 'command-string*
field-number-specification';

Required Arguments

command-string
is the command or partial command that is executed when the item is selected. The limit of the *command-string* that results after the substitutions are made is the command-line limit for your operating environment. Typically, the command-line limit is approximately 80 characters.

The limit for '*command-string field-number-specification*' is 200 characters.

Note: If you are using PROC PMENU to submit any command that is valid only in the PROGRAM EDITOR window (such as the INCLUDE command), then you must have the windowing environment running, and you must return control to the PROGRAM EDITOR window. △

dialog-box
is the same name specified for the DIALOG= option in a previous ITEM statement.

field-number-specification
can be one or more of the following:

@1…@*n*

%1…%*n*

&1…&*n*

You can embed the field numbers, for example @1, %1, or &1, in the command string and mix different types of field numbers within a command string. The numeric portion of the field number corresponds to the relative position of TEXT, RADIOBOX, and CHECKBOX statements, not to any actual number in these statements.

@1…@*n*
are optional TEXT statement numbers that can add information to the command before it is submitted. Numbers preceded by an at sign (@) correspond to TEXT statements that use the LEN= option to define input fields.

%1…%*n*
are optional RADIOBOX statement numbers that can add information to the command before it is submitted. Numbers preceded by a percent sign (%) correspond to RADIOBOX statements following the DIALOG statement.

Note: Keep in mind that the numbers correspond to RADIOBOX statements, not to RBUTTON statements. △

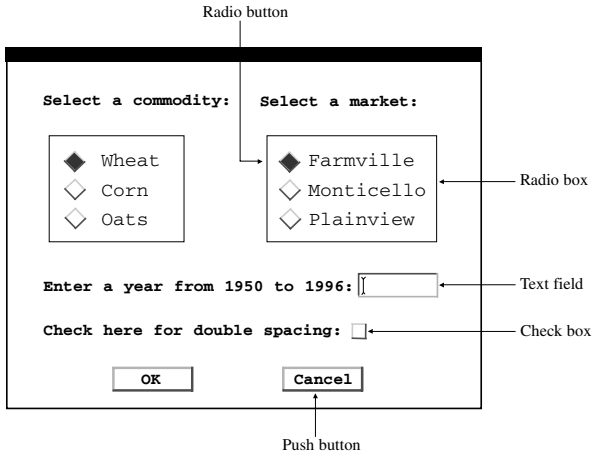
&1…&*n*
are optional CHECKBOX statement numbers that can add information to the command before it is submitted. Numbers preceded by an ampersand (&) correspond to CHECKBOX statements following the DIALOG statement.

Note: To specify a literal @ (at sign), % (percent sign), or & (ampersand) in the *command-string*, use a double character: @@ (at signs), %% (percent signs), or && (ampersands). △

Details

- You cannot control the placement of the dialog box. The dialog box is not scrollable. The size and placement of the dialog box are determined by your windowing environment.
- To use the DIALOG statement, specify an ITEM statement with the DIALOG= option in the ITEM statement.
- The ITEM statement creates an entry in a menu bar or in a menu, and the DIALOG= option specifies which DIALOG statement describes the dialog box.
- You can use CHECKBOX, RADIOBOX, and RBUTTON statements to define the contents of the dialog box.
- The following figure shows a typical dialog box. A dialog box can request information in three ways:
  - Fill in a field. Fields that accept text from a user are called *text fields*.
  - Choose from a list of mutually exclusive choices. A group of selections of this type is called a *radio 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 42.2 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 menu.

Featured in: Example 1 on page 794

ITEM *command* <*option(s)*><*action-options*>;

ITEM *'menu-item'* <*option(s)*><*action-options*>;

Task	Option
Specify the action for the item	
Associate the item with a dialog box	DIALOG=
Associate the item with a 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 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

* The actual names of the buttons vary in different windowing environments.

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 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 800

MENU=*pull-down-menu*
the name of an associated MENU statement, which displays a menu when the user selects this item.

Featured in: Example 1 on page 794

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 794

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 794

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 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 785

MNEMONIC=*character*
underlines the first occurrence of *character* in the text string that appears on the menu. The *character* must be in the text string.

The *character* is typically used in combination with another key, such as ALT. When you use the key sequence, it has the same effect as putting your cursor on the item. But it *does not* invoke the action that the item controls.

Restriction: This option is not available in all operating environments. If you include this option and it is not available in your operating environment, then the option is ignored.

STATE=CHECK|RADIO
provides the ability to place a check box or a radio button next to an item that has been selected.

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 menus. The items that you specify in the ITEM statement can be commands that are issued when the user selects the item, or they can be descriptions of other actions that are performed by associated DIALOG, MENU, SELECTION, or SUBMENU statements.

All ITEM statements for a menu must be placed immediately after the MENU statement and before any DIALOG, SELECTION, SUBMENU, or other MENU statements. In some operating environments, you can insert SEPARATOR statements between ITEM statements to produce lines separating groups of items in a menu. See “SEPARATOR Statement” on page 790 for more information.

Note: If you specify a menu bar that is too long for the window, then it might be truncated or wrapped to multiple lines. △

MENU Statement

Names the catalog entry that stores the menus or defines a menu.

Featured in: Example 1 on page 794

MENU *menu-bar*;

MENU *pull-down-menu*;

Required Arguments

One of the following arguments is required:

menu-bar
names the catalog entry that stores the menus.

pull-down-menu
names the menu that appears when the user selects an item in the menu bar. The value of *pull-down-menu* must match the *pull-down-menu* name that is specified in the MENU= option in a previous ITEM statement.

Defining Menus

When used to define a menu, the MENU statement must follow an ITEM statement that specifies the MENU= option. Both the ITEM statement and the MENU statement for the menu must be in the same RUN group as the MENU statement that defines the menu bar for the PMENU catalog entry.

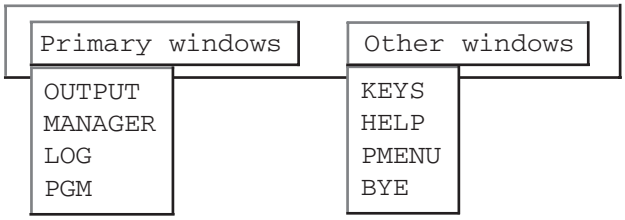
For both menu bars and menus, follow the MENU statement with ITEM statements that define each of the items that appear on the menu. Group all ITEM statements for a menu together. For example, the following PROC PMENU step creates one catalog entry, WINDOWS, which produces a menu bar with two items, **`Primary windows`** and **`Other windows`**. When you select one of these items, a menu is displayed.

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

proc pmenu cat=proclib.mycat;

      /* create catalog entry */
   menu windows;
   item 'Primary windows' menu=prime;
   item 'Other windows' menu=other;

      /* create first menu */
   menu prime;
   item output;
   item manager;
   item log;
   item pgm;

      /* create second menu */
   menu other;
   item keys;
   item help;
```

```
    item pmenu;
    item bye;

    /* end of run group */
run;
```

The following figure shows the resulting menu selections.

Figure 42.3 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 800

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 800

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 800

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 794
Example 4 on page 806

SELECTION *selection 'command-string'*;

Required Arguments

selection
is the same name specified for the SELECTION= option in a previous ITEM statement.

command-string
is a text string, enclosed in quotation marks, that is submitted as a command-line command when the user selects this item. There is a limit of 200 characters for *command-string*. However, the command-line limit of approximately 80 characters cannot be exceeded. The command-line limit differs slightly for various operating environments.

Note: SAS uses only the first eight characters of an item that is specified with a SELECTION statement. When a user selects an item from a menu list, the first eight characters of each item name in the list must be unique so that SAS can select the correct item in the list. If the first eight characters are not unique, SAS selects the last item in the list. △

Details

You define the name of the item in the ITEM statement and specify the SELECTION= option to associate the item with a subsequent SELECTION statement. The SELECTION statement then defines the actual command that is submitted when the user chooses the item in the menu bar or menu.

You are likely to use the SELECTION statement to define a command string. You create a simple alias by using the ITEM statement, which invokes a longer command string that is defined in the SELECTION statement. For example, you could include an item in the menu bar that invokes a WINDOW statement to enable data entry. The actual commands that are processed when the user selects this item are the commands to include and submit the application.

Note: If you are using PROC PMENU to issue any command that is valid only in the PROGRAM EDITOR window (such as the INCLUDE command), then you must have the windowing environment running, and you must return control to the PROGRAM EDITOR window. △

SEPARATOR Statement

Draws a line between items on a 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 794

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 797

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 797

'*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 might not support all of these attributes.

COLOR=*color*
defines the color for the text or input field characters. Here are the color values that you can use:

BLACK	BROWN
GRAY	MAGENTA
PINK	WHITE
BLUE	CYAN
GREEN	ORANGE
RED	YELLOW

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 might 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 792.

Ending the Procedure

Submit a QUIT, DATA, or new PROC statement to execute any statements that have not executed and end the PMENU procedure. Submit a RUN CANCEL statement to cancel any statements that have not executed and end the PMENU procedure.

Steps for Building and Using PMENU Catalog Entries

In most cases, building and using PMENU entries requires the following steps:

1 Use PROC PMENU to define the menu bars, menus and other features that you want. Store the output of PROC PMENU in a SAS catalog.
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 809.
   - 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 812.
   - the PMENU function in SAS/AF and SAS/FSP software.
   - the SETPMENU command in SAS/FSP software. See Example 1 on page 794.
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 menu:

```
proc pmenu;
    menu menu-bar;
    item 'menu-item' menu=pull-down-menu;
    ...more-ITEM-statements...
    menu pull-down-menu;
    ...ITEM-statements-for-pull-down-menu...
run;
```

- Create a menu bar with an item that submits a command other than the one that appears on the menu bar:

```
proc pmenu;
   menu menu-bar;
   item 'menu-item' selection=selection;
   ...more-ITEM-statements...
   selection selection 'command-string';
run;
```

- Create a menu bar with an item that opens a dialog box, which displays information and requests text input:

```
proc pmenu;
   menu menu-bar;
```

```
   item 'menu-item' menu=pull-down-menu;
   ...more-ITEM-statements...
   menu pull-down-menu;
      item 'menu-item' dialog=dialog-box;
      dialog dialog-box 'command @1';
         text #line @column 'text';
         text #line @column LEN=field-length;
run;
```

□ Create a menu bar with an item that opens a dialog box, which permits one choice from a list of possible values:

```
proc pmenu;
   menu menu-bar;
   item 'menu-item' menu=pull-down-menu;
   ...more-ITEM-statements...
   menu pull-down-menu;
      item 'menu-item' dialog=dialog-box;
      dialog dialog-box 'command %1';
         text #line @column 'text';
         radiobox default=button-number;
         rbutton #line @column
                 'text-for-selection';
         ...more-RBUTTON-statements...
run;
```

□ Create a menu bar with an item that opens a dialog box, which permits several independent choices:

```
proc pmenu;
   menu menu-bar;
   item 'menu-item' menu=pull-down-menu;
   ...more-ITEM-statements...
   menu pull-down-menu;
      item 'menu-item' dialog=dialog-box;
      dialog dialog-box 'command &1';
         text #line @column 'text';
         checkbox #line @column 'text';
         ...more-CHECKBOX-statements...
run;
```

Examples: PMENU Procedure

The windows in these examples were produced in the UNIX environment and might appear slightly different from the same windows in other operating environments.

You should know the operating environment-specific system options that can affect how menus are displayed and merged with existing SAS menus. For details, see the SAS documentation for your operating environment.

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 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

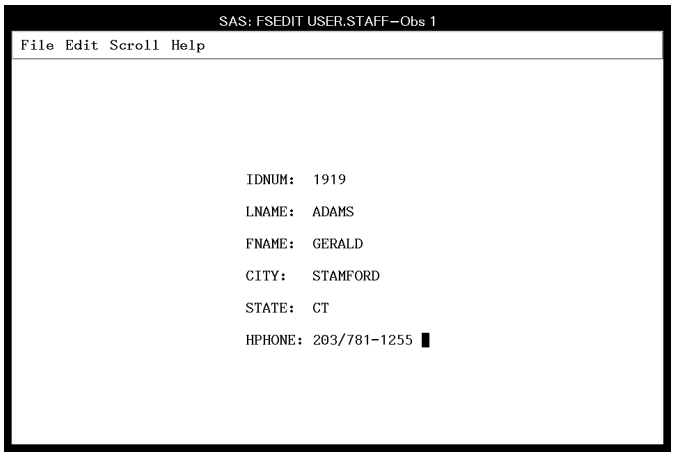
The following SETPMENU command associates the customized menu bar with the FSEDIT window.

```
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 803 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 794. 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

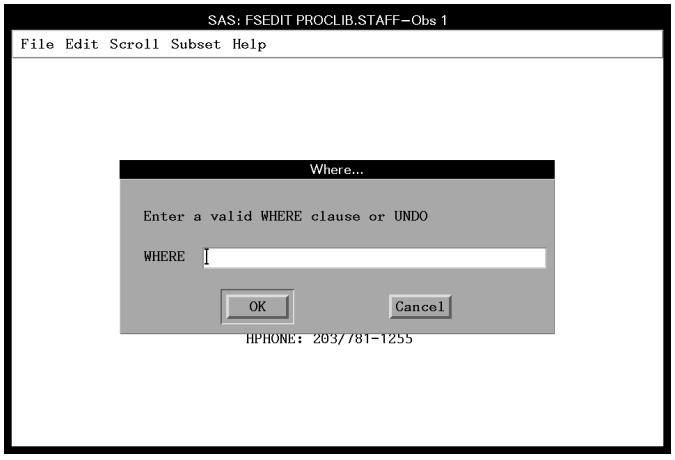
The following SETPMENU command associates the customized menu bar with the FSEDIT window.

```
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 803 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 805.

```
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';
```

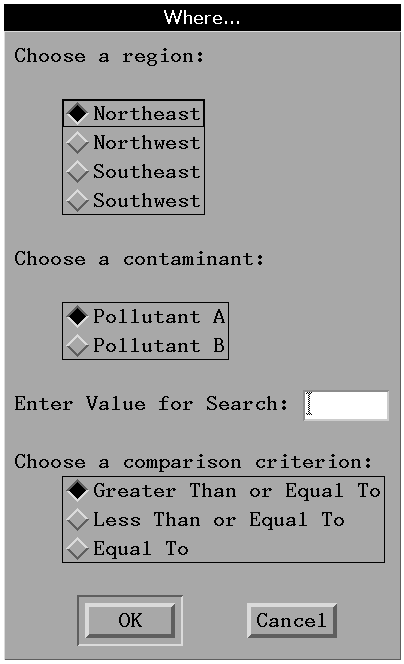
Add an input field. The first TEXT statement specifies text for the dialog box that appears on line 13 and begins in column 1. The second TEXT statement specifies an input field that is 6 bytes long that appears on line 13 and begins in column 25. The value that the user enters in the field is substituted 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 42.1

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 1616 creates PROCLIB.LAKES.

The following statements initiate a PROC FSEDIT session for PROCLIB.LAKES:

```
proc fsedit data=proclib.lakes screen=proclib.lakes;
run;
```

To associate the customized menu bar menu with the FSEDIT session, do any one of the following:

- enter a SETPMENU command on the command line. The command for this example is

  ```
  setpmenu proclib.menucat.project.pmenu
  ```

  Turn on the menus by entering PMENU ON on the command line.
- enter the SETPMENU command in a Command window.
- include an SCL program with the FSEDIT session that uses the customized menus and turns on the menus, for example:

  ```
  fseinit:
    call execcmd('setpmenu proclib.menucat.project.pmenu;
                  pmenu on;');
  return;
  init:
  return;
  main:
  return;
  term:
  return;
  ```

How the WBUILD Macro Works

Consider how you would learn whether any of the lakes in the Southwest region tested for a value of .50 or greater for pollutant A. Without the customized menu item, you would issue the following WHERE command in the FSEDIT window:

```
where region="SW" and (pol_a1 ge .50 or pol_a2 ge .50);
```

Using the custom menu item, you would select **Southwest**, **Pollutant A**, enter .50 as the value, and choose **Greater Than or Equal To** as the comparison criterion. Two lakes, **New Dam** and **Border**, meet the criteria.

The WBUILD macro uses the four pieces of information from the dialog box to generate a WHERE command:

- One of the values for region, either **NE**, **NW**, **SE**, or **SW**, becomes the value of the macro parameter REGION.
- Either **pol_a,2** or **pol_b,4** become the values of the PREFIX and NUMVAR macro parameters. The comma is part of the value that is passed to the WBUILD macro and serves to delimit the two parameters, PREFIX and NUMVAR.
- The value that the user enters for the search becomes the value of the macro parameter VALUE.
- The operator that the user chooses becomes the value of the macro parameter OPERATOR.

To see how the macro works, again consider the following example, in which you want to know whether any of the lakes in the southwest tested for a value of .50 or greater for pollutant A. The values of the macro parameters would be

REGION	**SW**
PREFIX	**pol_a**
NUMVAR	2
VALUE	.50
OPERATOR	GE

The first %IF statement checks to make sure that the user entered a value. If a value has been entered, then the macro begins to generate the WHERE command. First, the macro creates the beginning of the WHERE command:

```
where region="SW" and (
```

Next, the %DO loop executes. For pollutant A, it executes twice because NUMVAR=2. In the macro definition, the period in **&prefix.&i** concatenates **pol_a** with **1** and with **2**. At each iteration of the loop, the macro resolves PREFIX, OPERATOR, and VALUE, and it generates a part of the WHERE command. On the first iteration, it generates **pol_a1 GE .50**

The %IF statement in the loop checks to determine whether the loop is working on its last iteration. If it is not working, then the macro makes a compound WHERE command by putting an **OR** between the individual clauses. The next part of the WHERE command becomes **OR pol_a2 GE .50**

The loop ends after two executions for pollutant A, and the macro generates the end of the WHERE command:

```
)
```

Results from the macro are placed on the command line. The following code is the definition of the WBUILD macro. The underlined code shows the parts of the WHERE command that are text strings that the macro does not resolve:

```
%macro wbuild(region,prefix,numvar,value,operator);
       /* check to see if value is present */
    %if &value ne %then %do;
       where region="&region" AND (
               /* If the values are character,     */
               /* enclose &value in double quotation marks. */
         %do i=1 %to &numvar;
             &prefix.&i &operator &value
                /* if not on last variable,  */
                /* generate 'OR'             */
            %if &i ne &numvar %then %do;
                 OR
            %end;
         %end;
          )
    %end;

%mend wbuild;
```

Example 4: Creating Menus for a DATA Step Window Application

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 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 810.) 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 810.) The first two TEXT statements specify text that appears in the dialog box. The third TEXT statement specifies an input field. The name that is entered in this field is substituted for the @1 in the DIALOG statement.

```
dialog deother 'end;pgm;include de;c deptxx @1;submit';
   text #1 @1 'Enter department name';
   text #2 @3 'in the form DEPT99:';
   text #2 @25 len=7;
```

Design the Print_Report menu. This group of statements defines the choices under the **Print_Report** item. These ITEM statements specify that **For Dept01** and **For Dept02** appear in the menu. The value of the SELECTION= option equates to a subsequent SELECTION statement, which contains the string of commands that are actually submitted.

```
menu deptsprt;
   item 'For Dept01' selection=prt1;
   item 'For Dept02' selection=prt2;
   item 'Other Departments' dialog=prother;
```

Specify commands for the Print_Report menu. The commands in single quotation marks are submitted when the user selects **For Dept01** or **For Dept02**. The END command ends the current window and returns to the PROGRAM EDITOR window so that further commands can be submitted. The INCLUDE command includes the SAS statements that print the report. (See "Printing a Program" on page 811.) 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 811.) 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:

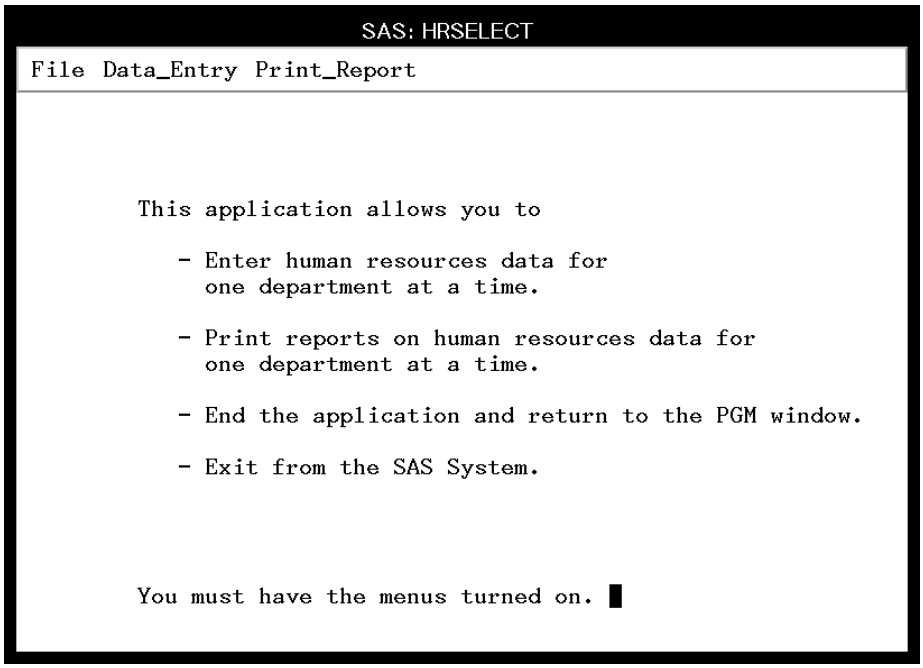
The WINDOW statement creates the HRSELECT window. MENU= associates the PROCLIB.MENUS.SELECT.PMENU entry with this window.

```
   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 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;
```

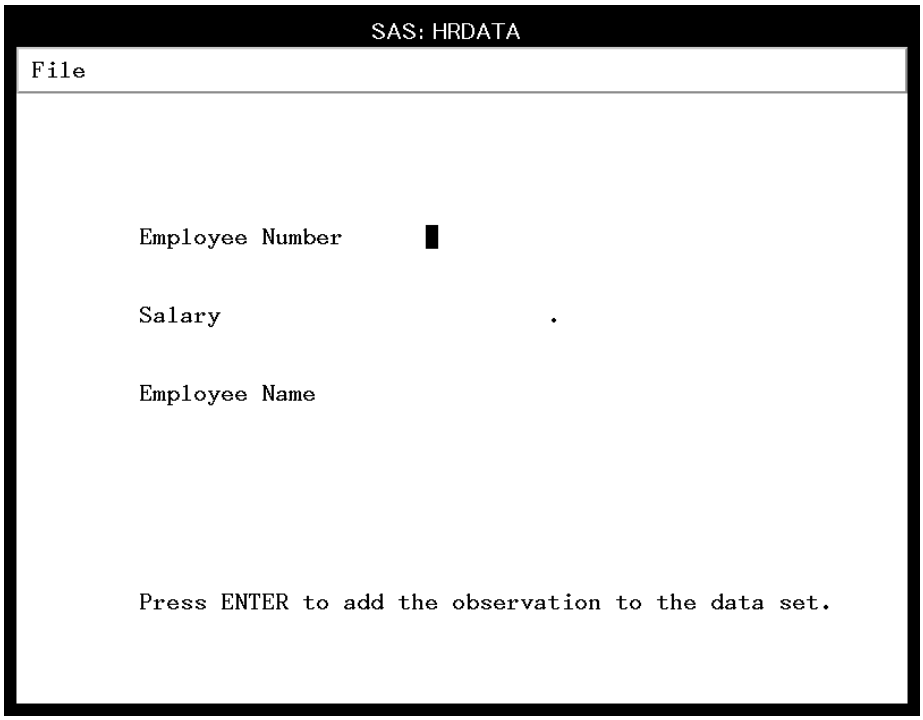
The %INCLUDE statement recalls the statements in the file HRWDW. The statements in HRWDW redisplay the primary window. See the HRSELECT window on page 810.

```
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 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 44, "The PRINTTO Procedure," on page 887 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;
```

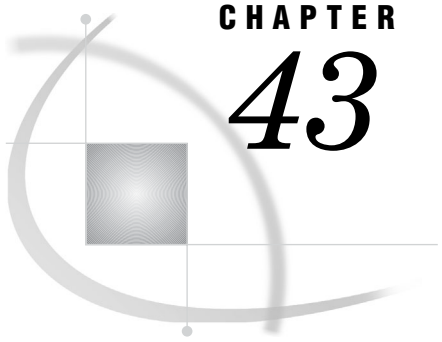
Steps to Associate Menus with a FRAME

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

43

The PRINT Procedure

Overview: PRINT Procedure **815**
What Does the PRINT Procedure Do? **815**
Simple Listing Report **816**
Customized Report **816**
Syntax: PRINT Procedure **818**
PROC PRINT Statement **818**
BY Statement **828**
ID Statement **829**
PAGEBY Statement **830**
SUM Statement **830**
SUMBY Statement **831**
VAR Statement **832**
Results: Print Procedure **832**
Procedure Output **832**
Page Layout **832**
Observations **833**
Column Headings **834**
Column Width **835**
Examples: PRINT Procedure **835**
Example 1: Selecting Variables to Print **835**
Example 2: Customizing Text in Column Headings **841**
Example 3: Creating Separate Sections of a Report for Groups of Observations **845**
Example 4: Summing Numeric Variables with One BY Group **852**
Example 5: Summing Numeric Variables with Multiple BY Variables **856**
Example 6: Limiting the Number of Sums in a Report **865**
Example 7: Controlling the Layout of a Report with Many Variables **871**
Example 8: Creating a Customized Layout with BY Groups and ID Variables **876**
Example 9: Printing All the Data Sets in a SAS Library **882**

Overview: PRINT Procedure

What Does the PRINT Procedure Do?

The PRINT procedure prints the observations in a SAS data set, using all or some of the variables. You can create a variety of reports ranging from a simple listing to a

highly customized report that groups the data and calculates totals and subtotals for numeric variables.

Simple Listing Report

The following output illustrates the simplest kind of report that you can produce. The statements that produce the output follow. Example 1 on page 835 creates the data set EXPREV.

```
options nodate pageno=1 linesize=64 pagesize=60 obs=10;


proc print data=exprev;
run;
```

Output 43.1 Simple Listing Report Produced with PROC PRINT

```
                        The SAS System                          1

                                                      Order_
 Obs    Country                       Emp_ID          Date

   1    Antarctica                    99999999        1/1/08
   2    Puerto Rico                   99999999        1/1/08
   3    Virgin Islands (U.S.)         99999999        1/1/08
   4    Aruba                         99999999        1/1/08
   5    Bahamas                       99999999        1/1/08
   6    Bermuda                       99999999        1/1/08
   7    Belize                        120458          1/2/08
   8    British Virgin Islands        99999999        1/2/08
   9    Canada                        99999999        1/2/08
  10    Cayman Islands                120454          1/2/08

        Ship_       Sale_
 Obs     Date        Type       Quantity    Price      Cost

   1    1/7/08     Internet          2        92.6     20.70
   2    1/5/08     Catalog          14        51.2     12.10
   3    1/4/08     In Store         25        31.1     15.65
   4    1/4/08     Catalog          30       123.7     59.00
   5    1/4/08     Catalog           8       113.4     28.45
   6    1/4/08     Catalog           7        41.0      9.25
   7    1/2/08     In Store          2       146.4     36.70
   8    1/5/08     Catalog          11        40.2     20.20
   9    1/5/08     Catalog         100        11.8      5.00
  10    1/2/08     In Store         20        71.0     32.30
```

Customized Report

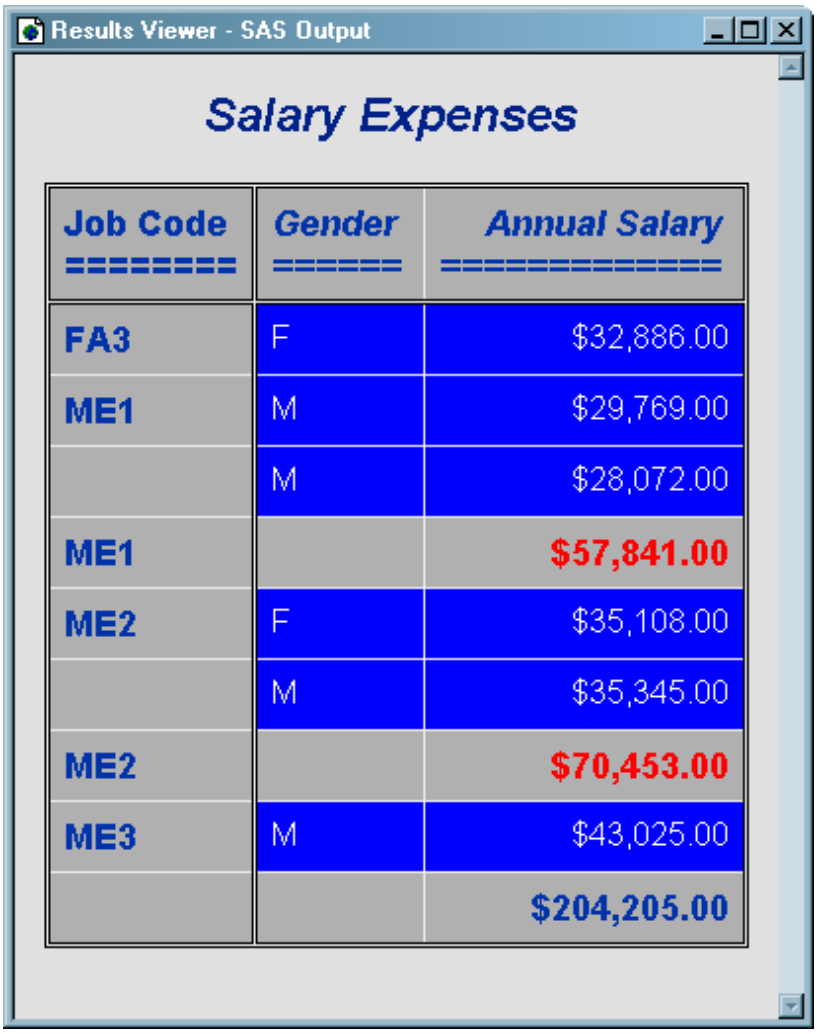
The following HTML report is a customized report that is produced by PROC PRINT using ODS. The statements that create this report do the following:

- create HTML output
- customize the appearance of the report
- customize the title and the column headings
- place dollar signs and commas in numeric output
- selectively include and control the order of variables in the report

- group the data by JobCode
- sum the values for Salary for each job code and for all job codes.

For an explanation of the program that produces this report, see "Program: Creating an HTML Report with the STYLE Option" on page 880.

Display 43.1 Customized Report Produced by PROC PRINT Using ODS



Syntax: PRINT Procedure

Tip: Supports the Output Delivery System. See "Output Delivery System: Basic Concepts in *SAS Output Delivery System: User's Guide* for details.

Tip: You can use the ATTRIB, FORMAT, LABEL, and WHERE statements. See Chapter 3, "Statements with the Same Function in Multiple Procedures," on page 35 for details. You can also use any global statements. See "Global Statements" on page 20 for a list.

PROC 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*>;

Task	Statement
Print observations in a data set.	"PROC PRINT Statement" on page 818
Produce a separate section of the report for each BY group	"BY Statement" on page 828
Identify observations by the formatted values of the variables that you list instead of by observation numbers	"ID Statement" on page 829
Control page ejects that occur before a page is full	"PAGEBY Statement" on page 830
Limit the number of sums that appear in the report	"SUMBY Statement" on page 831
Total values of numeric variables	"SUM Statement" on page 830
Select variables that appear in the report and determine their order	"VAR Statement" on page 832

PROC PRINT Statement

PROC PRINT <*option(s)*>;

Task	Option
Specify text for the HTML contents link to the output	CONTENTS=

Task	Option
Specify the input data set	DATA=
Control general format	
Write a blank line after *n* observations	BLANKLINE
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 heading 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
Display the BY variable label on the summary line	SUMLABEL
Determine the column width for each variable	WIDTH=

Options

BLANKLINE= *n*
BLANKLINE= (COUNT=*n* <STYLE= [*style-attibute-specification(s)*] >)
specifies to insert a blank line after every *n* observations. The observation count is reset at the beginning of each page and at the beginning of each BY group for all ODS destinations except for the RTF and PDF destination. For the RTF and PDF destinations, the observation count is reset only at the beginning of a BY group.

n | COUNT = *n*
specifies the observation number after which SAS inserts a blank line.

STYLE = [*style-attribute-specification(s)*]
specifies the style to use for the blank line.

Default: DATA

Tip: You can use the BACKGROUNDCOLOR style element to make a visual distinction between observations using color.

See: The STYLE option for valid style attributes.

Featured in: Example 1 on page 835

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 about 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 20

DOUBLE

writes a blank line between observations.

Alias: D

Restriction: This option is valid only for the listing destination.

Featured in: Example 1 on page 835

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 specifies whether column headings are vertical or horizontal.
- If you use LABEL and at least one variable has a label, all headings are horizontal.

LABEL

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 might 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 heading 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 35.

For information about using the LABEL statement in a DATA step to create permanent labels, see LABEL Statement in *SAS Language Reference: Dictionary*.

Featured in: Example 3 on page 845

Note: The SAS system option LABEL must be in effect in order for any procedure to use labels. For more information see LABEL System Option in *SAS 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.

N Option Use	PROC PRINT Action
with neither a BY nor a SUM statement	prints the number of observations in the data set at the end of the report and labels the number with the value of *string-1*.
with a BY statement	prints the number of observations in the BY group at the end of each BY group and labels the number with the value of *string-1*.
with a BY statement and a SUM statement	prints the number of observations in the BY group at the end of each BY group and prints the number of observations in the data set at the end of the report. The numbers for BY groups are labeled with *string-1*; the number for the entire data set is labeled with *string-2*.

Featured in: Example 2 on page 841 (alone)
Example 3 on page 845 (with a BY statement)
Example 4 on page 852 (with a BY statement and a SUM statement)

NOOBS

suppresses the observation number in the output.

Featured in: Example 3 on page 845

OBS="*column-header*"

specifies a column heading for the column that identifies each observation by number.

Tip: OBS= honors the split character (see the discussion of SPLIT= on page 823).

Featured in: Example 2 on page 841

ROUND

rounds unformatted numeric values to two decimal places. (Formatted values are already rounded by the format to the specified number of decimal places.) For both formatted and unformatted variables, PROC PRINT uses these rounded values to calculate any sums in the report.

If you omit ROUND, PROC PRINT adds the actual values of the rows to obtain the sum *even though it displays the formatted (rounded) values*. Any sums are also

rounded by the format, but they include only one rounding error, that of rounding the sum of the actual values. The ROUND option, on the other hand, rounds values before summing them, so there might be multiple rounding errors. The results without ROUND are more accurate, but ROUND is useful for published reports where it is important for the total to be the sum of the printed (rounded) values.

Be aware that the results from PROC PRINT with the ROUND option might differ from the results of summing the same data with other methods such as PROC MEANS or the DATA step. Consider a simple case in which

- the 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 43.1 Three Methods of Summing Variables

Actual Values	PROC PRINT without the ROUND option		PROC PRINT with the ROUND option		PROC MEANS
	OBS	X	OBS	X	Analysis Variable : X
.003	1	0.00	1	0.00	Sum
.004	2	0.00	2	0.00	0.0160000
.009	3	0.01	3	0.01	
.016		0.02		0.01	

Notice that the sum produced without the ROUND option (.02) is closer to the actual result (0.16) than the sum produced with ROUND (0.01). However, the sum produced with ROUND reflects the numbers displayed in the report.

Alias: R

CAUTION:

Do not use ROUND with PICTURE formats. ROUND is for use with numeric values. SAS procedures treat variables that have picture formats as character variables. Using ROUND with such variables might lead to unexpected results. △

ROWS= *page-format*

formats 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 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 832 for discussion of the default layout.

Featured in: Example 7 on page 871

SPLIT='*split-character*'
specifies the split character, which controls line breaks in column headings. It also uses labels as column headings. PROC PRINT breaks a column heading when it reaches the split character and continues the header on the next line. The split character is not part of the column heading although each occurrence of the split character counts toward the 256-character maximum for a label.

Alias: S=

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 821.)

Featured in: Example 2 on page 841

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. 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 example, if you specify a blue background and a white foreground for all column headings in the PROC PRINT statement, and you specify a gray background for the column headings of a variable in the VAR statement, the background for that particular column heading is gray, and the foreground is white (as specified in the PROC PRINT statement). △

Table 43.1 Specifying Locations in the STYLE Option

Location	Affected Report Part	Can Also Be Used In These Statements
BYLABEL	the label for the BY variable on the line containing the SUM totals	none
DATA	the cells of all columns	VAR ID SUM

Location	Affected Report Part	Can Also Be Used In These Statements
GRANDTOTAL	the SUM line containing the grand totals for the whole report	SUM
HEADER	all column headings	VAR ID SUM
N	N= table and contents	none
OBS	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 43.2 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

Location	Aliases
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 43.3 The Default Style Element for Each Location in PROC PRINT

Location	Default Style Element
BYLABEL	Header
DATA	Data (for all but ID statement)
	RowHeader (for ID statement)
GRANDTOTAL	Header
HEADER	Header
N	NoteContent
OBS	RowHeader
OBSHEADER	Header
TABLE	Table
TOTAL	Header

style-attribute-specification
describes the style attribute to change. Each *style-attribute-specification* has this general form:

style-attribute-name=*style-attribute-value*

You can set these style attributes in the TABLE location:

BACKGROUNDCOLOR=
BACKGROUNDIMAGE=
BORDERCOLOR=
BORDERCOLORDARK=
BORDERCOLORLIGHT=
BORDERWIDTH=
CELLPADDING=
CELLSPACING=
FONTWIDTH=*
COLOR=*
FRAME=
HTMLCLASS=
TEXTALIGN=
OUTPUTWIDTH=
POSTHTML=
POSTIMAGE=

FONT=*	POSTTEXT=
FONTFAMILY=*	PREHTML=
FONTSIZE=*	PREIMAGE=
FONTSTYLE=*	PRETEXT=
FONTWEIGHT=*	RULES=

*When you use these attributes, they affect only the text that is specified with the PRETEXT=, POSTTEXT=, PREHTML=, and POSTHTML= attributes. To alter the foreground color or the font for the text that appears in the table, you must set the corresponding attribute in a location that affects the cells rather than the table.

You can set these style attributes in all locations other than TABLE:

ASIS=	FONTWIDTH=
BACKGROUNDCOLOR=	HREFTARGET=
BACKGROUNDIMAGE=	CLASS=
BORDERCOLOR=	TEXTALIGN=
BORDERCOLORDARK=	NOBREAKSPACE=
BORDERCOLORLIGHT=	POSTHTML=
BORDERWIDTH=	POSTIMAGE=
HEIGHT=	POSTTEXT=
CELLWIDTH=	PREHTML=
FLYOVER=	PREIMAGE=
FONT=	PRETEXT=
FONTFAMILY=	PROTECTSPECIALCHARACTERS=
FONTSIZE=	TAGATTR=
FONTSTYLE=	URL=
FONTWEIGHT=	VERTICALALIGN=

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.

SUMLABEL

uses BY variable labels in the summary line in place of the BY variable name.

Default: If you omit SUMLABEL, PROC PRINT uses the BY variable names in the summary line.

Featured in:

Example 4 on page 852
Example 5 on page 856

Note: The SAS system option LABEL must be in effect in order for any procedure to use labels. For more information, see the LABEL System Option in *SAS Language Reference: Dictionary*: △

UNIFORM
See WIDTH=UNIFORM on page 827.

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 might 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 might lessen the usefulness of WIDTH=.

See also: For a discussion of default column widths, see "Column Width" on page 835.

BY Statement

Produces a separate section of the report for each BY group.

Main discussion: "BY" on page 36

Featured in:
Example 3 on page 845
Example 4 on page 852
Example 5 on page 856
Example 6 on page 865
Example 8 on page 876

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 is grouped in another way, such as chronological order.

The requirement for ordering or indexing observations according to the values of BY variables is suspended for BY-group processing when you use the NOTSORTED option. In fact, the procedure does not use an index if you specify NOTSORTED. The procedure defines a BY group as a set of contiguous observations that have the same values for all BY variables. If observations with the same values for the BY variables are not contiguous, the procedure treats each contiguous set as a separate BY group.

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 876.)

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 871

Example 8 on page 876

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 823 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 876.)

PAGEBY Statement

Controls page ejects that occur before a page is full.

Requirements: BY statement

Featured in: Example 3 on page 845

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 21.)

SUM Statement

Totals values of numeric variables.

Featured in:
Example 4 on page 852
Example 5 on page 856
Example 6 on page 865
Example 8 on page 876

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 823 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 852).

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 856.)

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 865

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 for the variables listed in the ID and BY statements.

VAR Statement

Selects variables that appear in the report and determines their order.

Tip: If you omit the VAR statement, PROC PRINT prints all variables in the data set.

Featured in:

Example 1 on page 835
Example 8 on page 876

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 823 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

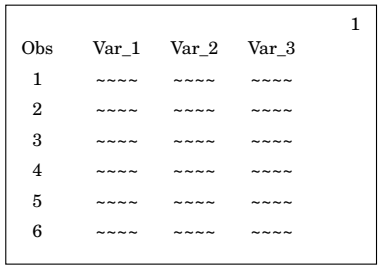
PROC PRINT always produces a printed report. You control the appearance of the report with statements and options. See “Examples: PRINT Procedure ”on page 835 for a sampling of the types of reports that the procedure produces.

Page Layout

Observations

By default, PROC PRINT uses an identical layout for all observations on a page of output. First, it attempts to print observations on a single line, as shown in the following figure.

Figure 43.2 Printing Observations on a Single Line

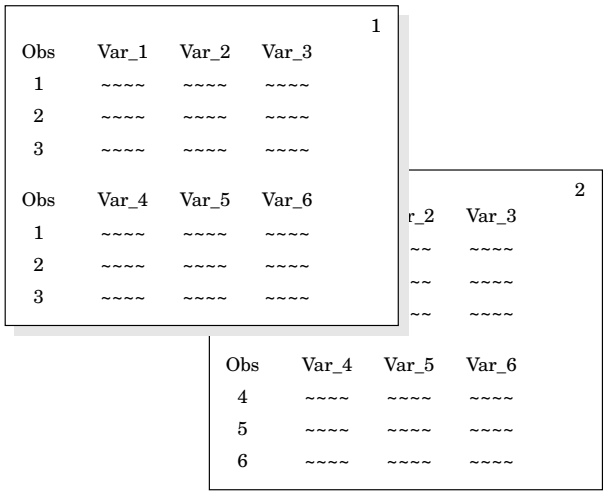


1

Obs	Var_1	Var_2	Var_3
1	~~~~	~~~~	~~~~
2	~~~~	~~~~	~~~~
3	~~~~	~~~~	~~~~
4	~~~~	~~~~	~~~~
5	~~~~	~~~~	~~~~
6	~~~~	~~~~	~~~~

If PROC PRINT cannot fit all the variables on a single line, it splits the observations into two or more sections and prints the observation number or the ID variables at the beginning of each line. For example, in the following figure, PROC PRINT prints the values for the first three variables in the first section of each page and the values for the second three variables in the second section of each page.

Figure 43.3 Splitting Observations into Multiple Sections on One Page



1

Obs	Var_1	Var_2	Var_3
1	~~~~	~~~~	~~~~
2	~~~~	~~~~	~~~~
3	~~~~	~~~~	~~~~

Obs	Var_4	Var_5	Var_6
1	~~~~	~~~~	~~~~
2	~~~~	~~~~	~~~~
3	~~~~	~~~~	~~~~

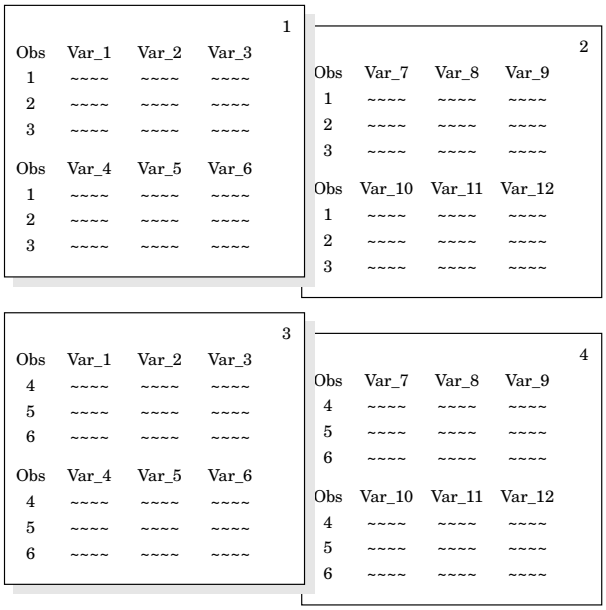
2

r_2	Var_3
~~	~~~~
~~	~~~~
~~	~~~~

Obs	Var_4	Var_5	Var_6
4	~~~~	~~~~	~~~~
5	~~~~	~~~~	~~~~
6	~~~~	~~~~	~~~~

If PROC PRINT cannot fit all the variables on one page, the procedure prints subsequent pages with the same observations until it has printed all the variables. For example, in the following figure, PROC PRINT uses the first two pages to print values for the first three observations and the second two pages to print values for the rest of the observations.

Figure 43.4 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	~~~~	~~~~	~~~~

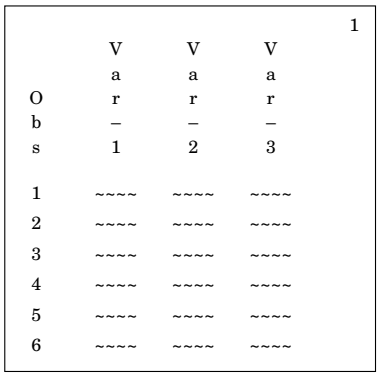
Note: You can alter the page layout with the ROWS= option in the PROC PRINT statement (see the discussion of ROWS= on page 822). △

Note: PROC PRINT might produce slightly different output if the data set is not RADIX addressable. Version 6 compressed files are not RADIX addressable, while, beginning with Version 7, compressed files are RADIX addressable. (The integrity of the data is not compromised; the procedure simply numbers the observations differently.) △

Column Headings

By default, spacing specifies whether PROC PRINT prints column headings horizontally or vertically. Figure 43.2 on page 833, Figure 43.3 on page 833, and Figure 43.4 on page 834 all illustrate horizontal headings. The following figure illustrates vertical headings.

Figure 43.5 Using Vertical Headings



1

Obs	Var 1	Var 2	Var 3
1	~~~~	~~~~	~~~~
2	~~~~	~~~~	~~~~
3	~~~~	~~~~	~~~~
4	~~~~	~~~~	~~~~
5	~~~~	~~~~	~~~~
6	~~~~	~~~~	~~~~

Note: If you use LABEL and at least one variable has a label, PROC PRINT prints all column headings horizontally unless you specify HEADING=VERTICAL. △

Column Width

By default, PROC PRINT uses a variable's formatted width as the column width. (The WIDTH= option overrides this default behavior.) If the variable does not have a format that explicitly specifies a field width, PROC PRINT uses the widest data value for that variable on that page as the column width.

If the formatted value of a character variable or the data width of an unformatted character variable exceeds the line size minus the length of all the ID variables, PROC PRINT might truncate the value. Consider the following situation:

- The line size is 80.
- IdNumber is a character variable with a length of 10. It is used as an ID variable.
- State is a character variable with a length of 2. It is used as an ID variable.
- Comment is a character variable with a length of 200.

When PROC PRINT prints these three variables on a line, it uses 14 print positions for the two ID variables and the space after each one. This arrangement leaves 80–14, or 66, print positions for COMMENT. Longer values of COMMENT are truncated.

WIDTH= controls the column width.

Note: Column width is affected not only by variable width but also by the length of column headings. Long column headings might lessen the usefulness of WIDTH=. △

Examples: PRINT Procedure

Example 1: Selecting Variables to Print

Procedure features:

PROC PRINT statement options:

BLANKLINE
DOUBLE
STYLE

VAR statement

Other Features:

DATA step

FOOTNOTE statement

ODS HTML statement

OPTIONS statement

TITLE statement

Data set: "EXPREV" on page 1610

Program Description

This example

- selects three variables for the reports
- uses variable labels as column headings
- double spaces between rows of the report
- creates a default HTML report

- creates a stylized HTML report.

Program: Creating a Listing Report

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. The PAGENO= option specifies the starting page number. The LINESIZE= option specifies the output line length, and the PAGESIZE= option specifies the number of lines on an output page. The OBS= option specifies the number of observations to display.

```
options nodate pageno=1 linesize=80 pagesize=30 obs=10;
```

Print the data set EXPREV. EXPREV contains information about a company's product order type and price per unit for two months. DOUBLE inserts a blank line between observations. The DOUBLE option has no effect on the HTML output.

```
proc print data=exprev double;
```

Select the variables to include in the report. The VAR statement creates columns for Country, Price, and Sale_Type, in that order.

```
   var country price sale_type;
```

Specify a title and a footnote. The TITLE statement specifies the title for the report. The FOOTNOTE statement specifies a footnote for the report.

```
   title 'Monthly Price Per Unit and Sale Type for Each Country';
   footnote '*prices in USD';
run;
```

Output: Listing

Output 43.2 Selecting Variables: Listing Output

By default, PROC PRINT identifies each observation by number under the column heading **Obs.**

Monthly Price Per Unit and Sale Type for Each Country 1

Obs	Country	Price	Sale_ Type
1	Antarctica	92.6	Internet
2	Puerto Rico	51.2	Catalog
3	Virgin Islands (U.S.)	31.1	In Store
4	Aruba	123.7	Catalog
5	Bahamas	113.4	Catalog
6	Bermuda	41.0	Catalog
7	Belize	146.4	In Store
8	British Virgin Islands	40.2	Catalog
9	Canada	11.8	Catalog
10	Cayman Islands	71.0	In Store

*prices in USD

Output 43.3 Selecting Variables: Listing Output

Monthly Price Per Unit and Sale Type for Each Country 1

Obs	Country	Price	Sale_ Type
1	Antarctica	92.6	Internet
2	Puerto Rico	51.2	Catalog
3	Virgin Islands (U.S.)	31.1	In Store
4	Aruba	123.7	Catalog
5	Bahamas	113.4	Catalog
6	Bermuda	41.0	Catalog
7	Belize	146.4	In Store
8	British Virgin Islands	40.2	Catalog
9	Canada	11.8	Catalog
10	Cayman Islands	71.0	In Store

*prices in USD

Program: Creating an HTML Report

You can easily create HTML output by adding ODS statements. In the following example, ODS statements are added to produce HTML output.

```
options nodate pageno=1 linesize=80 pagesize=30 obs=10;
```

Create HTML output and specify the file to store the output in. The ODS HTML statement opens the HTML destination. The FILE= option specifies the external file that you want to contain the HTML output.

```
ods html file='your_file.html';

proc print data=exprev double;

   var country price sale_type;
   title 'Monthly Price Per Unit and Sale Type for Each Country';
   footnote '*prices in USD';
run;

run;
```

Close the HTML destination. The ODS HTML CLOSE statement closes the HTML destination.

```
ods html close;
```

Output: HTML

Display 43.2 Selecting Variables: Default HTML Output

Results Viewer - SAS Output

Monthly Price Per Unit and Sale Type for Each Country

Obs	Country	Price	Sale_Type
1	Antarctica	92.6	Internet
2	Puerto Rico	51.2	Catalog
3	Virgin Islands (U.S.)	31.1	In Store
4	Aruba	123.7	Catalog
5	Bahamas	113.4	Catalog
6	Bermuda	41.0	Catalog
7	Belize	146.4	In Store
8	British Virgin Islands	40.2	Catalog
9	Canada	11.8	Catalog
10	Cayman Islands	71.0	In Store

**prices in USD*

Program: Creating an HTML Report with the STYLE and BLANKLINE Options

You can go a step further and add more formatting to your HTML output. The following example uses the STYLE option to add shading and spacing to your HTML report.

```
options nodate pageno=1 linesize=80 pagesize=40 obs=5;

ods html file='your_file_styles.html';
```

Create stylized HTML output. The first STYLE option specifies that the column headings be written in white italic font. The second STYLE option specifies that ODS change the color of the background of the observations column to red. The BLANKLINE option specifies to add a blank line between each observation and use a background color of red.

```
proc print data=exprev double
```

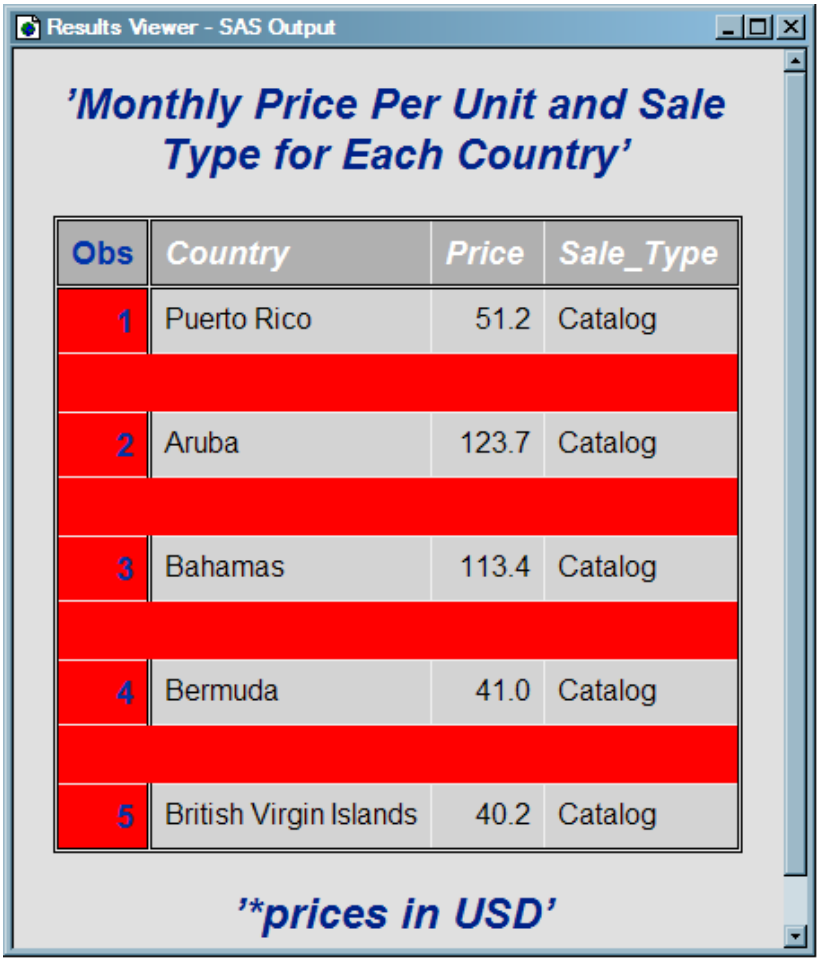
```
    style(header) = {fontstyle=italic color= white}
    style(obs) = {backgroundcolor=red}
    blankline=(count=1 style={backgroundcolor=red});

   var country price sale_type;

   title 'Monthly Price Per Unit and Sale Type for Each Country';
   footnote '*prices in USD';
run;

run;
```

Close the HTML destination. The ODS HTML CLOSE statement closes the HTML destination.

```
ods html close;
```

Output: HTML Output with Styles

Display 43.3 Selecting Variables: HTML Output Using Styles



Obs	Country	Price	Sale_Type
1	Puerto Rico	51.2	Catalog
2	Aruba	123.7	Catalog
3	Bahamas	113.4	Catalog
4	Bermuda	41.0	Catalog
5	British Virgin Islands	40.2	Catalog

Example 2: Customizing Text in Column Headings

Procedure features:

PROC PRINT statement options:

N
OBS=
SPLIT=
STYLE

VAR statement option:

STYLE

Other features:

LABEL statement
ODS PDF statement
FORMAT statement
TITLE statement

Data set: "EXPREV" on page 1610

This example

- customizes and underlines the text in column headings for variables
- customizes the column heading for the column that identifies observations by number
- shows the number of observations in the report
- writes the values of the variable Price with dollar signs and periods.
- creates a default PDF report
- creates a stylized PDF report.

Program: Creating a Listing Report

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. The PAGENO= option specifies the starting page number. The LINESIZE= option specifies the output line length, and the PAGESIZE= option specifies the number of lines on an output page. The OBS= option specifies the number of observations to be displayed.

```
options nodate pageno=1 linesize=80 pagesize=30 obs=10;
```

Print the report and define the column headings. SPLIT= identifies the asterisk as the character that starts a new line in column headings. The N option prints the number of observations at the end of the report. OBS= specifies the column heading for the column that identifies each observation by number. The split character (*) starts a new line in the column heading. The equal signs (=) in the value of OBS= underlines the column heading.

```
proc print data=exprev split='*' n obs='Observation*Number*==========';
```

Select the variables to include in the report. The VAR statement creates columns for Country, Sale_Type, and Price, in that order.

```
var country sale_type price;
```

Assign the variables' labels as column headings. The LABEL statement associates a label with each variable for the duration of the PROC PRINT step. When you use the SPLIT= option in the PROC PRINT statement, the procedure uses labels for column headings. The split character (*) starts a new line in the column heading. The equal signs (=) in the labels underlines the column headings.

```
label country='Country Name**============'
      sale_type='Order Type**==========='
      price='Price Per Unit*in USD*==============';
```

Specify a title for the report, and format any variable containing numbers. The FORMAT statement assigns the DOLLAR10.2 format to the variable Price in the report. The TITLE statement specifies a title.

```
   format price dollar10.2;
   title 'Order Type and Price Per Unit in Each Country';
run;
```

Output: Listing

Output 43.4 Customizing Text in Column Headings: Listing Output

Order Type and Price Per Unit in Each Country 1

Observation Number	Country Name	Order Type	Price Per Unit in USD
1	Antarctica	Internet	$92.60
2	Puerto Rico	Catalog	$51.20
3	Virgin Islands (U.S.)	In Store	$31.10
4	Aruba	Catalog	$123.70
5	Bahamas	Catalog	$113.40
6	Bermuda	Catalog	$41.00
7	Belize	In Store	$146.40
8	British Virgin Islands	Catalog	$40.20
9	Canada	Catalog	$11.80
10	Cayman Islands	In Store	$71.00

N = 10

Program: Creating a PDF Report

You can easily create PDF output by adding a few ODS statements. In the following example, ODS statements were added to produce PDF output.

```
options nodate pageno=1 linesize=80 pagesize=40 obs=10;
```

Create PDF output and specify the file to store the output in. The ODS PDF statement opens the PDF destination and creates PDF output. The FILE= argument specifies the external file that contains the PDF output.

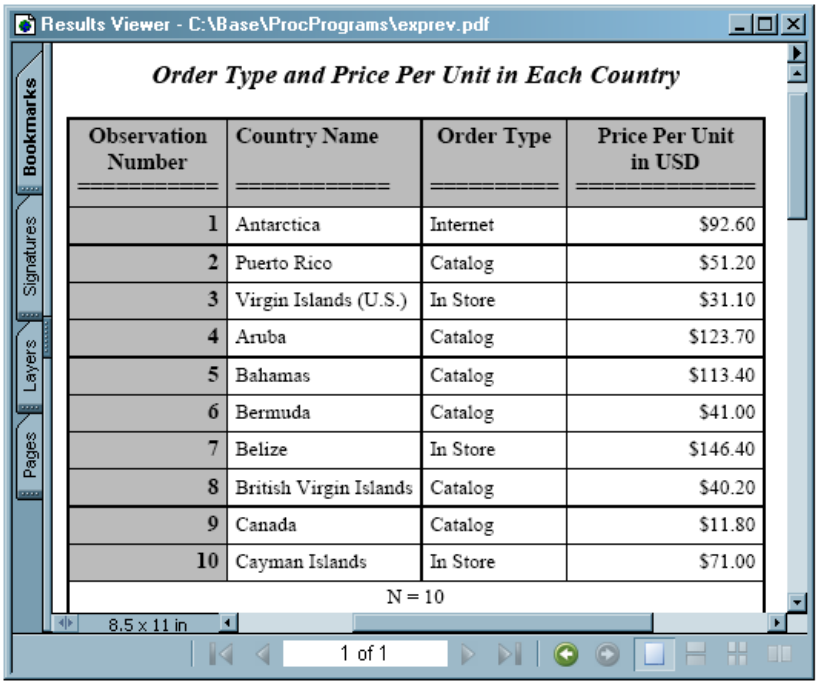
```
ods pdf file='your_file.pdf';

proc print data=exprev split='*' n obs='Observation*Number*==========';
   var country sale_type price;
   label country='Country Name**============'
         sale_type='Order Type**==========='
         price='Price Per Unit in USD**==============';


   format price dollar10.2;
   title 'Order Type and Price Per Unit in Each Country';
run;
```

Close the PDF destination. The ODS PDF CLOSE statement closes the PDF destination.

```
ods pdf close;
```

Output: PDF

Display 43.4 Customizing Text in Column Headings: Default PDF Output



Order Type and Price Per Unit in Each Country

Observation Number	Country Name	Order Type	Price Per Unit in USD
1	Antarctica	Internet	$92.60
2	Puerto Rico	Catalog	$51.20
3	Virgin Islands (U.S.)	In Store	$31.10
4	Aruba	Catalog	$123.70
5	Bahamas	Catalog	$113.40
6	Bermuda	Catalog	$41.00
7	Belize	In Store	$146.40
8	British Virgin Islands	Catalog	$40.20
9	Canada	Catalog	$11.80
10	Cayman Islands	In Store	$71.00
N = 10			

Program: Creating a PDF Report with the STYLE Option

```
options nodate pageno=1 linesize=80 pagesize=40 obs=10;

ods pdf file='your_file.pdf';
```

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 grey.

```
proc print data=exprev split='*' n obs='Observation*Number*==========='
     style(n) = {fontstyle=italic backgrouncolor= blue}
     style(header obs obsheader) = {backgrouncolor=yellow color=blue};
```

Create stylized PDF output. The STYLE option changes the color of the cells containing data to gray.

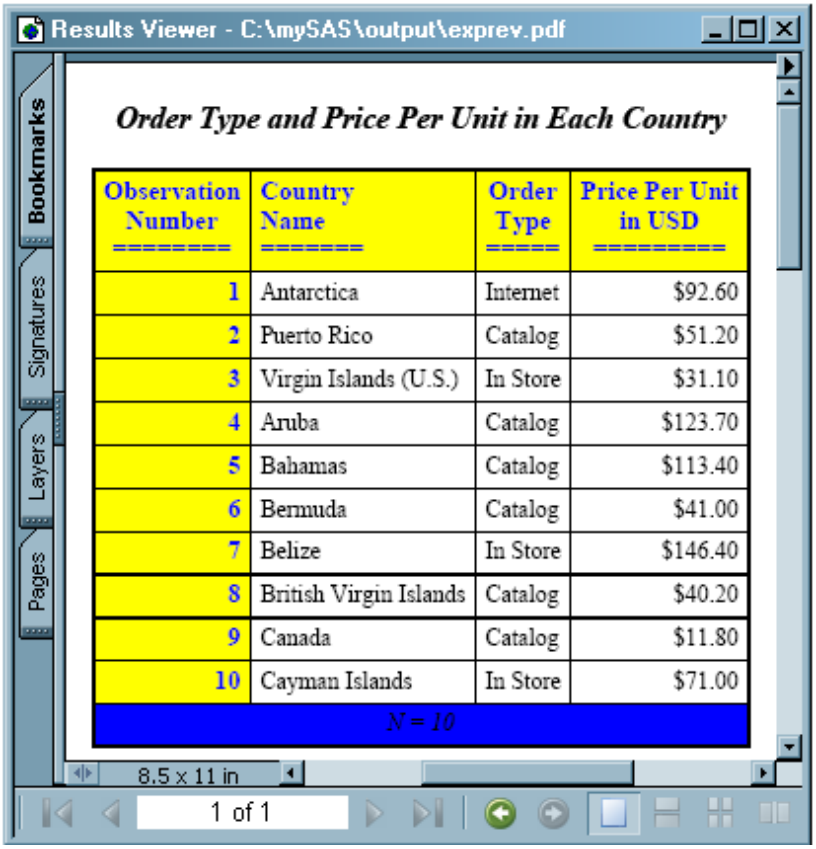
```
   var country sale_type price / style (data)= [ background = gray ];
   label country='Country Name**======'
         sale_type='Order Type**====='
         price='Price Per Unit in USD**========';
   format price dollar10.2;
run;

title 'Order Type and Price Per Unit in Each Country';
run;
```

Close the PDF destination. The ODS PDF CLOSE statement closes the PDF destination.

```
ods pdf close;
```

Output: PDF Report with Styles

Display 43.5 Customizing Text in Column Headings: PDF Output Using Styles



Order Type and Price Per Unit in Each Country

Observation Number ========	Country Name =======	Order Type =====	Price Per Unit in USD =========
1	Antarctica	Internet	$92.60
2	Puerto Rico	Catalog	$51.20
3	Virgin Islands (U.S.)	In Store	$31.10
4	Aruba	Catalog	$123.70
5	Bahamas	Catalog	$113.40
6	Bermuda	Catalog	$41.00
7	Belize	In Store	$146.40
8	British Virgin Islands	Catalog	$40.20
9	Canada	Catalog	$11.80
10	Cayman Islands	In Store	$71.00
N = 10			

Example 3: Creating Separate Sections of a Report for Groups of Observations

Procedure features:

PROC PRINT statement options:

LABEL
N=
NOOBS
STYLE

BY statement

PAGEBY statement

Other features:

SORT procedure

FORMAT statement

LABEL statement

ODS RTF statement

TITLE statement

Data set: "EXPREV" on page 1610

This example

- suppresses the printing of observation numbers at the beginning of each row
- presents the data for each sale type in a separate section of the report
- creates a default RTF report
- creates a stylized RTF report.

Program: Creating a Listing Report

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. The PAGENO= option specifies the starting page number. The LINESIZE= option specifies the output line length, and the PAGESIZE= option specifies the number of lines on an output page. The OBS= option specifies the number of observations to be displayed.

```
options nodate pageno=1 linesize=80 pagesize=40 obs=10;
```

Sort the EXPREV data set. PROC SORT sorts the observations by Sale_Type, Order_Date, and Quantity.

```
proc sort data=exprev;
   by sale_type order_date quantity;
run;
```

Print the report, specify the total number of observations in each BY group, and suppress the printing of observation numbers. N= prints the number of observations in a BY group at the end of that BY group. The explanatory text that the N= option provides precedes the number. NOOBS suppresses the printing of observation numbers at the beginning of the rows. LABEL uses variables' labels as column headings.

```
proc print data=exprev n='Number of observations for the month: '
           noobs label;
```

Specify the variables to include in the report. The VAR statement creates columns for Quantity, Cost, and Price, in that order.

```
    var quantity cost price;
```

Create a separate section for each order type and specify page breaks for each BY group of Order_Date. The BY statement produces a separate section of the report for each BY group and prints a heading above each one. The PAGEBY statement starts a new page each time the value of Order_Date changes.

```
    by sale_type order_date;
    pageby order_date;
```

Establish the column headings. The LABEL statement associates labels with the variables Sale_Type and Order_Date for the duration of the PROC PRINT step. When you use the LABEL option in the PROC PRINT statement, the procedure uses labels for column headings.

```
    label sale_type='Order Type' order_date='Order Date';
```

Format the columns that contain numbers and specify a title and footnote. The FORMAT statement assigns a format to Price and Cost for this report. The TITLE statement specifies a title. The TITLE2 statement specifies a second title.

```
    format price dollar7.2 cost dollar7.2;
    title 'Prices and Cost Grouped by Date and Order Type';
    title2 'in USD';
run;
```

Output: Listing

Output 43.5 Creating Separate Sections of a Report for Groups of Observations: Listing Output

```
                Prices and Cost Grouped by Date and Order Type                1
                                in USD

---------------------- Order Type=Catalog Order Date=1/1/08 ----------------------

                          Quantity       Cost      Price

                              7          $9.25     $41.00
                              8         $28.45    $113.40
                             14         $12.10     $51.20
                             30         $59.00    $123.70

                     Number of observations for the month: 4
```

```
                Prices and Cost Grouped by Date and Order Type                2
                                in USD

---------------------- Order Type=Catalog Order Date=1/2/08 ----------------------

                          Quantity       Cost      Price

                             11        $20.20     $40.20
                            100         $5.00     $11.80

                     Number of observations for the month: 2
```

```
                Prices and Cost Grouped by Date and Order Type                3
                                in USD

--------------------- Order Type=In Store Order Date=1/1/08 ----------------------

                          Quantity       Cost      Price

                             25         $15.65     $31.10

                     Number of observations for the month: 1
```

```
                Prices and Cost Grouped by Date and Order Type                4
                                in USD

--------------------- Order Type=In Store Order Date=1/2/08 ----------------------

                          Quantity       Cost      Price

                              2         $36.70    $146.40
                             20         $32.30     $71.00

                     Number of observations for the month: 2
```

```
                Prices and Cost Grouped by Date and Order Type                5
                                in USD

-------------------- Order Type=Internet Order Date=1/1/08 ---------------------

                         Quantity       Cost      Price

                             2         $20.70     $92.60

                    Number of observations for the month: 1
```

Program: Creating an RTF Report

```
options nodate pageno=1 linesize=80 pagesize=40 obs=10;
```

Create output for Microsoft Word and specify the file to store the output in. The ODS RTF statement opens the RTF destination and creates output formatted for Microsoft Word. The FILE= option specifies the external file that contains the RTF output. The STARTPAGE=NO option specifies that no new pages be inserted explicitly at the start of each by group.

```
ods rtf file='your_file.rtf' startpage=no;

proc sort data=exprev;
   by sale_type order_date quantity;
run;

proc print data=exprev n='Number of observations for each order type:'
           noobs label;
   var quantity cost price;
   by sale_type order_date;
   pageby order_date;
   label sale_type='Order Type' order_date='Order Date';
   format price dollar7.2 cost dollar7.2;
   title 'Price and Cost Grouped by Date and Order Type';
   title2 'in USD';
run;
```

Close the RTF destination. The ODS RTF CLOSE statement closes the RTF destination.

```
ods rtf close;
```

Output: RTF

Display 43.6 Creating Separate Sections of a Report for Groups of Observations: Default RTF Output

Price and Cost Grouped by Date and Order Type in USD

Order Type=Catalog Order Date=1/1/08

Quantity	Cost	Price
7	$9.25	$41.00
8	$28.45	$113.40
14	$12.10	$51.20
30	$59.00	$123.70
Number of observations for each order type:4		

Order Type=Catalog Order Date=1/2/08

Quantity	Cost	Price
11	$20.20	$40.20
100	$5.00	$11.80
Number of observations for each order type:2		

Order Type=In Store Order Date=1/1/08

Quantity	Cost	Price
25	$15.65	$31.10
Number of observations for each order type:1		

Order Type=In Store Order Date=1/2/08

Quantity	Cost	Price
2	$36.70	$146.40
20	$32.30	$71.00
Number of observations for each order type:2		

Order Type=Internet Order Date=1/1/08

Quantity	Cost	Price
2	$20.70	$92.60
Number of observations for each order type:1		

Program: Creating an RTF Report with the STYLE Option

```
options nodate pageno=1 linesize=80 pagesize=40 obs=10;

ods rtf file='your_file.rtf' startpage=no;

proc sort data=exprev;
by sale_type order_date quantity;
run;
```

Create a stylized RTF report. The first STYLE option specifies that the background color of the cell containing the number of observations be changed to gray. The second STYLE option specifies that the background color of the column heading for the variable Quantity be changed to white. The third STYLE option specifies that the background color of the column heading for the variable Cost be changed to blue and the font color be changed to white. The fourth STYLE option specifies that the background color of the column heading for the variable Sale_Type be changed to gray.

```
proc print data=exprev n='Number of observations for the month: '
           noobs label style(N) = {background = gray};

   var quantity / style(header) = [background = white];
   var cost / style(header) = [background = blue foreground = white];
   var price / style(header) = [background = gray];
   by sale_type order_date;
   pageby order_date;
   label sale_type='Order Type' order_date='Order Date';
   format price dollar7.2 cost dollar7.2;

title 'Prices and Cost Grouped by Date and Order Type';
title2 '*prices in USD';
run;

ods rtf close;
```

Output: RTF with Styles

Display 43.7 Creating Separate Sections of a Report for Groups of Observations: RTF Output Using Styles

Prices and Cost Grouped by Date and Order Type

**prices in USD*

Order Type=Catalog Order Date=1/1/08

Quantity	Cost	Price
7	$9.25	$41.00
8	$28.45	$113.40
14	$12.10	$51.20
30	$59.00	$123.70
Number of observations for the month: 4		

Order Type=Catalog Order Date=1/2/08

Quantity	Cost	Price
11	$20.20	$40.20
100	$5.00	$11.80
Number of observations for the month: 2		

Order Type=In Store Order Date=1/1/08

Quantity	Cost	Price
25	$15.65	$31.10
Number of observations for the month: 1		

Order Type=In Store Order Date=1/2/08

Quantity	Cost	Price
2	$36.70	$146.40
20	$32.30	$71.00
Number of observations for the month: 2		

Order Type=Internet Order Date=1/1/08

Quantity	Cost	Price
2	$20.70	$92.60
Number of observations for the month: 1		

Example 4: Summing Numeric Variables with One BY Group

Procedure features:

PROC PRINT statement options:

N=
SUMLABEL

BY statement

SUM statement

Other features:

ODS CSVALL statement

SORT procedure

TITLE statement

#BYVAL specification

SAS system options:

BYLINE
NOBYLINE

Data set: "EXPREV" on page 1610

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
- creates a CSV file.

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 the NOBYLINE option, each BY group starts on a new page. The NODATE option suppresses the display of the date and time in the output. The PAGENO= option specifies the starting page number. The LINESIZE= option specifies the output line length, and the PAGESIZE= option specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=40 obs=10 nobyline;
```

Sort the data set. PROC SORT sorts the observations by Sale_Type.

```
proc sort data=exprev;
   by sale_type;
run;
```

Print the report, suppress the printing of observation numbers, and print the total number of observations for the selected variables. NOOBS suppresses the printing of observation numbers at the beginning of the rows. SUMLABEL prints the BY variable label on the summary line of each. N= prints the number of observations in a BY group at the end of that BY group and (because of the SUM statement) prints the number of observations in the data set at the end of the report. The first piece of explanatory text that N= provides precedes the number for each BY group. The second piece of explanatory text that N= provides precedes the number for the entire data set.

```
proc print data=exprev noobs label sumlabel
           n='Number of observations for the order type: '
          'Number of observations for the data set: ';
```

Select the variables to include in the report. The VAR statement creates columns for Country, Order_Date, Quantity, and Price, in that order.

```
   var country order_date quantity price;
```

Assign the variables' labels as column headings. The LABEL statement associates a label with each variable for the duration of the PROC PRINT step.

```
   label  sale_type='Sale Type'
          price='Total Retail Price* in USD'
          country='Country' order_date='Date' quantity='Quantity';
```

Sum the values for the selected variables. The SUM statement alone sums the values of Price and Quantity for the entire data set. Because the PROC PRINT step contains a BY statement, the SUM statement also sums the values of Price and Quantity for each sale type that contains more than one observation.

```
   sum price quantity;
   by sale_type;
```

Format the numeric values for a specified column. The FORMAT statement assigns the DOLLAR10.2. format to Price for this report.

```
   format price dollar7.2;
```

Specify and format a dynamic (or current) title. The TITLE statement specifies a title. The #BYVAL specification places the current value of the BY variable Sale_Type in the title. Because NOBYLINE is in effect, each BY group starts on a new page, and the title serves as a BY line.

```
   title 'Retail and Quantity Totals for #byval(sale_type) Sales';
run;
```

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

```
options byline;
```

Output: Listing

Output 43.6

```
                Retail and Quantity Totals for Catalog Sales                1

                                                          Total
                                                         Retail
                                                         Price*
Country                     Date     Quantity           in USD

Puerto Rico                1/1/08         14            $51.20
Aruba                      1/1/08         30           $123.70
Bahamas                    1/1/08          8           $113.40
Bermuda                    1/1/08          7            $41.00
British Virgin Islands     1/2/08         11            $40.20
Canada                     1/2/08        100            $11.80
----------------------                --------         -------
Sale Type                                170           $381.30

      Number of observations for the order type: 6
```

```
               Retail and Quantity Totals for In Store Sales                2

                                                          Total
                                                         Retail
                                                         Price*
Country                     Date     Quantity           in USD

Virgin Islands (U.S.)      1/1/08         25            $31.10
Belize                     1/2/08          2           $146.40
Cayman Islands             1/2/08         20            $71.00
----------------------                --------         -------
Sale Type                                47           $248.50

     Number of observations for the order type: 3
```

```
               Retail and Quantity Totals for Internet Sales                3

                                                   Total
                                                  Retail
                                                  Price*
  Country       Date     Quantity                 in USD

 Antarctica    1/1/08          2                  $92.60
                            ========           =======
                               219             $722.40

Number of observations for the order type: 1
  Number of observations for the data set: 10
```

Program: Creating a CSV File

```
options nodate pageno=1 linesize=80 pagesize=40 obs=10 nobyline;
```

Produce CSV formatted output and specify the file to store it in. The ODS CSVALL statement opens the CSVALL destination and creates a file containing tabular output with titles, notes, and bylines. The FILE= argument specifies the external file that contains the CSV output.

```
ods csvall file='your_file.csv';


proc sort data=exprev;
   by sale_type;
run;


proc print data=exprev noobs label sumlabel
           n='Number of observations for the order type: '
          'Number of observations for the data set: ';


var country order_date quantity price;


   label  price='Total Retail Price* in USD'
          country='Country' order_date='Date' quantity='Quantity';


   sum price quantity;
   by sale_type;


   format price dollar7.2;


   title 'Retail and Quantity Totals for #byval(sale_type) Sales';
run;


options byline;
```

Close the CSVALL destination. The ODS CSVALL CLOSE statement closes the CSVALL destination.

```
ods csvall close;
```

Output: CSV File

Display 43.8 Summing Numeric Variables with One BY Group: CSV Output Viewed with a Microsoft Excel

	A	B	C	D
1	Retail and Quantity Totals for Catalog Sales			
2				
3	Country	Date	Quantity	Total Retail Price* in USD
4	Puerto Rico	1/1/08	14	$51.20
5	Aruba	1/1/08	30	$123.70
6	Bahamas	1/1/08	8	$113.40
7	Bermuda	1/1/08	7	$41.00
8	British Virgi	1/2/08	11	$40.20
9	Canada	1/2/08	100	$11.80
10	Sale Type		170	$381.30
11	Number of observations for the order type: 6			
12				
13	Retail and Quantity Totals for In Store Sales			
14				
15	Country	Date	Quantity	Total Retail Price* in USD
16	Virgin Islan	1/1/08	25	$31.10
17	Belize	1/2/08	2	$146.40
18	Cayman Isl	1/2/08	20	$71.00
19	Sale Type		47	$248.50
20	Number of observations for the order type: 3			
21				
22	Retail and Quantity Totals for Internet Sales			
23				
24	Country	Date	Quantity	Total Retail Price* in USD
25	Antarctica	1/1/08	2	$92.60
26			219	$722.40
27	Number of observations for the order type: 1			
28	Number of observations for the data set: 10			

Example 5: Summing Numeric Variables with Multiple BY Variables

Procedure features:

PROC PRINT statement options:

- N=
- NOOBS
- STYLE
- SUMLABEL

BY statement

SUM statement

Other features:

ODS HTML statement

LABEL statement
FORMAT statement
SORT procedure
TITLE statement

Data set: "EXPREV" on page 1610

This example demonstrates the following tasks:

- sums quantities and retail prices for the following items:
  - each order date
  - each sale type with more than one row in the report
  - all rows in the report
- shows the number of observations in each BY group and in the whole report
- displays the BY group label in place of the BY group variable name on the summary line
- creates a default HTML report
- creates a stylized HTML report

Program: Creating a Listing Report

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. The PAGENO= option specifies the starting page number. The LINESIZE= option specifies the output line length, and the PAGESIZE= option specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=40;
```

Sort the data set. PROC SORT sorts the observations by Sale_Type and Order_Date.

```
proc sort data=exprev;
   by sale_type order_date;
run;
```

Print the report, suppress the printing of observation numbers, print the total number of observations for the selected variables and use the BY variable labels in place of the BY variable names in the summary line. The N option prints the number of observations in a BY group at the end of that BY group and prints the total number of observations used in the report at the bottom of the report. NOOBS suppresses the printing of observation numbers at the beginning of the rows. The SUMLABEL option prints the BY variable labels in the summary line in place of the BY variables.

```
proc print data=exprev n noobs sumlabel;
```

Create a separate section of the report for each BY group, and sum the values for the selected variables. The BY statement produces a separate section of the report for each BY group. The SUM statement alone sums the values of Price and Quantity for the entire data set. Because the program contains a BY statement, the SUM statement also sums the values of Price and Quantity for each BY group that contains more than one observation.

```
    by sale_type order_date;
    sum price quantity;
```

Establish a label for selected variables, format the values of specified variables, and create a title. The LABEL statement associates a labels with the variables Sale_Type and Order_Date for the duration of the PROC PRINT step. The labels are used in the BY line at the beginning of each BY group and in the summary line in place of BY variables. The FORMAT statement assigns a format to the variables Price and Cost for this report. The TITLE statement specifies a title.

```
    label  sale_type='Sale Type'
           order_date='Sale Date';
    format price dollar10.2 cost dollar10.2;
    title 'Retail and Quantity Totals for Each Sale Date and Sale Type';
run;
```

Output: Listing

Output 43.7 Summing Numeric Variables with Multiple BY Variables: Listing Output

The report uses default column headings (variable names) because neither the SPLIT= nor the LABEL option is used. Nevertheless, the BY line at the top of each section of the report shows the BY variables' labels and their values. The BY variables' labels identifies the subtotals in the report summary line.

PROC PRINT sums Price and Quantity for each BY group that contains more than one observation. However, sums are shown only for the BY variables whose values change from one BY group to the next. For example, in the first BY group, where the sale type is **`Catalog Sale`** and the sale date is **`1/1/08`**, Quantity and Price are summed only for the sale date because the next BY group is for the same sale type.

```
          Retail and Quantity Totals for Each Sale Date and Sale Type          1

---------------------- Sale Type=Catalog Sale Date=1/1/08 ----------------------

                            Ship_
   Country         Emp_ID    Date     Quantity         Price          Cost

   Puerto Rico    99999999  1/5/08        14           $51.20        $12.10
   Aruba          99999999  1/4/08        30          $123.70        $59.00
   Bahamas        99999999  1/4/08         8          $113.40        $28.45
   Bermuda        99999999  1/4/08         7           $41.00         $9.25
   -----------                          --------    ----------
   Sale Date                              59          $329.30

                                    N = 4


---------------------- Sale Type=Catalog Sale Date=1/2/08 ----------------------

                                  Ship_
Country                    Emp_ID  Date    Quantity        Price         Cost

British Virgin Islands   99999999 1/5/08       11         $40.20       $20.20
Canada                   99999999 1/5/08      100         $11.80        $5.00
El Salvador              99999999 1/6/08       21        $266.40       $66.70
Brazil                   120127   1/2/08       12         $73.40       $18.45
French Guiana            120935   1/2/08       15         $96.40       $43.85
Grenada                  120931   1/2/08       19         $56.30       $25.05
Paraguay                 120603   1/2/08       17        $117.60       $58.90
Peru                     120845   1/2/08       12         $93.80       $41.75
----------------------                      --------   ----------
Sale Date                                      207        $755.90
Sale Type                                      266      $1,085.20

                                    N = 8
```

Retail and Quantity Totals for Each Sale Date and Sale Type 2

---------- Sale Type=In Store Sale Date=1/1/08 ----------

Country	Emp_ID	Ship_ Date	Quantity	Price	Cost
Virgin Islands (U.S.)	99999999	1/4/08	25	$31.10	$15.65

N = 1

---------- Sale Type=In Store Sale Date=1/2/08 ----------

Country	Emp_ID	Ship_ Date	Quantity	Price	Cost
Belize	120458	1/2/08	2	$146.40	$36.70
Cayman Islands	120454	1/2/08	20	$71.00	$32.30
Guatemala	120931	1/2/08	13	$144.40	$65.70
Jamaica	99999999	1/4/08	23	$169.80	$38.70
Mexico	120127	1/2/08	30	$211.80	$33.65
Montserrat	120127	1/2/08	19	$184.20	$36.90
Anguilla	99999999	1/6/08	15	$233.50	$22.25
Antigua/Barbuda	120458	1/2/08	31	$99.60	$45.35
Argentina	99999999	1/6/08	42	$408.80	$87.15
Barbados	99999999	1/6/08	26	$94.80	$42.60
Bolivia	120127	1/2/08	26	$66.00	$16.60
Chile	120447	1/2/08	20	$19.10	$8.75
Ecuador	121042	1/2/08	11	$100.90	$50.55
Falkland Islands	120932	1/2/08	15	$61.40	$30.80
Guyana	120455	1/2/08	25	$132.80	$30.25
Martinique	120841	1/3/08	16	$56.30	$31.05
Netherlands Antilles	99999999	1/6/08	31	$41.80	$19.45
Sale Date			365	$2,242.60	
Sale Type			390	$2,273.70	

N = 17

```
          Retail and Quantity Totals for Each Sale Date and Sale Type          3

---------------------- Sale Type=Internet Sale Date=1/1/08 ----------------------

                          Ship_
     Country       Emp_ID  Date     Quantity         Price          Cost

    Antarctica    99999999  1/7/08        2           $92.60        $20.70

                                  N = 1


---------------------- Sale Type=Internet Sale Date=1/2/08 ----------------------

                              Ship_
 Country                 Emp_ID  Date     Quantity        Price         Cost

 Costa Rica              99999999  1/6/08        31         $53.00       $26.60
 Cuba                    121044    1/2/08        12         $42.40       $19.35
 Dominican Republic      121040    1/2/08        13         $48.00       $23.95
 Haiti                   121059    1/2/08         5         $47.90       $23.45
 Honduras                120455    1/2/08        20         $66.40       $30.25
 Nicaragua               120932    1/2/08        16        $122.00       $28.75
 Panama                  99999999  1/6/08        20         $88.20       $38.40
 Saint Kitts/Nevis       99999999  1/6/08        20         $41.40       $18.00
 St. Helena              120360    1/2/08        19         $94.70       $47.45
 St. Pierre/Miquelon     120842    1/16/08       16        $103.80       $47.25
 Turks/Caicos Islands    120372    1/2/08        10         $57.70       $28.95
 United States           120372    1/2/08        20         $88.20       $38.40
 Colombia                121059    1/2/08        28        $361.40       $90.45
 Dominica                121043    1/2/08        35        $121.30       $57.80
 Guadeloupe              120445    1/2/08        21        $231.60       $48.70
 St. Lucia               120845    1/2/08        19         $64.30       $28.65
 --------------------                          --------   ----------
 Sale Date                                        305      $1,632.30

                                  N = 16
```

```
          Retail and Quantity Totals for Each Sale Date and Sale Type          4

---------------------- Sale Type=Internet Sale Date=1/3/08 ----------------------

                          Ship_
      Country     Emp_ID  Date     Quantity         Price          Cost

     Suriname     120538  1/3/08        22          $110.80        $29.35
     ---------                      --------     ----------
     Sale Type                         329        $1,835.70
                                    ========     ==========
                                       985        $5,194.60

                                N = 1
                            Total N = 48
```

Program: Creating an HTML Report

```
options nodate pageno=1 linesize=80 pagesize=40 obs=10;
```

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 the external file that contains the HTML output.

```
ods html file='your_file.html';
```

```
proc sort data=exprev;
   by sale_type order_date;
run;

proc print data=exprev n noobs sumlabel;
   by sale_type order_date;
   sum price quantity;

   label  sale_type='Sale Type' order_date='Sale Date';
   format price dollar10.2 cost dollar10.2;
   title 'Retail and Quantity Totals for Each Sale Date and Sale Type';
run;
```

Close the HTML destination. The ODS HTML CLOSE statement closes the HTML destination.

```
ods html close;
```

Output: HTML

The following three displays comprise the output that creates default HTML output.

Display 43.9 Summing Numeric Variables with Multiple BY Variables: Catalog Sales: Default HTML Output

'Retail and Quantity Totals for Each Sale Date and Sale Type'

'Sale Type'=Catalog 'Sale Date'=1/1/08

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Puerto Rico	99999999	1/5/08	14	$51.20	$12.10
Aruba	99999999	1/4/08	30	$123.70	$59.00
Bahamas	99999999	1/4/08	8	$113.40	$28.45
Bermuda	99999999	1/4/08	7	$41.00	$9.25
'Sale Date'			59	$329.30	
N = 4					

'Sale Type'=Catalog 'Sale Date'=1/2/08

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
British Virgin Islands	99999999	1/5/08	11	$40.20	$20.20
Canada	99999999	1/5/08	100	$11.80	$5.00
'Sale Date'			111	$52.00	
'Sale Type'			170	$381.30	
N = 2					

Display 43.10 Summing Numeric Variables with Multiple BY Variables: In Store Sales: Default HTML Output

'Sale Type'=In Store 'Sale Date'=1/1/08

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Virgin Islands (U.S.)	99999999	1/4/08	25	$31.10	$15.65

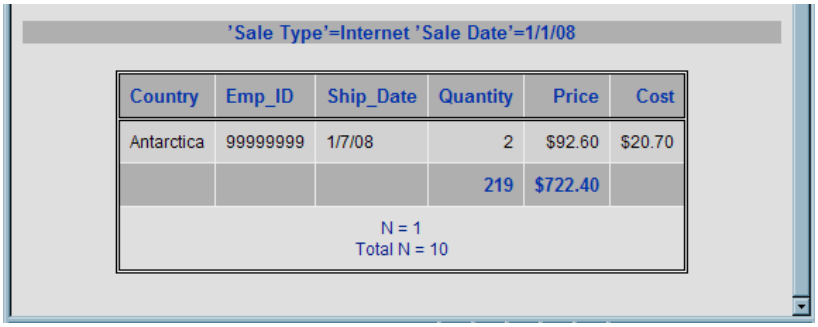
N = 1

'Sale Type'=In Store 'Sale Date'=1/2/08

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Belize	120458	1/2/08	2	$146.40	$36.70
Cayman Islands	120454	1/2/08	20	$71.00	$32.30
'Sale Date'			22	$217.40	
'Sale Type'			47	$248.50	

N = 2

Display 43.11 Summing Numeric Variables with Multiple BY Variables: Internet Sales: Default HTML Output



'Sale Type'=Internet 'Sale Date'=1/1/08

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Antarctica	99999999	1/7/08	2	$92.60	$20.70
			219	$722.40	

N = 1
Total N = 10

Program: Creating an HTML Report with the STYLE Option

```
options nodate pageno=1 linesize=80 pagesize=40 obs=10

ods html file='your_file.html';

proc sort data=exprev;
   by sale_type order_date;
run;

proc print data=exprev n noobs sumlabel;
```

Create stylized HTML output. The STYLE option in the first SUM statement specifies that the background color of the cell containing the grand total for the variable Price be changed to white and the font color be changed to blue. The STYLE option in the second SUM statement specifies that the background color of cells containing totals for the variable Quantity be changed to dark blue and the font color be changed to white.

```
   by sale_type order_date;
sum price / style(GRANDTOTAL) = [background =white color=blue];
sum quantity / style(TOTAL) = [background =dark blue color=white];

   label  sale_type='Sale Type' order_date='Sale Date';
   format price dollar10.2 cost dollar10.2;
```

```
   title 'Retail and Quantity Totals for Each Sale Date and Sale Type';
run;

ods html close;
```

Output: HTML with Styles

The following three displays comprise the output that creates styles in the HTML output.

Display 43.12 Summing Numeric Variables with Multiple BY Variables: Catalog Sales: HTML Output Using Styles

'Retail and Quantity Totals for Each Sale Date and Sale Type'

'Sale Type'=Catalog 'Sale Date'=1/1/08

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Puerto Rico	99999999	1/5/08	14	$51.20	$12.10
Aruba	99999999	1/4/08	30	$123.70	$59.00
Bahamas	99999999	1/4/08	8	$113.40	$28.45
Bermuda	99999999	1/4/08	7	$41.00	$9.25
'Sale Date'			59	$329.30	

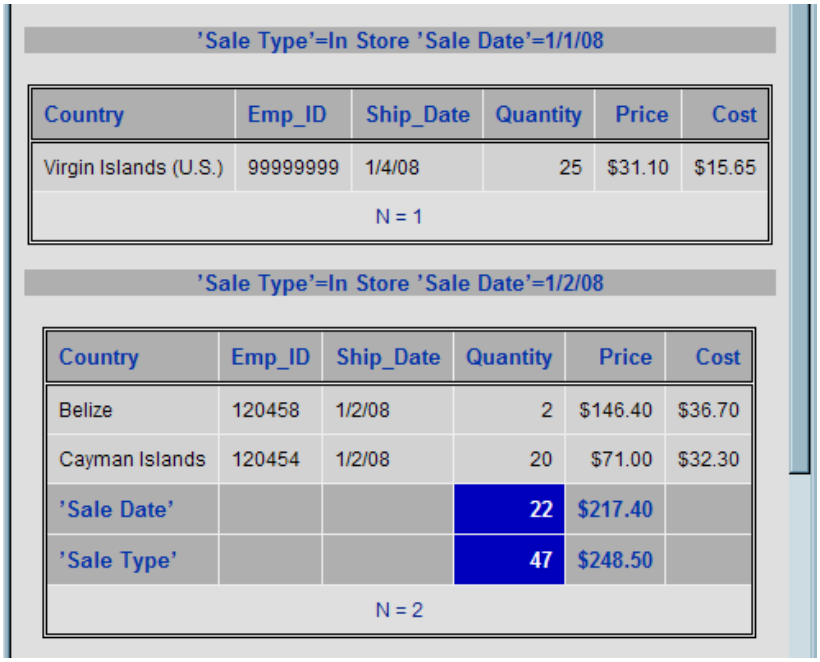
N = 4

'Sale Type'=Catalog 'Sale Date'=1/2/08

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
British Virgin Islands	99999999	1/5/08	11	$40.20	$20.20
Canada	99999999	1/5/08	100	$11.80	$5.00
'Sale Date'			111	$52.00	
'Sale Type'			170	$381.30	

N = 2

Display 43.13 Summing Numeric Variables with Multiple BY Variables: In Store Sales: HTML Output Using Styles



'Sale Type'=In Store 'Sale Date'=1/1/08

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Virgin Islands (U.S.)	99999999	1/4/08	25	$31.10	$15.65

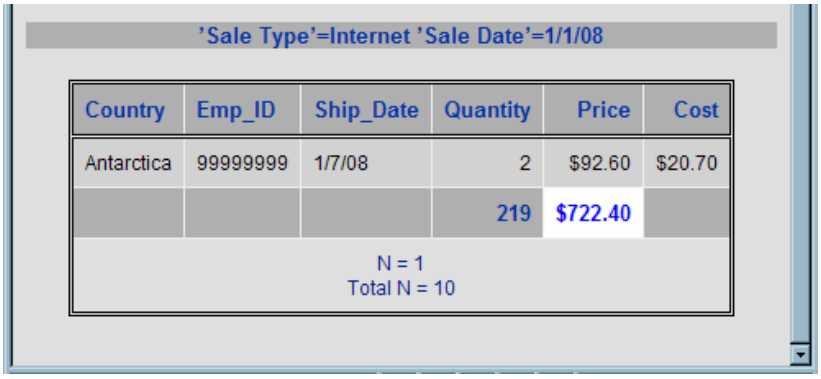
N = 1

'Sale Type'=In Store 'Sale Date'=1/2/08

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Belize	120458	1/2/08	2	$146.40	$36.70
Cayman Islands	120454	1/2/08	20	$71.00	$32.30
'Sale Date'			22	$217.40	
'Sale Type'			47	$248.50	

N = 2

Display 43.14 Summing Numeric Variables with Multiple BY Variables: Internet Sales: HTML Output Using Styles



'Sale Type'=Internet 'Sale Date'=1/1/08

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Antarctica	99999999	1/7/08	2	$92.60	$20.70
			219	$722.40	

N = 1
Total N = 10

Example 6: Limiting the Number of Sums in a Report

Features:

BY statement
SUM statement
SUMBY statement

Other features:

FORMAT statement
LABEL statement
ODS PDF statement
SORT procedure
TITLE statement

Data set: "EXPREV" on page 1610

This example

- creates a separate section of the report for each combination of sale type and sale date
- sums quantities and retail prices only for each sale type and for all sale types, not for individual dates.

Program: Creating a Listing Report

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. The PAGENO= option specifies the starting page number. The LINESIZE= option specifies the output line length, and the PAGESIZE= option specifies the number of lines on an output page. The OBS= option specifies the number of observations to be displayed.

```
options nodate pageno=1 linesize=80 pagesize=40 obs=10;
```

Sort the data set. PROC SORT sorts the observations by Sale_Type and Order_Date.

```
proc sort data=exprev;
   by sale_type order_date;
run;
```

Print the report and remove the observation numbers. NOOBS suppresses the printing of observation numbers at the beginning of the rows. The SUMLABEL uses the label for the BY variable on the summary line of each BY group.

```
proc print data=exprev noobs sumlabel;
```

Sum the values for each region. The SUM and BY statements work together to sum the values of Price and Quantity for each BY group as well as for the whole report. The SUMBY statement limits the subtotals to one for each type of sale.

```
   by sale_type order_date;
   sum price quantity;
   sumby sale_type;
```

Assign labels to specific variables. The LABEL statement associates a label with the variables Sale_Type and Order_Date for the duration of the PROC PRINT step. These labels are used in the BY group title or the summary line.

```
   label sale_type='Sale Type' order_date='Sale Date';
```

Assign a format to the necessary variables and specify a title. The FORMAT statement assigns the COMMA10. format to Cost and Price for this report. The TITLE statement specifies a title.

```
    format price dollar10.2 cost dollar10.2;
    title 'Retail and Quantity Totals for Each Sale Type';
run;
```

Output: Listing

Output 43.8 Limiting the Number of Sums in a Report: Listing Output

The report uses default column headings (variable names) because neither the SPLIT= nor the LABEL option is used. Nevertheless, the BY line at the top of each section of the report shows the BY variables' labels and their values. Because the SUMLABEL option is used, the BY variable label identifies the subtotals in the report.

```
                Retail and Quantity Totals for Each Sale Type                1

---------------------- Sale Type=Catalog Sale Date=1/1/08 ----------------------

                            Ship_
   Country         Emp_ID    Date     Quantity         Price          Cost

   Puerto Rico    99999999   1/5/08       14           $51.20        $12.10
   Aruba          99999999   1/4/08       30          $123.70        $59.00
   Bahamas        99999999   1/4/08        8          $113.40        $28.45
   Bermuda        99999999   1/4/08        7           $41.00         $9.25


---------------------- Sale Type=Catalog Sale Date=1/2/08 ----------------------

                                Ship_
Country                    Emp_ID    Date    Quantity        Price         Cost

British Virgin Islands   99999999  1/5/08       11          $40.20       $20.20
Canada                   99999999  1/5/08      100          $11.80        $5.00
----------------------                         --------   ----------
Sale Type                                         170        $381.30


--------------------- Sale Type=In Store Sale Date=1/1/08 ----------------------

                                Ship_
       Country             Emp_ID    Date    Quantity        Price         Cost

 Virgin Islands (U.S.)   99999999  1/4/08       25          $31.10       $15.65


--------------------- Sale Type=In Store Sale Date=1/2/08 ----------------------

                            Ship_
   Country           Emp_ID    Date     Quantity         Price          Cost

   Belize            120458   1/2/08        2           $146.40        $36.70
   Cayman Islands    120454   1/2/08       20            $71.00        $32.30
```

```
                Retail and Quantity Totals for Each Sale Type                2

---------------------- Sale Type=In Store Sale Date=1/2/08 ----------------------
                                (continued)

                          Ship_
      Country     Emp_ID    Date     Quantity         Price          Cost

      ---------                      --------    ----------
      Sale Type                         47          $248.50


---------------------- Sale Type=Internet Sale Date=1/1/08 ----------------------

                           Ship_
     Country       Emp_ID    Date     Quantity         Price          Cost

    Antarctica    99999999   1/7/08         2          $92.60        $20.70
                                       ========    ==========
                                          219          $722.40
```

Program: Creating a PDF file

```
options nodate pageno=1 linesize=80 pagesize=40 obs=10;
```

Produce PDF output and specify the file to store the output in. The ODS PDF statement opens the PDF destination and creates a file that contains PDF output. The FILE= argument specifies the external file that contains the PDF output.

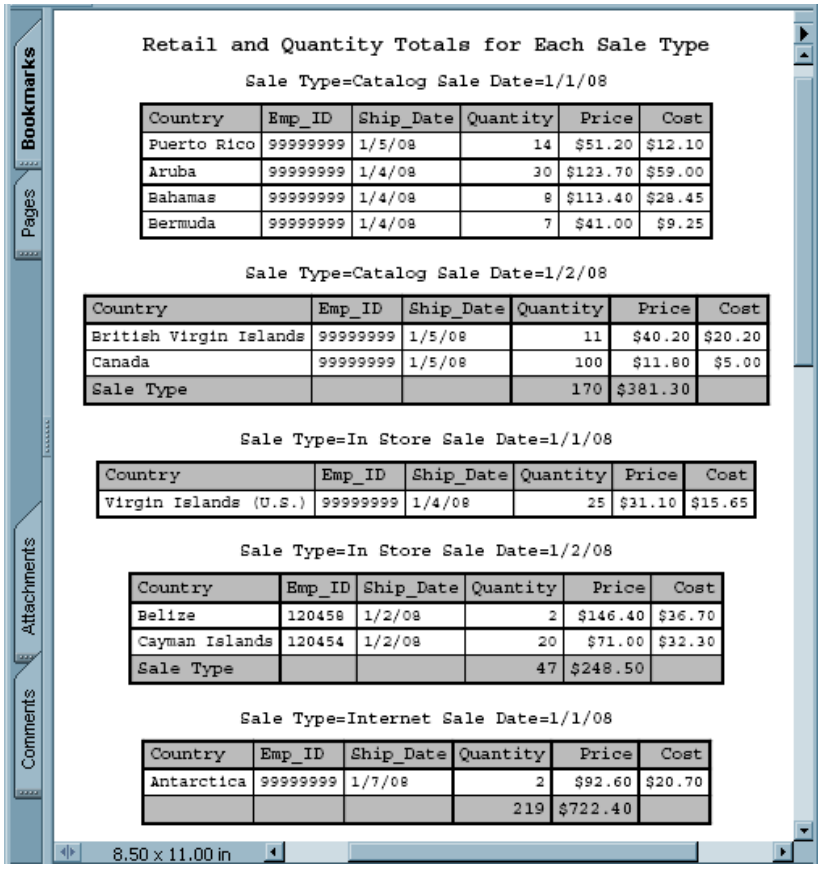
```
ods pdf file='your_file.pdf';


proc sort data=exprev;
   by sale_type order_date;
run;


proc print data=exprev noobs sumlabel;


by sale_type order_date;
   sum price quantity;
   sumby sale_type;


   label sale_type='Sale Type' order_date='Sale Date';


   format price dollar10.2 cost dollar10.2;
   title 'Retail and Quantity Totals for Each Sale Type';
run;
```

Close the PS destination. The ODS PDF CLOSE statement closes the PDF destination.

```
ods pdf close;
```

Output: PDF

Display 43.15 Limiting the Number of Sums in a Report: PDF Output



Retail and Quantity Totals for Each Sale Type

Sale Type=Catalog Sale Date=1/1/08

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Puerto Rico	99999999	1/5/08	14	$51.20	$12.10
Aruba	99999999	1/4/08	30	$123.70	$59.00
Bahamas	99999999	1/4/08	8	$113.40	$28.45
Bermuda	99999999	1/4/08	7	$41.00	$9.25

Sale Type=Catalog Sale Date=1/2/08

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
British Virgin Islands	99999999	1/5/08	11	$40.20	$20.20
Canada	99999999	1/5/08	100	$11.80	$5.00
Sale Type			170	$381.30	

Sale Type=In Store Sale Date=1/1/08

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Virgin Islands (U.S.)	99999999	1/4/08	25	$31.10	$15.65

Sale Type=In Store Sale Date=1/2/08

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Belize	120458	1/2/08	2	$146.40	$36.70
Cayman Islands	120454	1/2/08	20	$71.00	$32.30
Sale Type			47	$248.50	

Sale Type=Internet Sale Date=1/1/08

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Antarctica	99999999	1/7/08	2	$92.60	$20.70
			219	$722.40	

Program: Creating a PDF Report with the STYLE Option

```
options nodate pageno=1 linesize=80 pagesize=40 obs=10;

ods pdf file='your_file.pdf';

proc sort data=exprev;
   by sale_type order_date;
run;

proc print data=exprev noobs;
```

```
   by sale_type order_date;
```

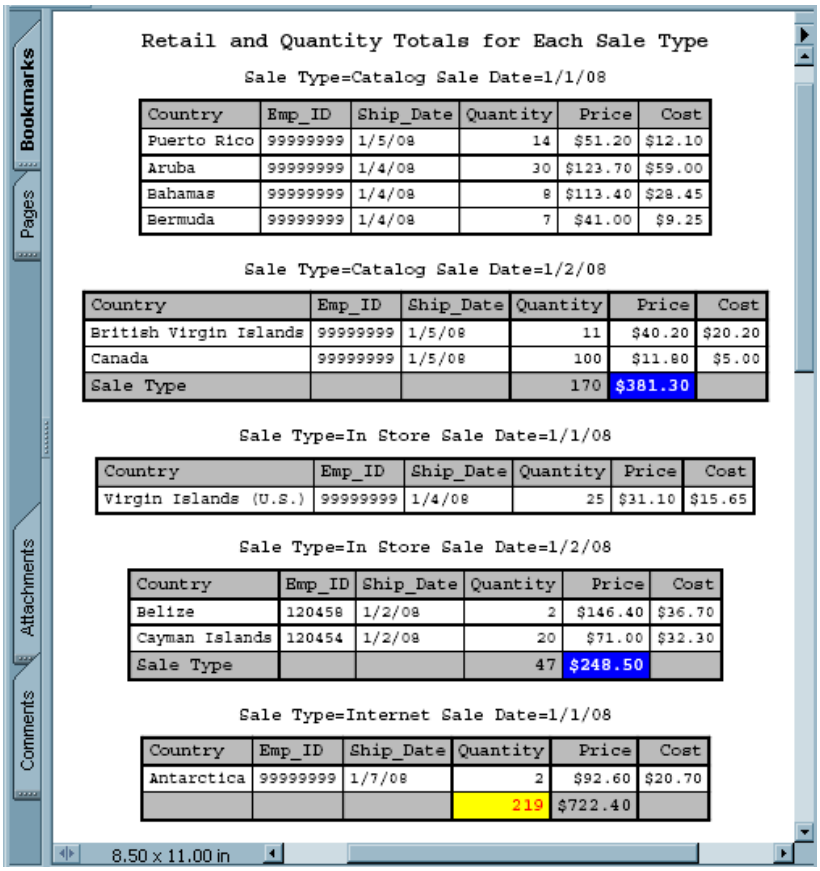
Create stylized PDF output. The STYLE option in the first SUM statement specifies that the background color of cells containing totals for the variable Price be changed to blue and the font color be changed to white.

The STYLE option in the second SUM statement specifies that the background color of the cell containing the grand total for the Quantity variable be changed to yellow and the font color be changed to red.

```
   sum price / style(TOTAL) = [background =blue color=white];
   sum quantity / style(GRANDTOTAL) = [background =yellow color=red];
   sumby sale_type;


   label sale_type='Sale Type' order_date='Sale Date';


   format price dollar10.2 cost dollar10.2;
   title 'Retail and Quantity Totals for Each Sale Type';
run;


ods pdf close;
```

Output: PDF with Styles

Display 43.16 Limiting the Number of Sums in a Report: PostScript Output Using Styles



Retail and Quantity Totals for Each Sale Type

Sale Type=Catalog Sale Date=1/1/08

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Puerto Rico	99999999	1/5/08	14	$51.20	$12.10
Aruba	99999999	1/4/08	30	$123.70	$59.00
Bahamas	99999999	1/4/08	8	$113.40	$28.45
Bermuda	99999999	1/4/08	7	$41.00	$9.25

Sale Type=Catalog Sale Date=1/2/08

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
British Virgin Islands	99999999	1/5/08	11	$40.20	$20.20
Canada	99999999	1/5/08	100	$11.80	$5.00
Sale Type			170	$381.30	

Sale Type=In Store Sale Date=1/1/08

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Virgin Islands (U.S.)	99999999	1/4/08	25	$31.10	$15.65

Sale Type=In Store Sale Date=1/2/08

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Belize	120458	1/2/08	2	$146.40	$36.70
Cayman Islands	120454	1/2/08	20	$71.00	$32.30
Sale Type			47	$248.50	

Sale Type=Internet Sale Date=1/1/08

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Antarctica	99999999	1/7/08	2	$92.60	$20.70
			219	$722.40	

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=

Data set: "EMPDATA" on page 1606

This example shows two ways of printing a data set with a large number of variables: one is the default, printing multiple rows when there are a large number of variables, and the other uses ROWS= option to print one row. For detailed explanations

of the layouts of these two reports, see the ROWS= option on page 822 and see "Page Layout" on page 832.

These reports use a page size of 24 and a line size 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

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 1606 creates this data set.

```
data empdata;
   input IdNumber $ 1-4 LastName $ 9-19 FirstName $ 20-29
         City $ 30-42 State $ 43-44 /
         Gender $ 1 JobCode $ 9-11 Salary 20-29 @30 Birth date9.
         @43 Hired date9. HomePhone $ 54-65;
   format birth hired date9.;
   datalines;
1919    Adams      Gerald    Stamford     CT
M       TA2        34376     15SEP1970      07JUN2005    203/781-1255
1653    Alexander  Susan     Bridgeport   CT
F       ME2        35108     18OCT1972      12AUG1998    203/675-7715

. . . more lines of data . . .

1407    Grant      Daniel    Mt. Vernon   NY
M       PT1        68096     26MAR1977      21MAR1998    914/468-1616
1114    Green      Janice    New York     NY
F       TA2        32928     21SEP1977      30JUN2006    212/588-1092
;
```

Print only the first 12 observations in a data set. The OBS= data set option uses only the first 12 observations to create the report. (This is just to conserve space here.) The ID statement identifies observations with the formatted value of IdNumber rather than with the observation number. This report is shown in Output 43.9

```
proc print data=empdata(obs=12);
   id idnumber;
   title 'Personnel Data';
run;
```

Print a report that contains only one row of variables on each page. ROWS=PAGE prints only one row of variables for each observation on a page. This report is shown in Output 43.10.

```
proc print data=empdata(obs=12) rows=page;
   id idnumber;
   title 'Personnel Data';
run;
```

Output: Listing

Output 43.9 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 headings at a change in capitalization if the entire name does not fit in the column. Compare, for example, the column headings for LastName (which fits in the column) and FirstName (which does not fit in the column).

```
                        Personnel Data                          1

  Id                 First                                    Job
Number  LastName     Name          City       State  Gender   Code

 1919   Adams        Gerald     Stamford       CT       M      TA2
 1653   Alexander    Susan      Bridgeport     CT       F      ME2
 1400   Apple        Troy       New York       NY       M      ME1
 1350   Arthur       Barbara    New York       NY       F      FA3
 1401   Avery        Jerry      Paterson       NJ       M      TA3
 1499   Barefoot     Joseph     Princeton      NJ       M      ME3
 1101   Baucom       Walter     New York       NY       M      SCP

  Id
Number  Salary          Birth          Hired       HomePhone

 1919     34376     15SEP1970     07JUN2005     203/781-1255
 1653     35108     18OCT1982     12AUG1998     203/675-7715
 1400     29769     08NOV1985     19OCT2006     212/586-0808
 1350     32886     03SEP1963     01AUG2000     718/383-1549
 1401     38822     16DEC1968     20NOV1993     201/732-8787
 1499     43025     29APR1962     10JUN1995     201/812-5665
 1101     18723     09JUN1980     04OCT1998     212/586-8060
```

```
                        Personnel Data                          2

  Id                 First                                    Job
Number  LastName     Name          City       State  Gender   Code

 1333   Blair        Justin     Stamford       CT       M      PT2
 1402   Blalock      Ralph      New York       NY       M      TA2
 1479   Bostic       Marie      New York       NY       F      TA3
 1403   Bowden       Earl       Bridgeport     CT       M      ME1
 1739   Boyce        Jonathan   New York       NY       M      PT1

  Id
Number  Salary          Birth          Hired       HomePhone

 1333     88606     02APR1979     13FEB2003     203/781-1777
 1402     32615     20JAN1971     05DEC1998     718/384-2849
 1479     38785     25DEC1966     08OCT2003     718/384-8816
 1403     28072     31JAN1979     24DEC1999     203/675-3434
 1739     66517     28DEC1982     30JAN2000     212/587-1247
```

Output 43.10 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                                    Job
Number  LastName     Name          City       State  Gender  Code

 1919   Adams        Gerald     Stamford       CT       M     TA2
 1653   Alexander    Susan      Bridgeport     CT       F     ME2
 1400   Apple        Troy       New York       NY       M     ME1
 1350   Arthur       Barbara    New York       NY       F     FA3
 1401   Avery        Jerry      Paterson       NJ       M     TA3
 1499   Barefoot     Joseph     Princeton      NJ       M     ME3
 1101   Baucom       Walter     New York       NY       M     SCP
 1333   Blair        Justin     Stamford       CT       M     PT2
 1402   Blalock      Ralph      New York       NY       M     TA2
 1479   Bostic       Marie      New York       NY       F     TA3
 1403   Bowden       Earl       Bridgeport     CT       M     ME1
 1739   Boyce        Jonathan   New York       NY       M     PT1
```

```
                        Personnel Data                          2

   Id
Number  Salary        Birth         Hired      HomePhone

 1919    34376    15SEP1970     07JUN2005     203/781-1255
 1653    35108    18OCT1982     12AUG1998     203/675-7715
 1400    29769    08NOV1985     19OCT2006     212/586-0808
 1350    32886    03SEP1963     01AUG2000     718/383-1549
 1401    38822    16DEC1968     20NOV1993     201/732-8787
 1499    43025    29APR1962     10JUN1995     201/812-5665
 1101    18723    09JUN1980     04OCT1998     212/586-8060
 1333    88606    02APR1979     13FEB2003     203/781-1777
 1402    32615    20JAN1971     05DEC1998     718/384-2849
 1479    38785    25DEC1966     08OCT2003     718/384-8816
 1403    28072    31JAN1979     24DEC1999     203/675-3434
 1739    66517    28DEC1982     30JAN2000     212/587-1247
```

Program: Creating an RTF Report

```
options nodate pageno=1 linesize=64 pagesize=24;
```

Create output for Microsoft Word and specify the file to store the output in. The ODS RTF statement opens the RTF destination and creates output formatted for Microsoft Word. The FILE= argument specifies the external file that contains the RTF output.

```
ods rtf file='your_file.rtf';
```

```
proc print data=empdata(obs=12);
   id idnumber;
   title 'Personnel Data';
run;
```

Close the RTF destination. The ODS RTF CLOSE statement closes the RTF destination.

```
ods rtf close;
```

Output: RTF

Display 43.17 Layout for a Report with Many Variables: RTF Output

Personnel Data

IdNumber	LastName	FirstName	City	State	Gender	JobCode	Salary	Birth	Hired	HomePhone
1919	Adams	Gerald	Stamford	CT	M	TA2	34376	15SEP70	07JUN05	203/781-1255
1653	Alexander	Susan	Bridgeport	CT	F	ME2	35108	18OCT72	12AUG98	203/675-7715
1400	Apple	Troy	New York	NY	M	ME1	29769	08NOV85	19OCT06	212/586-0808
1350	Arthur	Barbara	New York	NY	F	FA3	32886	03SEP63	01AUG00	718/383-1549
1401	Avery	Jerry	Paterson	NJ	M	TA3	38822	16DEC68	20NOV93	201/732-8787
1499	Barefoot	Joseph	Princeton	NJ	M	ME3	43025	29APR62	10JUN95	201/812-5665
1101	Baucom	Walter	New York	NY	M	SCP	18723	09JUN80	04OCT98	212/586-8060
1333	Blair	Justin	Stamford	CT	M	PT2	88606	02APR79	13FEB03	203/781-1777
1402	Blalock	Ralph	New York	NY	M	TA2	32615	20JAN71	05DEC98	718/384-2849
1479	Bostic	Marie	New York	NY	F	TA3	38785	25DEC66	08OCT03	718/384-8816
1403	Bowden	Earl	Bridgeport	CT	M	ME1	28072	31JAN79	24DEC99	203/675-3434
1739	Boyce	Jonathan	New York	NY	M	PT1	66517	28DEC82	30JAN00	212/587-1247

Program: Creating an RTF Report with the STYLE Option

```
options nodate pageno=1 linesize=64 pagesize=24;

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 43.18 Layout for a Report with Many Variables: RTF Output Using Styles

Personnel Data

IdNumber	LastName	FirstName	City	State	Gender	JobCode	Salary	Birth	Hired	HomePhone
1919	Adams	Gerald	Stamford	CT	M	TA2	34376	15SEP70	07JUN05	203/781-1255
1653	Alexander	Susan	Bridgeport	CT	F	ME2	35108	18OCT72	12AUG98	203/675-7715
1400	Apple	Troy	New York	NY	M	ME1	29769	08NOV85	19OCT06	212/586-0808
1350	Arthur	Barbara	New York	NY	F	FA3	32886	03SEP63	01AUG00	718/383-1549
1401	Avery	Jerry	Paterson	NJ	M	TA3	38822	16DEC68	20NOV93	201/732-8787
1499	Barefoot	Joseph	Princeton	NJ	M	ME3	43025	29APR62	10JUN95	201/812-5665
1101	Baucom	Walter	New York	NY	M	SCP	18723	09JUN80	04OCT98	212/586-8060
1333	Blair	Justin	Stamford	CT	M	PT2	88606	02APR79	13FEB03	203/781-1777
1402	Blalock	Ralph	New York	NY	M	TA2	32615	20JAN71	05DEC98	718/384-2849
1479	Bostic	Marie	New York	NY	F	TA3	38785	25DEC66	08OCT03	718/384-8816
1403	Bowden	Earl	Bridgeport	CT	M	ME1	28072	31JAN79	24DEC99	203/675-3434
1739	Boyce	Jonathan	New York	NY	M	PT1	66517	28DEC82	30JAN00	212/587-1247

Example 8: Creating a Customized Layout with BY Groups and ID Variables

Procedure features:

BY statement
ID statement
SUM statement
VAR statement

Other features:

SORT procedure

Data set: "EMPDATA" on page 1606

This customized report

- selects variables to include in the report and the order in which they appear
- selects observations to include in the report
- groups the selected observations by JobCode
- sums the salaries for each job code and for all job codes
- displays numeric data with commas and dollar signs.

Program: Creating a Listing Report

Create and sort a temporary data set. PROC SORT creates a temporary data set in which the observations are sorted by JobCode and Gender.

```
options nodate pageno=1 linesize=64 pagesize=60;
proc sort data=empdata out=tempemp;
```

```
   by jobcode gender;
run;
```

Identify the character that starts a new line in column headings. SPLIT= identifies the asterisk as the character that starts a new line in column headings.

```
proc print data=tempemp split='*';
```

Specify the variables to include in the report. The VAR statement and the ID statement together select the variables to include in the report. The ID statement and the BY statement produce the special format.

```
   id jobcode;
   by jobcode;
   var gender salary;
```

Calculate the total value for each BY group. The SUM statement totals the values of Salary for each BY group and for the whole report.

```
   sum salary;
```

Assign labels to the appropriate variables. The LABEL statement associates a label with each variable for the duration of the PROC PRINT step. When you use SPLIT= in the PROC PRINT statement, the procedure uses labels for column headings.

```
   label jobcode='Job Code*========'
         gender='Gender*======'
         salary='Annual Salary*=============';
```

Create formatted columns. The FORMAT statement assigns a format to Salary for this report. The WHERE statement selects for the report only the observations for job codes that contain the letters 'FA' or 'ME'. The TITLE statements specify two titles.

```
   format salary dollar11.2;
   where jobcode contains 'FA' or jobcode contains 'ME';
   title 'Salay Expenses';
run;
```

Output: Listing

Output 43.11 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.

```
                          Salay Expenses                                1

          Job Code    Gender    Annual Salary
          ========    ======    =============

            FA1         F         $23,177.00
                        F         $22,454.00
                        M         $22,268.00
          --------              -------------
            FA1                   $67,899.00

            FA2         F         $28,888.00
                        F         $27,787.00
                        M         $28,572.00
          --------              -------------
            FA2                   $85,247.00

            FA3         F         $32,886.00
                        F         $33,419.00
                        M         $32,217.00
          --------              -------------
            FA3                   $98,522.00

            ME1         M         $29,769.00
                        M         $28,072.00
                        M         $28,619.00
          --------              -------------
            ME1                   $86,460.00

            ME2         F         $35,108.00
                        F         $34,929.00
                        M         $35,345.00
                        M         $36,925.00
                        M         $35,090.00
                        M         $35,185.00
          --------              -------------
            ME2                  $212,582.00

            ME3         M         $43,025.00
                                 =============
                                  $593,735.00
```

Program: Creating an HTML Report

```
options nodate pageno=1 linesize=64 pagesize=60 obs=15;
proc sort data=empdata out=tempemp;
   by jobcode gender;
run;
```

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 the external file that contains the HTML output.

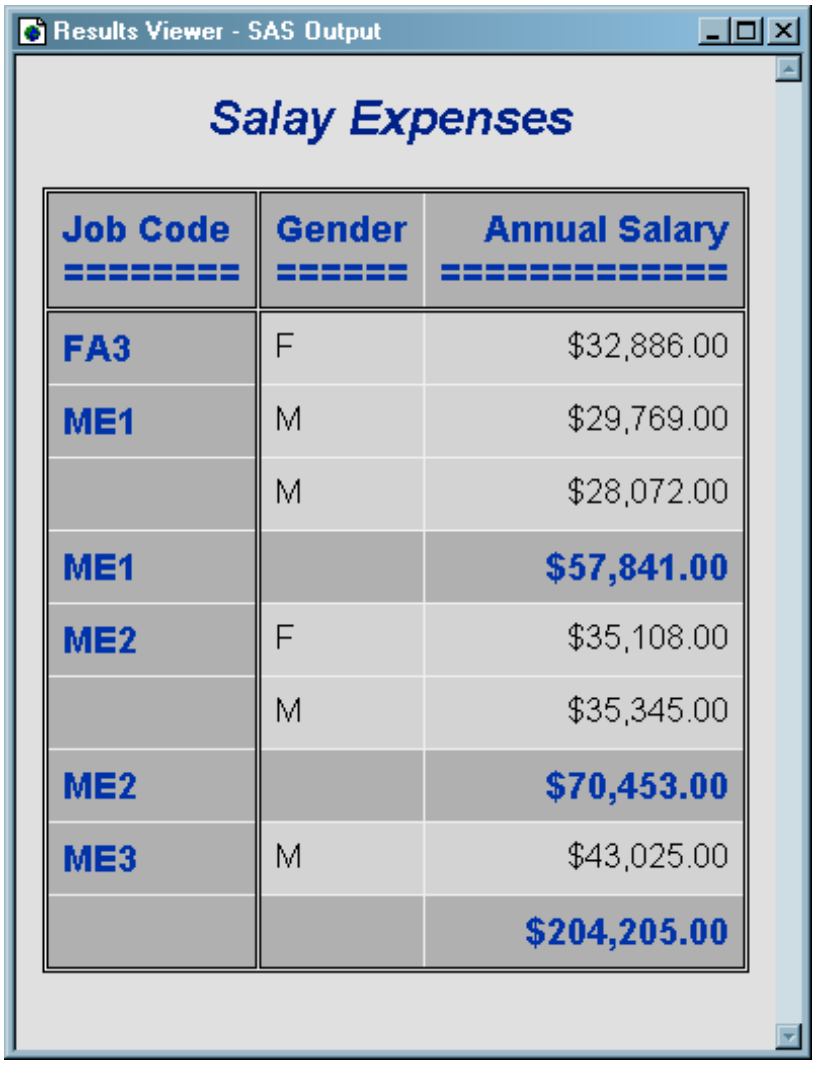
```
ods html file='your_file.html';


proc print data=tempemp (obs=10) split='*';


 id jobcode;
   by jobcode;
   var gender salary;


   sum salary;


   label jobcode='Job Code*========'
         gender='Gender*======'
         salary='Annual Salary*=============';


   format salary dollar11.2;
   where jobcode contains 'FA' or jobcode contains 'ME';
   title 'Salary Expenses';
run;
```

Close the HTML destination. The ODS HTML CLOSE statement closes the HTML destination.

```
ods html close;
```

Output: HTML

Display 43.19 Creating a Customized Layout with BY Groups and ID Variables: Default HTML Output



Program: Creating an HTML Report with the STYLE Option

```
options nodate pageno=1 linesize=64 pagesize=60 obs=15;
proc sort data=empdata out=tempemp;
   by jobcode gender;
run;


ods html file='your_file.html';
```

Create stylized HTML output. The first STYLE option specifies that the font of the headers be changed to italic. The second STYLE option specifies that the background of cells that contain input data be changed to blue and the foreground of these cells be changed to white.

```
proc print data=tempemp (obs=10) split='*' style(HEADER) =
                                  {fontstyle=italic}
                                  style(DATA) =
                                  {backgrouncolor=blue foreground = white};


 id jobcode;
   by jobcode;
   var gender salary;
```

Create total values that are written in red. The STYLE option specifies that the color of the foreground of the cell that contain the totals be changed to red.

```
   sum salary  / style(total)= [color=red];


   label jobcode='Job Code*========'
         gender='Gender*======'
         salary='Annual Salary*=============';


   format salary dollar11.2;
   where jobcode contains 'FA' or jobcode contains 'ME';
   title 'Expenses Incurred for';
   title2 'Salaries for Flight Attendants and Mechanics';
run;


ods html close;
```

Output: HTML with Styles

Display 43.20 Creating a Customized Layout with BY Groups and ID Variables: HTML Output Using Styles

Example 9: Printing All the Data Sets in a SAS Library

Features:
Macro facility
DATASETS procedure
PRINT procedure

Data set: "PROCLIB.DELAY" on page 1613 and "PROCLIB.INTERNAT" on page 1616

Program Overview

This example prints all the data sets in a SAS library. You can use the same programming logic with any procedure. Just replace the PROC PRINT step near the end of the example with whatever procedure step you want to execute. The example

uses the macro language. For details about the macro language, see *SAS Macro Language: Reference*.

Program

```
libname printlib 'SAS-data-library';
libname proclib '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=proclib memtype=data nolist;
   copy out=printlib;
      select delay internat;
run;
```

Create a macro and specify the parameters. The %MACRO statement creates the macro PRINTALL. When you call the macro, you can pass one or two parameters to it. The first parameter is the name of the library whose data set you want to print. The second parameter is a library used by the macro. If you do not specify this parameter, the WORK library is the default.

```
%macro printall(libname,worklib=work);
```

Create the local macro variables. The %LOCAL statement creates two local macro variables, NUM and I, to use in a loop.

```
   %local num i;
```

Produce an output data set. This PROC DATASETS step reads the library that you specify as a parameter when you invoke the macro. The CONTENTS statement produces an output data set called TEMP1 in WORKLIB. This data set contains an observation for each variable in each data set in the library LIBNAME. By default, each observation includes the name of the data set that the variable is included in as well as other information about the variable. However, the KEEP= data set option writes only the name of the data set to TEMP1.

```
   proc datasets library=&libname memtype=data nodetails;
      contents out=&worklib..temp1(keep=memname) data=_all_ noprint;
   run;
```

Specify the unique values in the data set, assign a macro variable to each one, and assign DATA step information to a macro variable. This DATA step increments the value of N each time it reads the last occurrence of a data set name (when IF LAST.MEMNAME is true). The CALL SYMPUT statement uses the current value of N to create a macro variable for each unique value of MEMNAME in the data set TEMP1. The TRIM function removes extra blanks in the TITLE statement in the PROC PRINT step that follows.

```
    data _null_;
       set &worklib..temp1 end=final;
       by memname notsorted;
       if last.memname;
       n+1;
       call symput('ds'||left(put(n,8.)),trim(memname));
```

When it reads the last observation in the data set (when FINAL is true), the DATA step assigns the value of N to the macro variable NUM. At this point in the program, the value of N is the number of observations in the data set.

```
       if final then call symput('num',put(n,8.));
```

Run the DATA step. The RUN statement is crucial. It forces the DATA step to run, thus creating the macro variables that are used in the CALL SYMPUT statements before the %DO loop, which uses them, executes.

```
    run;
```

Print the data sets and end the macro. The %DO loop issues a PROC PRINT step for each data set. The %MEND statement ends the macro.

```
    %do i=1 %to &num;
       proc print data=&libname..&&ds&i noobs;
          title "Data Set &libname..&&ds&i";
       run;
    %end;
%mend printall;
```

Print all the data sets in the PRINTLIB library. This invocation of the PRINTALL macro prints all the data sets in the library PRINTLIB.

```
options nodate pageno=1 linesize=70 pagesize=60;
%printall(printlib)
```

Output

Output 43.12 Printing All the Data Sets in a SAS Library: Listing Output

Data Set printlib.DELAY 1

flight	date	orig	dest	delaycat	destype	delay
114	01MAR08	LGA	LAX	1-10 Minutes	Domestic	8
202	01MAR08	LGA	ORD	No Delay	Domestic	-5
219	01MAR08	LGA	LON	11+ Minutes	International	18
622	01MAR08	LGA	FRA	No Delay	International	-5
132	01MAR08	LGA	YYZ	11+ Minutes	International	14
271	01MAR08	LGA	PAR	1-10 Minutes	International	5
302	01MAR08	LGA	WAS	No Delay	Domestic	-2
114	02MAR08	LGA	LAX	No Delay	Domestic	0
202	02MAR08	LGA	ORD	1-10 Minutes	Domestic	5
219	02MAR08	LGA	LON	11+ Minutes	International	18
622	02MAR08	LGA	FRA	No Delay	International	0
132	02MAR08	LGA	YYZ	1-10 Minutes	International	5
271	02MAR08	LGA	PAR	1-10 Minutes	International	4
302	02MAR08	LGA	WAS	No Delay	Domestic	0
114	03MAR08	LGA	LAX	No Delay	Domestic	-1
202	03MAR08	LGA	ORD	No Delay	Domestic	-1
219	03MAR08	LGA	LON	1-10 Minutes	International	4
622	03MAR08	LGA	FRA	No Delay	International	-2
132	03MAR08	LGA	YYZ	1-10 Minutes	International	6
271	03MAR08	LGA	PAR	1-10 Minutes	International	2
302	03MAR08	LGA	WAS	1-10 Minutes	Domestic	5
114	04MAR08	LGA	LAX	11+ Minutes	Domestic	15
202	04MAR08	LGA	ORD	No Delay	Domestic	-5
219	04MAR08	LGA	LON	1-10 Minutes	International	3
622	04MAR08	LGA	FRA	11+ Minutes	International	30
132	04MAR08	LGA	YYZ	No Delay	International	-5
271	04MAR08	LGA	PAR	1-10 Minutes	International	5
302	04MAR08	LGA	WAS	1-10 Minutes	Domestic	7
114	05MAR08	LGA	LAX	No Delay	Domestic	-2
202	05MAR08	LGA	ORD	1-10 Minutes	Domestic	2
219	05MAR08	LGA	LON	1-10 Minutes	International	3
622	05MAR08	LGA	FRA	No Delay	International	-6
132	05MAR08	LGA	YYZ	1-10 Minutes	International	3
271	05MAR08	LGA	PAR	1-10 Minutes	International	5
114	06MAR08	LGA	LAX	No Delay	Domestic	-1
202	06MAR08	LGA	ORD	No Delay	Domestic	-3
219	06MAR08	LGA	LON	11+ Minutes	International	27
132	06MAR08	LGA	YYZ	1-10 Minutes	International	7
302	06MAR08	LGA	WAS	1-10 Minutes	Domestic	1
114	07MAR08	LGA	LAX	No Delay	Domestic	-1
202	07MAR08	LGA	ORD	No Delay	Domestic	-2
219	07MAR08	LGA	LON	11+ Minutes	International	15
622	07MAR08	LGA	FRA	11+ Minutes	International	21
132	07MAR08	LGA	YYZ	No Delay	International	-2
271	07MAR08	LGA	PAR	1-10 Minutes	International	4
302	07MAR08	LGA	WAS	No Delay	Domestic	0

Data Set printlib.INTERNAT 2

flight	date	dest	boarded
219	01MAR08	LON	198
622	01MAR08	FRA	207
132	01MAR08	YYZ	115
271	01MAR08	PAR	138
219	02MAR08	LON	147
622	02MAR08	FRA	176
132	02MAR08	YYZ	106
271	02MAR08	PAR	172
219	03MAR08	LON	197
622	03MAR08	FRA	180
132	03MAR08	YYZ	75
271	03MAR08	PAR	147
219	04MAR08	LON	232
622	04MAR08	FRA	137
132	04MAR08	YYZ	117
271	04MAR08	PAR	146
219	05MAR08	LON	160
622	05MAR08	FRA	185
132	05MAR08	YYZ	157
271	05MAR08	PAR	177
219	06MAR08	LON	163
132	06MAR08	YYZ	150
219	07MAR08	LON	241
622	07MAR08	FRA	210
132	07MAR08	YYZ	164
271	07MAR08	PAR	155

CHAPTER

44

The PRINTTO Procedure

Overview: PRINTTO Procedure **887**
Syntax: PRINTTO Procedure **888**
PROC PRINTTO Statement **888**
Concepts: PRINTTO Procedure **891**
Page Numbering **891**
Routing SAS Log or Procedure Output Directly to a Printer **892**
Examples: PRINTTO Procedure **892**
Example 1: Routing to External Files **892**
Example 2: Routing to SAS Catalog Entries **895**
Example 3: Using Procedure Output as an Input File **898**
Example 4: Routing to a Printer **901**

Overview: PRINTTO Procedure

The PRINTTO procedure defines destinations, other than ODS destinations, for SAS procedure output and for the SAS log. By default, SAS procedure output and the SAS log are routed to the default procedure output file and the default SAS log file for your method of operation. The PRINTTO procedure does not define ODS destinations. See the following table.

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 44.1 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)*>;

Task	Statement
Defines destinations, other than ODS destinations, for SAS procedure output and for the SAS Log	"PROC PRINTTO Statement" on page 888

PROC PRINTTO Statement

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 901.

Restriction: The PRINTTO procedure does not define ODS destinations.

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.

PROC PRINTTO <*option(s)*>;

Task	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

When no options are specified, the PROC PRINTTO statement does the following:

- closes any files opened by a PROC PRINTTO statement
- points both the SAS log and SAS procedure output to their default destinations.

Interaction: To close the appropriate file and to return only the SAS log or procedure output to its default destination, use LOG=LOG or PRINT=PRINT.

Featured in: Example 1 on page 892 and Example 2 on page 895

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 895

LOG=LOG | *file-specification* | *SAS-catalog-entry*
routes the SAS log to one of three locations:

LOG
routes the SAS log to its default destination.

file-specification
routes the SAS log to an external file. *file-specification* can be one of the following:

'*external-file*'
the name of an external file specified in quotation marks.

Restriction: *external-file* cannot be longer than 1024 characters.

log-filename
is an unquoted alphanumeric text string. SAS creates a log that uses *log-filename*.log as the log filename.

Operating Environment Information: For more information about *log-filename*, see the documentation for your operating environment. △

fileref
a fileref previously assigned to an external file.

SAS-catalog-entry
routes the SAS log to a SAS catalog entry. By default, *libref* is SASUSER, *catalog* is PROFILE, and *type* is LOG. Express *SAS-catalog-entry* in one of the following ways:

libref.catalog.entry<.LOG>
a SAS catalog entry stored in the SAS library and SAS catalog specified.

catalog.entry<.LOG>
a SAS catalog entry stored in the specified SAS catalog in the default SAS library SASUSER.

entry.LOG
a SAS catalog entry stored in the default SAS library and catalog: SASUSER.PROFILE.

fileref

a fileref previously assigned to a SAS catalog entry. Search for "FILENAME, CATALOG Access Method" in the SAS online documentation.

Default: LOG.

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.

Interaction: When the log is routed to a file other than the default log file and programs are submitted from multiple sources, the final SAS system messages that contain the real and CPU times are written to the default SAS log.

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.)

Featured in: Example 1 on page 892, Example 2 on page 895, and Example 3 on page 898

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 892, Example 2 on page 895, and Example 3 on page 898

PRINT= PRINT | *file-specification* | *SAS-catalog-entry*
routes procedure output to one of three locations:

PRINT
routes procedure output to its default destination.

Tip: After routing it to an external file or a catalog entry, you can specify PRINT to route subsequent procedure output to its default destination.

file-specification
routes procedure output to an external file. *file-specification* can be one of the following:

'*external-file*'
the name of an external file specified in quotation marks.

Restriction: *external-file* cannot be longer than 1024 characters.

print-filename
is an unquoted alphanumeric text string. SAS creates a print file that uses *print-filename* as the print filename.

Operating Environment Information: For more information about using *print-filename*, see the documentation for your operating environment. △

fileref
a fileref previously assigned to an external file.

Operating Environment Information: For additional information about *file-specification* for the PRINT option, see the documentation for your operating environment. △

SAS-catalog-entry
routes procedure output to a SAS catalog entry. By default, *libref* is SASUSER, *catalog* is PROFILE, and *type* is OUTPUT. Express *SAS-catalog-entry* in one of the following ways:

libref.catalog.entry<.OUTPUT>
a SAS catalog entry stored in the SAS library and SAS catalog specified.

catalog.entry<.OUTPUT>
a SAS catalog entry stored in the specified SAS catalog in the default SAS library SASUSER.

entry.OUTPUT
a SAS catalog entry stored in the default SAS library and catalog: SASUSER.PROFILE.

fileref
a fileref previously assigned to a SAS catalog entry. Search for "FILENAME, CATALOG Access Method" in the SAS online documentation.

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 898

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 NUMBER SAS system option is in effect, there is a single page-numbering sequence for all output in the current job or session. When NONUMBER is in effect, output pages are not numbered.

- You can specify the beginning page number for the output 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 901.

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. △

The PRINTTO procedure does not support the COLORPRINTING system option. If you route the SAS log or procedure output to a color printer, the output does not print in color.

Examples: PRINTTO Procedure

Example 1: Routing to External Files

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 44.1 Portion of Log Routed to the Default Destination

```
01   options nodate pageno=1 linesize=80 pagesize=60 source;
02
03   proc printto log='log-file';
04   run;

NOTE: PROCEDURE PRINTTO used (Total process time):
      real time           0.01 seconds
      cpu time            0.00 seconds
```

Output 44.2 Portion of Log Routed to an External File

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


5
6      data numbers;
7       input x y z;
8       datalines;

NOTE: The data set WORK.NUMBERS has 6 observations and 3 variables.
NOTE: DATA statement used (Total process time):
      real time           0.06 seconds
      cpu time            0.04 seconds


15   ;
16
17     proc printto print='output-log' new;
18   run;

NOTE: PROCEDURE PRINTTO used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds


19
20     proc print data=numbers;
21      title 'Listing of NUMBERS Data Set';
22   run;

NOTE: There were 6 observations read from the data set WORK.NUMBERS.
NOTE: The PROCEDURE PRINT printed page 1.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.26 seconds
      cpu time            0.07 seconds


23
24     proc printto;
25   run;
```

Output

Output 44.3 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-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 2006 3070 20411  spring 2007
3070 20412  spring 2007 3070 20413  spring 2007
3070 20414  spring 2006 3070 20416  spring 2005
3071 20500  spring 2006 3071 20501  spring 2005
```

```
3071 20502  spring 2006 3071 20503  spring 2006
3071 20505  spring 2005 3071 20506  spring 2005
3071 20507  spring 2004 3071 20424  spring 2006
;
```

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 44.4 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.

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



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 44.5 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       2006
                       3070   20411   spring       2007
                       3070   20412   spring       2007
                       3070   20413   spring       2007
                       3070   20414   spring       2006
                       3070   20416   spring       2005
                       3071   20500   spring       2006
                       3071   20501   spring       2005
                       3071   20502   spring       2006
                       3071   20503   spring       2006
                       3071   20505   spring       2005
                       3071   20506   spring       2005
                       3071   20507   spring       2004
                       3071   20424   spring       2006
```

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 44.6.

```
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 44.7.

```
proc print data=probtest;
   title 'Chi-Square Analysis for Table of X by Y';
run;
```

Output 44.6 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 2.90 23.02 15.18	33 3.30 26.19 16.18	12 1.20 9.52 6.25	25 2.50 19.84 11.74	27 2.70 21.43 13.50	126 12.60
1	23 2.30 19.66 12.04	26 2.60 22.22 12.75	29 2.90 24.79 15.10	20 2.00 17.09 9.39	19 1.90 16.24 9.50	117 11.70
2	28 2.80 19.86 14.66	26 2.60 18.44 12.75	32 3.20 22.70 16.67	30 3.00 21.28 14.08	25 2.50 17.73 12.50	141 14.10
3	26 2.60 15.95 13.61	24 2.40 14.72 11.76	36 3.60 22.09 18.75	32 3.20 19.63 15.02	45 4.50 27.61 22.50	163 16.30
4	25 2.50 16.78 13.09	31 3.10 20.81 15.20	28 2.80 18.79 14.58	36 3.60 24.16 16.90	29 2.90 19.46 14.50	149 14.90
5	32 3.20 21.77 16.75	29 2.90 19.73 14.22	26 2.60 17.69 13.54	33 3.30 22.45 15.49	27 2.70 18.37 13.50	147 14.70
6	28 2.80 17.83 14.66	35 3.50 22.29 17.16	29 2.90 18.47 15.10	37 3.70 23.57 17.37	28 2.80 17.83 14.00	157 15.70
Total	191 19.10	204 20.40	192 19.20	213 21.30	200 20.00	1000 100.00

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 44.7 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.

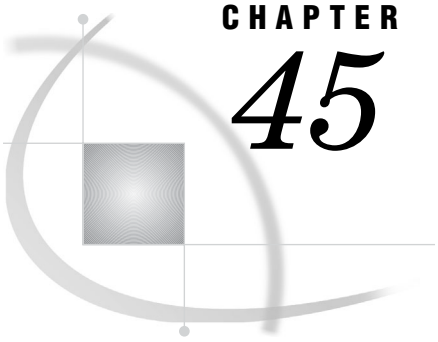
```
options nodate pageno=1 linesize=80 pagesize=60;
```

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

45

The PROTO Procedure

Overview: PROTO Procedure **903**
Syntax: PROTO Procedure **904**
PROC PROTO Statement **904**
LINK Statement **905**
MAPMISS Statement **906**
Concepts: PROTO Procedure **906**
Registering Function Prototypes **907**
Arguments **907**
Options **907**
Supported C Return Types **908**
Supported C Argument Types **908**
Basic C Language Types **909**
Working with Character Variables **909**
Working with Numeric Variables **909**
Working with Missing Values **910**
Interfacing with External C Functions **910**
C Structures in SAS **911**
Basic Concepts **911**
Declaring and Referencing Structures in SAS **912**
Structure Example **912**
Enumerations in SAS **913**
Enumerated Types Example **913**
C-Source Code in SAS **914**
Limitations for C Language Specifications **915**
C Helper Functions and CALL Routines **916**
What Are C Helper Functions and CALL Routines? **916**
ISNULL C Helper Function **916**
SETNULL C Helper CALL Routine **917**
STRUCTINDEX C Helper CALL Routine **917**
Results: PROTO Procedure **918**
Examples: PROTO Procedure **919**
Example 1: Splitter Function Example **919**

Overview: PROTO Procedure

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

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

Operating Environment Information: PROC PROTO is not available in the z/OS operating environment. △

Syntax: PROTO Procedure

PROC PROTO PACKAGE=*catalog-entry* <*options*>;

MAPMISS *type1=value1 type2=value2 ...*;

LINK *load-module* <NOUNLOAD>;

Task	Statement
Register, in batch mode, external functions that are written in the C or C++ programming languages.	"PROC PROTO Statement" on page 904
Specify the name, path, and load module that contains your functions.	"LINK Statement" on page 905
Specify alternative values, by type, to pass to functions if values are missing.	"MAPMISS Statement" on page 906

PROC PROTO Statement

PROC PROTO PACKAGE=*catalog-entry* <*options*>;

Task	Option
For XML databases only, enable the code to be encoded within a data set.	ENCRYPT | HIDE
Specify a text string to describe or label a package.	LABEL
Specify that none of the functions in a package will produce exceptions.	NOSIGNALS
For Windows PC platforms only, indicate that functions be called using the "_ stdcall" convention.	STDCALL
For Windows PC platforms only, specify that all structures in a package be compiled with a specific N-BYTE packing pragma.	STRUCTPACK*n* | PACK *n*

Arguments

PACKAGE=*catalog-entry*
specifies the SAS catalog entry where the function package information is saved. *Catalog-entry* is a three-level catalog name having one of the following forms: ***member.catalog.member*** or ***library.catalog.member***.

option
can be one of the following items:

ENCRYPT | HIDE
specifies that encoding within a database is allowed.

Restriction: This option is available for XML databases only.

LABEL=*package-label*
specifies a text string that is used to describe or label the package. The maximum length of the label is 256 characters.

NOSIGNALS
specifies that none of the functions in a package will produce exceptions or signals.

STDCALL
for Windows PC platforms only, indicates that all functions in the package are called using the "_stdcall" convention.

STRUCTPACK*n* | PACK*n*
for Windows PC platforms only, specifies that all structures in this package were compiled with the given N-BYTE packing pragma. That is, STRUCTURE4 specifies that all structures in the package were compiled with the “#pragma pack(4)” option.

LINK Statement

Specifies the name, and optionally the path, of the load module that contains your functions.

LINK *load-module*;

Arguments

load-module
specifies the load module that contains your functions. You can add more LINK statements to include as many libraries as you need for your prototypes. *Load-module* can have the following forms, depending on your operating environment:

```
'c:\mylibs\xxx.dll';
'c:\mylibs\xxx';
'/users/me/mylibs/xxx';
```

Tip: Full pathname specification is the safest and recommended way to link your modules with the PROTO procedure.

NOUNLOAD
specifies that selected libraries remain loaded when the SAS session ends.

Details

You do not need to specify your module's extension. SAS loads your module with the extension that is specific to your operating environment.

All functions must be declared externally in your load module so that SAS can find them. For most platforms, external declaration is the default behavior for the compiler. However, many C compilers do not export function names by default. The following examples show how to declare your functions for external loading for most PC compilers:

```
_declspec(dllexport) int myfunc(int, double);
_declspec(dllexport) int price2(int a, double foo);
```

MAPMISS Statement

Specifies alternative values, by type, to pass to functions if values are missing.

MAPMISS <POINTER=*pointer-value* INT=*integer-value* DOUBLE=*double-value*><
LONG=*long-value* SHORT=*short-value*>;

Arguments

POINTER=*pointer-value*
specifies the pointer value to pass to functions for pointer values that are missing. The default value is NULL.

INT=*integer-value*
specifies an integer value to pass to functions for integer values that are missing.

DOUBLE=*double-value*
specifies a double value to pass to functions for double values that are missing.

LONG=*long-value*
specifies a long value to pass to functions for long values that are missing.

SHORT=*short-value*
specifies a short value to pass to functions for short values that are missing.

Details

The MAPMISS statement is used to specify alternative values, by data type or pointer value. These values are passed to functions if values are missing. The values are specified as arguments on the MAPMISS statement.

If you set POINTER=NULL, a NULL value pointer is passed to the functions for pointer variables that are missing. If you do not specify a mapping for a type that is used as an argument to a function, the function is not called when an argument of that type is missing.

MAPMISS values have no affect on arrays because array elements are not checked for missing values when they are passed as parameters to C functions.

Concepts: PROTO Procedure

Registering Function Prototypes

Function prototypes are registered (declared) in the PROTO procedure. Use the following form:

return-type function-name (arg-type <*arg-name*> / <*iotype*> <*arg-label*>, *...)*
<*options*>;

Arguments

return-type
specifies a C language type for the returned value. See Table 45.1 on page 908 for a list of supported C return types.

Tip: *Return-type* can be preceded by either the *unsigned* or *Exceldate* modifiers. You need to use *Exceldate* if the return type is a Microsoft Excel date.

function-name
specifies the name of the function to be registered.

Tip: Function names within a given package must be unique in the first 32 characters. Function names do not need to be unique across different packages.

arg-type
specifies the C language type for the function argument. See Table 45.2 on page 908 for a list of supported C argument types.

You must specify *arg-type* for each argument in the function's argument list. The argument list must be between the left and closed parentheses. If the argument is an array, then you must specify the argument name prefixed to square brackets that contain the array size (for example, **`double A[10]`**). If the size is not known or if you want to disable verification of the length, then use **`type * name`** instead (for example, **`double * A`**).

Tip: *Arg-type* can be preceded by either the *unsigned*, *const*, or *Exceldate* modifiers. You need to use *Exceldate* if the return type is a Microsoft Excel date.

arg-name
specifies the name of the argument.

iotype
specifies the I/O type of the argument. Use I for input, O for output, or U for update.

Tip: By default, all parameters that are pointers are assumed to be input type U. All non-pointer values are assumed to be input type I. This behavior parallels the C language parameter passing scheme.

arg-label
specifies a description or label for the argument.

Options

LABEL="*text-string*"
specifies a description or a label for the function. Enclose the text string in quotation marks.

KIND | GROUP=*group-type*
specifies the group that the function belongs to. The KIND= or GROUP= option allows for convenient grouping of functions in a package.

You can use any string (up to 40 characters) in quotation marks to group similar functions.

Tip: The following special cases provided for Risk Dimensions do not require quotation marks: INPUT (Instrument Input), TRANS (Risk Factor Transformation), PRICING (Instrument Pricing), and PROJECT. The default is PRICING.

Supported C Return Types

The following C return types are supported in the PROTO procedure.

Table 45.1 Supported C Return Types

Function prototype	SAS variable type	C variable type
short	numeric	short, short *, short **
short *	numeric, array	short, short *, short **
int	numeric	int, int *, int **
int *	numeric, array	int, int *, int **
long	numeric	long, long *, long **
long *	numeric, array	long, long *, long **
double	numeric	double, double *, double **
double *	numeric, array	double, double *, double **
char *	character	char *, char **
struct *	struct	struct *, struct **
void		void

Supported C Argument Types

The following C argument types are supported in the PROTO procedure.

Table 45.2 Supported C Argument Types

Function prototype	SAS variable type	C variable type
short	numeric	short, short *, short **
short *	numeric, array	short, short *, short **
short **	array	short *, short **
int	numeric	int, int *, int **
int *	numeric, array	int, int *, int **
int **	array	int *, int **
long	numeric	long, long *, long **
long *	numeric, array	long, long *, long **
long **	array	long *, long **
double	numeric	double, double *, double **
double *	numeric, array	double, double *, double **
double **	array	double *, double **

Function prototype	SAS variable type	C variable type
char *	character	char *, char **
char **	character	char *, char **
struct *	structure	struct *, struct **
struct **	structure	struct *, struct **

Basic C Language Types

The SAS language supports two data types: character and numeric. These types correspond to an array of characters and a double (double-precision floating point) data type in the C programming language. When SAS variables are used as arguments to external C functions, they are converted (cast) into the proper types.

Working with Character Variables

You can use character variables for arguments that require a "char *" value only. The character string that is passed is a null string that is terminated at the current length of the string. The current length of the character string is the minimum of the allocated length of the string and the length of the last value that was stored in the string. The allocated length of the string (by default, 32 bytes) can be specified by using the LENGTH statement. Functions that return "char *" can return a null or zero-delimited string that is copied to the SAS variable. If the current length of the character string is less than the allocated length, the character string is padded with a blank.

In the following example, the allocated length of *str* is 10, but the current length is 5. When the string is NULL-terminated at the allocated length, "hello " is passed to the function xxx:

```
length str $ 10;
str = "hello";
call xxx(str);
```

To avoid the blank padding, use the SAS function TRIM on the parameter within the function call:

```
length str $ 10;
str = "hello";
call xxx(trim(str));
```

In this case, the value "hello" is passed to the function xxx.

Working with Numeric Variables

You can use numeric variables for an argument that requires a short, int, long, or double data type, as well as for pointers to those types. Numeric variables are converted to the required type automatically. If the conversion fails, then the function is not called and the output to the function is set to missing. If pointers to these types are requested, the address of the converted value is passed. On return from the call, the value is converted back to a double type and stored in the SAS variable. SAS scalar variables cannot be passed as arguments that require two or more levels of indirection. For example, a SAS variable cannot be passed as an argument that requires a cast to a "long **" type.

Working with Missing Values

SAS variables that contain missing values are converted according to how the function that is being called has mapped missing values when using the PROTO procedure. All variables that are returned from the function are checked for the mapped missing values and converted to SAS missing values.

For example, if an argument to a function is missing, and the argument is to be converted to an integer, and an integer was mapped to –99, then –99 is passed to the function. If the same function returns an integer with the value –99, then the variable that this value is returned to would have a value of missing.

Interfacing with External C Functions

To make it easier to interface with external C functions, many PROTO-compatible procedures have been enhanced to support most of these C types.

There is no way to return and save a pointer to any type in a SAS variable (see Table 45.3 on page 910. Pointers are always dereferenced, and their contents are converted and copied to SAS variables.

The EXTERNC statement is used to specify C variables in PROTO compatible procedures. The syntax of the EXTERNC statement has the following form:

EXTERNC DOUBLE | INT | LONG | SHORT | CHAR <[*][*]> *var-1* <*var-2* ... *var-n*>;

The following table shows how these variables are treated when they are positioned on the left side of an expression. The table shows the automatic casting that is performed for a short type on the right side of an assignment. (Explicit type conversions can be forced in any expression, with a unary operator called a cast.) The table lists all the allowed combinations of short types that are associated with SAS variables.

Note: A table for int, long, and double types can be created by substituting any of these types for "short" in this table. △

If any of the pointers are null and require dereferencing, then the result is set to missing if there is a missing value set for the result variable (see "MAPMISS Statement" on page 906 for more information).

Table 45.3 Automatic Type Casting for the *short* Data Type in an Assignment Statement

Type for left side of assignment	Type for right side of assignment	Cast performed
short	SAS numeric	y = (short) x
short	short	y = x
short	short *	y = * x
short	short **	y = ** x
short *	SAS numeric	* y = (short) x
short *	short	y = & x
short *	short *	y = x
short *	short **	y = * x
short **	SAS numeric	**y = (short) x

Type for left side of assignment	Type for right side of assignment	Cast performed
short **	short *	y = & x
short **	short **	y = x
SAS numeric	short	y = (double) x
SAS numeric	short *	y = (double) * x
SAS numeric	short **	y = (double) ** x

The following table shows how these variables are treated when they are passed as arguments to an external C function.

Table 45.4 Types That Are Allowed for External C Arguments

Function prototype	SAS variable type	C variable type
short	numeric	short, short *, short **
short *	numeric, array	short, short *, short **
short **	array	short *, short **
int	numeric	int, int *, int **
int *	numeric, array	int, int *, int **
int **	array	int *, int **
long	numeric	long, long *, long **
long *	numeric, array	long, long *, long **
long **	array	long *, long **
double	numeric	double, double *, double **
double *	numeric, array	double, double *, double **
double **	array	double *, double **
char *	character	char *, char **
char **	character	char *, char **
struct *	structure	struct *, struct **
struct **	structure	struct *, struct **

Note: Automatic conversion between two different C types is never performed. △

C Structures in SAS

Basic Concepts

Many C language libraries contain functions that have structure pointers as arguments. In SAS, structures can be defined only in PROC PROTO. After being defined, they can be declared and instantiated within many PROC PROTO compatible procedures, such as PROC COMPILE.

A C structure is a template that is applied to a contiguous piece of memory. Each entry in the template is given a name and a type. The type of each element determines the number of bytes that are associated with each entry and how each entry is to be used. Because of various alignment rules and base type sizes, SAS relies on the current machine compiler to determine the location of each entry in the memory of the structure.

Declaring and Referencing Structures in SAS

The syntax of a structure declaration in SAS is the same as for C non-pointer structure declarations. A structure declaration has the following form:

struct *structure_name structure_instance*;

Each structure is set to zero values at declaration time. The structure retains the value from the previous pass through the data to start the next pass.

Structure elements are referenced by using the static period (.) notation of C. There is no pointer syntax for SAS. If a structure points to another structure, the only way to reference the structure that is pointed to is by assigning the pointer to a declared structure of the same type. You use that declared structure to access the elements.

If a structure entry is a short, int, or long type, and it is referenced in an expression, it is first cast to a double type and then used in the calculations. If a structure entry is a pointer to a base type, then the pointer is dereferenced and the value is returned. If the pointer is NULL, then a missing value is returned. The missing value assignments that are made in the PROC PROTO code are used when conversions fail or when missing values are assigned to non-double structure entities.

The length of arrays must be known to SAS so that an array entry in a structure can be used in the same way as an array in SAS, as long as its dimension is declared in the structure. This requirement includes arrays of short, int, and long types. If the entry is actually a pointer to an array of a double type, then the array elements can be accessed by assigning that pointer to a SAS array. Pointers to arrays of other types cannot be accessed by using the array syntax.

Structure Example

```
proc proto package = sasuser.mylib.struct
   label = "package of structures";

   #define MAX_IN 20;

   typedef char * ptr;
   struct foo {
      double hi;
      int mid;
      ptr buf1;
      long * low;
      struct {
         short ans[MAX_IN + 1];
         struct {     /* inner */
            int inner;
         } n2;
         short outer;
      } n;
   };

   typedef struct foo *str;
```

```
   struct foo2 {
      str tom;
   };

   str get_record(char *name, int userid);
run;

proc fcmp library = sasuser.mylib;
   struct foo result;
   result = get_record("Mary", 32);
   put result=;
run;
```

Enumerations in SAS

Enumerations are mnemonics for integer numbers. Enumerations enable you to set a literal name as a specific number and aid in the readability and supportability of C programs. Enumerations are used in C language libraries to simplify the return codes. After a C program is compiled, you can no longer access enumeration names.

Enumerated Types Example

The following example shows how to set up two enumerated value types in PROC PROTO: YesNoMaybeType and Tens. Both are referenced in the structure EStructure:

```
proc proto package = sasuser.mylib.str2
   label = "package of structures";

   #define E_ROW 52;
   #define L_ROW 124;
   #define S_ROW 15;

   typedef double ExerciseArray[S_ROW][2];
   typedef double LadderArray[L_ROW];
   typedef double SamplingArray[S-Row];

   typedef enum
   {
      True, False, Maybe
   }  YesNoMaybeType;

   typedef enum {
      Ten = 10, Twnety = 20, Thirty = 30, Forty = 40, Fifty = 50
      } Tens;

   typedef struct {
      short           rows;
      short           cols;
      YesNoMaybeType  type;
      Tens            dollar;
      ExerciseArray   dates;
    } EStructure;
run;
```

The following PROC FCMP example shows how to access these enumerated types. In this example, the enumerated values that are set up in PROC PROTO are implemented in SAS as macro variables. Therefore, they must be accessed using the & symbol.

```
proc fcmp library = sasuser.mylib;
   EStructure mystruct;

   mystruct.type = &True;
   mystruct.dollar = &Twenty;
run;
```

C-Source Code in SAS

You can use PROC PROTO in a limited way to compile external C functions. The C source code can be specified in PROC PROTO in the following way:

```
EXTERNC function-name;
   ... C-source-statements ...
EXTERNCEND;
```

The function name tells PROC PROTO which function's source code is specified between the EXTERNC and EXTERNCEND statements. When PROC PROTO compiles source code, it includes any structure definitions and C function prototypes that are currently declared. However, typedef and #define are not included.

This functionality is provided to enable the creation of simple "helper" functions that facilitate the interface to preexisting external C libraries. Any valid C statement is permitted except for the #include statement. Only a limited subset of the C-stdlib functions are available. However, you can call any other C function that is already declared within the current PROC PROTO step.

The following C-stdlib functions are available:

Table 45.5 Supported stdlib Functions

Function	Description
double sin(double x)	returns the sine of x (radians)
double cos(double x)	returns the cosine of x (radians)
double tan(double x)	returns the tangent of x (radians)
double asin(double x)	returns the arcsine of x (-pi/2 to pi/2 radians)
double acos(double x)	returns the arccosine of x (0 to pi radians)
double atan(double x)	returns the arctangent of x (-pi/2 to pi/2 radians)
double atan2(double x, double y)	returns the arctangent of y/x (-pi to pi radians)
double sinh(double x)	returns the hyperbolic sine of x (radians)
double cosh(double x)	returns the hyperbolic cosine of x (radians)
double tanh(double x)	returns the hyperbolic tangent of x (radians)
double exp(double x)	returns the exponential value of x
double log(double x)	returns the logarithm of x
double log2(double x)	returns the logarithm of x base-2
double log10(double x)	returns the logarithm of x base-10

Function	Description
double pow(double x, double y)	returns x raised to the y power of x**y
double sqrt(double x)	returns the square root of x
double ceil(double x)	returns the smallest integer not less than x
double fmod(double x, double y)	returns the remainder of (x/y)
double floor(double x)	returns the largest integer not greater than x
int abs(int x)	returns the absolute value of x
double fabs(double)	returns the absolute value of x
int min(int x, int y)	returns the minimum of x and y
double fmin(double x, double y)	returns the minimum of x and y
int max(int x, int y)	returns the maximum of x and y
double fmax(double x, double y)	returns the maximum of x and y
char* malloc(int x)	allocates memory of size x
void free(char*)	frees memory allocated with malloc

The following example shows a simple C function written directly in PROC PROTO:

```
proc proto package=sasuser.mylib.foo;
   struct mystruct {
      short a;
      long b;
   };
   int fillMyStruct(short a, short b, struct mystruct * s);
   externc  fillMyStruct;
      int fillMyStruct(short a, short b, struct mystruct * s) {
         s ->a = a;
         s ->b = b;
         return(0);
      }
   externcend;
run;
```

Limitations for C Language Specifications

The limitations for the C language specifications in the PROTO procedure are as follows:

- #define statements must be followed by a semicolon (;) and must be numeric in value.
- The #define statement functionality is limited to simple replacement and unnested expressions. The only symbols that are affected are array dimension references.
- The C preprocessor statements #include and #if are not supported. The SAS macro %INC can be used in place of #include.
- A maximum of two levels of indirection are allowed for structure elements. Elements like "double ***" are not allowed. If these element types are needed in the structure, but are not accessed in SAS, you can use placeholders.
- The float type is not supported.
- A specified bit size or byte size for structure variables is not supported.
- Function pointers and definitions of function pointers are not supported.

- The union type is not supported. However, if you plan to use only one element of the union, you can declare the variable for the union as the type for that element.
- All non-pointer references to other structures must be defined before they are used.
- You cannot use the *enum* key word in a structure. In order to specify *enum* in a structure, use the *typedef* key word.
- Structure elements with the same alphanumeric name but with different cases (for example, ALPHA, Alpha, and alpha) are not supported. SAS is not case-sensitive. Therefore, all structure elements must be unique when compared in a case-insensitive program.

C Helper Functions and CALL Routines

What Are C Helper Functions and CALL Routines?

Several helper functions and CALL routines are provided with the package to handle C-language constructs in PROC FCMP. Most C-language constructs must be defined in a catalog package that is created by PROC PROTO before the constructs can be referenced or used by PROC FCMP. The ISNULL function and the STRUCTINDEX and SETNULL CALL routines have been added to extend the SAS language to handle C-language constructs that do not naturally fit into the SAS language.

The following C helper functions and CALL routines are available:

Table 45.6 C Helper Functions and CALL Routines

C helper function or CALL routine	Description
"ISNULL C Helper Function" on page 916	determines whether a pointer element of a structure is NULL.
"SETNULL C Helper CALL Routine" on page 917	sets a pointer element of a structure to NULL.
"STRUCTINDEX C Helper CALL Routine" on page 917	enables you to access each structure element in an array of structures.

ISNULL C Helper Function

The ISNULL function determines whether a pointer element of a structure is NULL. The function has the following form:

double **ISNULL** (*pointer-element*);

where *pointer-element* refers to the pointer element.

In the following example, the LINKLIST structure and the GET_LIST function are defined by using PROC PROTO. The GET_LIST function is an external C routine that generates a linked list with as many elements as requested.

```
struct linklist{
   double value;
   struct linklist * next;
};
```

```
struct linklist * get_list(int);
```

The following example shows how to use the ISNULL helper function to loop over the linked list that is created by the GET_LIST function.

```
struct linklist list;

list = get_list(3);
put list.value=;

do while (^isnull(list.next));
   list = list.next;
   put list.value=;
end;
```

The program writes the following results to the SAS log:

```
LIST.value=0
LIST.value=1
LIST.value=2
```

SETNULL C Helper CALL Routine

The SETNULL CALL routine sets a pointer element of a structure to NULL. It has the following form:

CALL SETNULL(*pointer-element*);

Pointer-element is a pointer to a structure.

When you specify a variable that has a pointer value (a structure entry), then SETNULL sets the pointer to null:

```
call setnull(12.next);
```

The following example assumes that the same LINKLIST structure that is described in "ISNULL C Helper Function" on page 916 is defined using PROC PROTO. The SETNULL CALL routine can be used to set the next element to null:

```
proc proto;
   struct linklist list;
   call setnull(list.next);
run;
```

STRUCTINDEX C Helper CALL Routine

The STRUCTINDEX CALL routine enables you to access each structure element in an array of structures. When a structure contains an array of structures, you can access each structure element of the array by using the STRUCTINDEX CALL routine. The STRUCTINDEX CALL routine has the following form:

CALL STRUCTINDEX(*struct_array*, *index*, *struct_element*);

Struct_array specifies an array; *index* is a 1–based index as used in SAS arrays; and *struct_element* points to an element in the arrays.

In the first part of this example, the following structures and function are defined using PROC PROTO:

```
struct point{
   short s;
   int i;
   long l;
   double d;
};

struct point_array {
   int            length;
   struct point * p;
   char           name[32];
};

struct point * struct_array(int);
```

In the second part of this example, the PROC FCMP code segment shows how to use the STRUCTUREINDEX CALL routine to get and set each POINT structure element of an array called P in the POINT_ARRAY structure:

```
struct point_array pntarray;
struct point pnt;

   /* Call struct_array to allocate an array of 2 POINT structures. */
pntarray.p = struct_array(2);
pntarray.plen = 2;
pntarray.name = "My funny structure";

   /* Get each element using the STRUCTINDEX CALL routine and set values. */
do i = 1 to 2;
   call structindex(pntarray.p, i, pnt);
   put "Before setting the" i "element: " pnt=;
   pnt.s = 1;
   pnt.i = 2;
   pnt.l = 3;
   pnt.d = 4.5;
   put "After setting the" i "element: " pnt=;
end;

run;
```

The program writes the following results to the SAS log.

Output 45.1 Output from the STRUCTINDEX CALL Routine

```
Before setting the 1 element: PNT {s=0, i=0, l=0, d=0}
After setting the 1 element: PNT {s=1, i=2, l=3, d=4.5}
Before setting the 2 element: PNT {s=0, i=0, l=0, d=0}
After setting the 2 element: PNT {s=1, i=2, l=3, d=4.5}
```

Results: PROTO Procedure

The PROTO procedure creates functions and subroutines that you can use with other SAS procedures.

Examples: PROTO Procedure

Example 1: Splitter Function Example

Procedure features:

INT statements

KIND= prototype argument

Other features:

PROC FCMP

This example shows how to use PROC PROTO to prototype two external C language functions called SPLIT and CASHFLOW. These functions are contained in the two shared libraries that are specified by the LINK statements.

Program

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGNO= specifies the starting page number. LINESIZE= specifies the output line length. PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=40;
```

Specify the catalog entry where the function package information is saved. The catalog entry is a three-level name.

```
proc proto package = sasusser.myfuncs.mathfun
           label = "package of math functions";
```

Specify the libraries that contain the SPLIT and CASHFLOW functions. You can add more LINK statements to include as many libraries as you need for your prototypes.

```
           link "link-library";
           link "link-library";
```

Prototype the SPLIT function. The INT statement prototypes the SPLIT function and assigns a label to the function.

```
    int split(int x "number to split")
        label = "splitter function" kind=PRICING;
```

Prototype the CASHFLOW function. The INT statement prototypes the CASHFLOW function and assigns a label to the function.

```
    int cashflow(double amt, double rate, int periods,
                 double * flows / iotype=0)
        label = "cash flow function" kind=PRICING;
```

Execute the PROTO procedure. The RUN statement executes the PROTO procedure.

```
run;
```

Call the SPLIT and CASHFLOW functions. PROC FCMP calls the SPLIT and CASHFLOW functions. Output from PROC FCMP is created.

```
proc fcmp libname=sasusser.myfuncs;
   array flows[20];
   a = split(32);
   put a;
   b = cashflow(1000, .07, 20, flows);
   put b;
   put flows;
run;
```

Output: Listing

Output 45.2 Output from the SPLIT and CASHFLOW Functions

```
                                    The SAS System                              1

                                   The FCMP Procedure

16
12
70 105 128.33333333 145.83333333 159.83333333 171.5 181.5 190.25 198.02777778 205.02777778
211.39141414 217.22474747 222.60936286 227.60936286 232.27602953 236.65102953 240.76867658
244.65756547 248.341776 251.841776
```

CHAPTER

46

The PRTDEF Procedure

Overview: PRTDEF Procedure **921**
Syntax: PRTDEF Procedure **921**
PROC PRTDEF Statement **922**
Input Data Set: PRTDEF Procedure **923**
Summary of Valid Variables **923**
Required Variables **924**
Optional Variables **925**
Examples: PRTDEF Procedure **928**
Example 1: Defining Multiple Printer Definitions **928**
Example 2: Creating a Ghostview Printer in SASUSER to Preview PostScript Printer Output in SASUSER **928**
Example 3: Creating a Single Printer Definition That Is Available to All Users **930**
Example 4: Adding, Modifying, and Deleting Printer Definitions **931**
Example 5: Deleting a Single Printer Definition **932**

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)*>;

Task	Statement
Creates printer definitions in batch mode either for an individual user or for all SAS users at your site.	Chapter 46, "The PRTDEF Procedure," on page 921

PROC PRTDEF Statement

PROC PRTDEF <*option(s)*>;

Task	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	USEASHELP

Options

DATA=*SAS-data-set*
specifies the SAS 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 that case only the NAME variable is required.

See: "Input Data Set: PRTDEF Procedure" on page 923

DELETE
specifies that the default operation is to delete the printer definitions from the registry.

Interaction: If both DELETE and REPLACE are specified, then DELETE is the default operation.

Tip: If the user-defined printer definition is deleted, then the administrator-defined printer can still appear if it exists in the SASHELP catalog.

FOREIGN
specifies that the registry entries are being created for export to a different host. As a consequence, tests of any host-dependent items, such as the TRANTAB, are skipped.

LIST
specifies that a list of printers that is created or replaced is written to the log.

REPLACE
specifies that the default operation is to modify existing printer definitions. Any printer name that already exists is modified by using the information in the printer attributes data set. Any printer name that does not exist is added.

Interaction: If both REPLACE and DELETE are specified, then a DELETE is performed.

USESASHELP
specifies that the printer definitions are to be placed in the SASHELP library, where they are available to all users. If the USESASHELP option is not specified, then the printer definitions are placed in the current SASUSER library, where they are available to the local user only.

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

Variable Name	Variable Description
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
PAPERTYP	Paper type
PREVIEW	Preview
PROTOCOL	Protocol
RES	Default printer resolution
RIGHT	Default right margin
STYLE	Default font style
TOP	Default top margin
TRANTAB	Translation table
TYPEFACE	Default font
UNITS	CM or IN units
VIEWER	Viewer
WEIGHT	Default font weight

Required Variables

To create or modify a printer, you must supply the NAME, MODEL, DEVICE, and DEST variables. All the other variables use default values from the printer prototype that is specified by the MODEL variable.

To delete a printer, specify only the required NAME variable.

The following variables are required in the input data set:

DEST specifies the output destination for the printer.

Operating Environment Information: DEST is case sensitive for some devices. △

Restriction: DEST is limited to 1023 characters.

DEVICE
specifies the type of I/O device to use when sending output to the printer. Valid devices are listed in the Printer Definition wizard and in the SAS Registry Editor.

Restriction: DEVICE is limited to 31 characters.

MODEL
specifies the printer prototype to use when defining the printer.

For a valid list of prototypes or model descriptions, you can look in the SAS Registry Editor under CORE\PRINTING\PROTOTYPES.

Restriction: MODEL is limited to 127 characters.

Tip: While in interactive mode, you can invoke the registry with the REGEDIT command.

Tip: While in interactive mode, you can invoke the Print Setup dialog box (DMPRTSETUP) and press New to view the list that is specified in the second window of the Printer Definition wizard.

NAME
specifies the printer definition name that is associated with the rest of the attributes in the printer definition.

The name is unique within a given registry. If a new printer definition contains a name that already exists, then the record is not processed unless the REPLACE option has been specified or unless the value of the OPCODE variable is **`Modify`**.

Restriction: NAME 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.

Default: DESC defaults to the prototype that is used to create the printer.

Restriction: The description can have a maximum of 1023 characters.

FONTSIZE
specifies the point size of the default font.

HOSTOPT
specifies any host options for the output destination. The host options are not case sensitive.

Restriction: The host options can have a maximum of 1023 characters.

LEFT
specifies the default left margin in the units that are specified by the UNITS variable.

LRECL
specifies the buffer size or record length to use when sending output to the printer.

Default: If LRECL is less than zero when modifying an existing printer, the printer's buffer size is reset to the size that is specified by the printer prototype.

OPCODE
is a character variable that specifies what action (Add, Delete, or Modify) to perform on the printer definition.

Add
creates a new printer definition in the registry. If the REPLACE option has been specified, then this operation will also modify an existing printer definition.

Delete
removes an existing printer definition from the registry.

Restriction: This operation requires only the NAME variable to be defined. The other variables are ignored.

Modify
changes an existing printer definition in the registry or adds a new one.

Restriction: OPTCODE is limited to eight characters.

Tip: If a user modifies and saves new attributes on a printer in the SASHELP library, then these modifications are stored in the SASUSER library. Values that are specified by the user will override values that are set by the administrator, but they will not replace them.

PAPERIN
specifies the default paper source or input tray.

Restriction: The value of PAPERIN must be one of the paper source names in the printer prototype that is specified by the MODEL variable.

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 eight characters.

TYPEFACE
specifies the typeface of the default font.

Restriction: The typeface must be one of the typeface names available to the printer prototype that is specified by the MODEL variable.

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.

Restriction: VIEWER is limited to 127 characters.

Tip: The values of the PREVIEW, PROTOCOL, DEST, and HOSTOPT variables are ignored when a value for VIEWER has been specified. Place **`%s`** where the input filename would normally be in the viewer command. The **`%s`** can be used as many times as needed.

WEIGHT
specifies the default font weight.

Restriction: The value must be one of the valid weights for the typeface that is specified by the TYPEFACE variable.

Examples: PRTDEF Procedure

Example 1: Defining Multiple Printer Definitions

Procedure features:
PROC PRTDEF statement options:
DATA=

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 and create the printer definitions. PROC PRTDEF creates the printer definitions for the SAS registry, and the DATA= option specifies PRINTERS as the input data set that contains the printer attributes.

```
proc prtdef data=printers;
run;
```

Example 2: Creating a Ghostview Printer in SASUSER to Preview PostScript Printer Output in SASUSER

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=
USESASHELP

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 is associated with the rest of the attributes in the printer definition data record.

The DESC variable specifies the description of the printer.

The MODEL variable specifies the printer prototype to use when defining this printer.

The DEVICE variable specifies the type of I/O device to use when sending output to the printer.

The DEST variable specifies the output destination for the printer.

The PREVIEW variable specifies which printer is used for print preview.

The UNITS variable specifies whether the margin variables are measured in centimeters or inches.

The BOTTOM variable specifies the default bottom margin in the units that are specified by the UNITS variable.

The FONTSIZE variable specifies the point size of the default font.

The PAPERSIZ variable specifies the default paper size.

```
data tek780;
   name = "Tek780";
   desc = "Test Lab Phaser 780P";
   model = "Tek Phaser 780 Plus";
   device = "PRINTER";
   dest = "testlab3";
   preview = "Ghostview";
   units = "cm";
   bottom = 2.5;
   fontsize = 14;
   papersiz = "ISO A4";
run;
```

Create the TEK780 printer definition and make the definition available to all users. The DATA= option specifies TEK780 as the input data set.

The USESASHELP option specifies that the printer definition will be available to all users.

```
proc prtdef data=tek780 usesashelp;
run;
```

Example 4: Adding, Modifying, and Deleting Printer Definitions

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 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

Chapter 47, "The PRTEXP Procedure," on page 933

CHAPTER
47

The PRTEXP Procedure

Overview: PRTEXP Procedure **933**
Syntax: PRTEXP Procedure **933**
PROC PRTEXP Statement **934**
EXCLUDE Statement **934**
SELECT Statement **935**
Concepts: PRTEXP Procedure **935**
Examples: PRTEXP Procedure **935**
Example 1: Writing Attributes to the SAS Log **935**
Example 2: Writing Attributes to a SAS Data Set **936**

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

Tip: If neither the SELECT nor the EXCLUDE statement is used, then all of the printers will be included in the output.

PROC PRTEXP<*option(s)*>;
<**SELECT** printer_1 …< printer_n>>;
<**EXCLUDE** printer_1 … <printer_n>>;

Task	Statement
Obtain printer attributes from the SAS registry	"PROC PRTEXP Statement" on page 934
Obtain printer attributes for the specified printers	"SELECT Statement" on page 935
Obtain printer attributes for all printers except for the specified printers	"EXCLUDE Statement" on page 934

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 causes the output to contain information from all printers that are not listed.

EXCLUDE printer_1 … <printer_n>;

Required Arguments

printer_1 printer_n
specifies the printers that you do not want the output to contain information about.

SELECT Statement

The SELECT statement causes the output to contain information from only the printers that are listed.

Featured in:

Example 1 on page 935
Example 2 on page 936

SELECT printer_1 … <printer_n>;

Required Arguments

printer_1 printer_n
specifies the printers 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 923.

Examples: PRTEXP Procedure

Example 1: Writing Attributes to the SAS Log

Procedure Features:

PROC PRTEXP statement option:
USESASHELP option
SELECT statement

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

Chapter 46, "The PRTDEF Procedure," on page 921

CHAPTER

48

The PWENCODE Procedure

Overview: PWENCODE Procedure **937**
Syntax: PWENCODE Procedure **937**
PROC PWENCODE Statement **937**
Concepts: PWENCODE Procedure **938**
Using Encoded Passwords in SAS Programs **938**
Encoding versus Encryption **939**
Examples: PWENCODE Procedure **939**
Example 1: Encoding a Password **939**
Example 2: Using an Encoded Password in a SAS Program **940**
Example 3: Saving an Encoded Password to the Paste Buffer **941**
Example 4: Specifying an Encoding Method for a Password **942**

Overview: PWENCODE Procedure

The PWENCODE procedure enables you to encode passwords. Encoded passwords can be used in place of plaintext passwords in SAS programs that access relational database management systems (RDBMSs) and various servers, such as SAS/CONNECT servers, SAS/SHARE servers, and SAS Integrated Object Model (IOM) servers (such as the SAS Metadata Server).

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. The password can contain up to a maximum of 512 characters, which include alphanumeric characters, spaces, and special characters. If the password contains embedded single or double quotation marks, 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 939, Example 2 on page 940, and Example 3 on page 941

Options

OUT=*fileref*
specifies a fileref to which the output string is to be written. If the OUT= option is not specified, the output string is written to the SAS log.

Featured in: Example 2 on page 940

METHOD=*encoding-method*
specifies the encoding method. Here are the supported values for *encoding-method*:

Table 48.1 Supported Encoding Methods

Encoding Method	Description	Supported Data Encryption Algorithm
sas001	Uses base64 to encode passwords.	None
sas002, which can also be specified as **sasenc**	Uses a 32-bit key to encode passwords. This is the default.	SASProprietary, which is included in SAS software.
sas003	Uses a 256-bit key to encode passwords.	AES (Advanced Encryption Standard), which is supported in SAS/SECURE*.

* SAS/SECURE is an add-on product that requires a separate license. For details about SAS/SECURE, the SASProprietary algorithm, and the AES algorithm, see *Encryption in SAS*.

If the METHOD= option is omitted, the default encoding method, **sas002**, is used automatically.

Concepts: PWENCODE Procedure

Using Encoded Passwords in SAS Programs

When a password is encoded with PROC PWENCODE, the output string includes a *tag* that identifies the string as having been encoded. An example of a tag is **{sas001}**.

The tag indicates the encoding method. SAS servers and SAS/ACCESS engines recognize the tag and decode the string before using it. Encoding a password enables you to write SAS programs without having to specify a password in plaintext.

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 48.1

```
6    proc pwencode in='my password';
7    run;

{sas002}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 illustrates the following:

- encoding a password and saving it to an external file
- reading the encoded password with a DATA step, storing it in a macro variable, and using it in a SAS/ACCESS LIBNAME statement

Program 1: Encoding the Password

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=len;
30      input @;
31      input @1 line $varying1024. len;
32      call symput('dbpass',substr(line,1,len));
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 {sas002}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 access 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;
```

Example 4: Specifying an Encoding Method for a Password

Procedure features: METHOD= argument

This example shows a simple case of encoding a password using the **sas003** encoding method and writing the encoded password to the SAS log.

Program

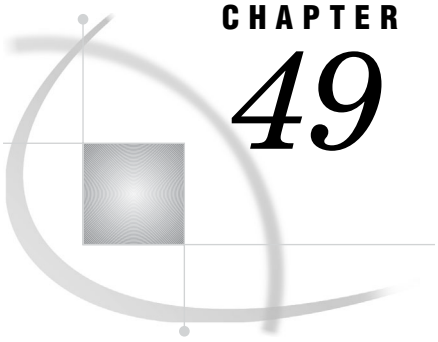
Encode the password.

```
proc pwencode in='my password' method=sas003;
run;
```

Log

Output 48.2

```
6    proc pwencode in='my password' encoding=sas003;
7    run;

{sas003}6EDB396015B96DBD9E80F0913A543819A8E5

NOTE: PROCEDURE PWENCODE used (Total process time):
      real time           0.14 seconds
      cpu time            0.09 seconds
```



CHAPTER

49

The RANK Procedure

Overview: RANK Procedure **943**
What Does the RANK Procedure Do? **943**
Ranking Data **943**
Syntax: RANK Procedure **945**
PROC RANK Statement **946**
BY Statement **949**
RANKS Statement **950**
VAR Statement **951**
Concepts: RANK Procedure **951**
Computer Resources **951**
Statistical Applications **951**
Treatment of Tied Values **952**
In-Database Processing for PROC RANK **953**
Results: RANK Procedure **954**
Missing Values **954**
Output Data Set **954**
Numeric Precision **954**
Examples: RANK Procedure **955**
Example 1: Ranking Values of Multiple Variables **955**
Example 2: Ranking Values within BY Groups **956**
Example 3: Partitioning Observations into Groups Based on Ranks **959**
References **961**

Overview: RANK Procedure

What Does the RANK Procedure Do?

The RANK procedure computes ranks for one or more numeric variables across the observations of a SAS data set and outputs the ranks to a new SAS data set. PROC RANK by itself produces no printed output.

Ranking Data

The following output shows the results of ranking the values of one variable with a simple PROC RANK step. In this example, the new ranking variable shows the order of finish of five golfers over a four-day competition. The player with the lowest number of strokes finishes in first place. The following statements produce the output:

```
proc rank data=golf out=rankings;
   var strokes;
   ranks Finish;
run;

proc print data=rankings;
run;
```

Output 49.1 Assignment of the Lowest Rank Value to the Lowest Variable Value

```
                          The SAS System                              1

                Obs    Player    Strokes    Finish

                 1     Jack        279         2
                 2     Jerry       283         3
                 3     Mike        274         1
                 4     Randy       296         4
                 5     Tito        302         5
```

In the following output, the candidates for city council are ranked by district according to the number of votes that they received in the election and according to the number of years that they have served in office.

This example shows how PROC RANK can do the following tasks:

- reverse the order of the rankings so that the highest value receives the rank of 1, the next highest value receives the rank of 2, and so on
- rank the observations separately by values of multiple variables
- rank the observations within BY groups
- handle tied values.

For an explanation of the program that produces this report, see Example 2 on page 956.

Output 49.2 Assignment of the Lowest Rank Value to the Highest Variable Value within Each BY Group

```
                        Results of City Council Election                        1

---------------------------------- District=1 ----------------------------------

                                                          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

Tip: You can use the ATTRIB, FORMAT, LABEL, and WHERE statements with the RANK procedure. See Chapter 3, "Statements with the Same Function in Multiple Procedures," on page 35 for details. You can also use any global statements. See "Global Statements" on page 20 for a list.

Tip: For in-database processing to occur, your data must reside within a supported version of a DBMS that has been properly configured for SAS in-database processing. For more information, see "In-Database Processing for PROC RANK" on page 953.

PROC RANK <*option(s)*>;
  BY <DESCENDING> *variable-1*
      <…<DESCENDING> *variable-n*>
      <NOTSORTED>;
  VAR *data-set-variables(s)*;
  RANKS *new-variables(s)*;

Task	Statement
Compute the ranks for one or more numeric variables in a SAS data set and output the ranks to a new SAS data set	"PROC RANK Statement" on page 946
Calculate a separate set of ranks for each BY group	"BY Statement" on page 949
Identify a variable that contains the ranks	"RANKS Statement" on page 950
Specify the variables to rank	"VAR Statement" on page 951

PROC RANK Statement

PROC RANK <*option(s)*>;

Task	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.

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.

Restriction: For in-database processing to occur, it is necessary that the data set specification refer to a table residing on a supported DBMS.

Main discussion: "Input Data Sets" on page 20

DESCENDING
reverses the direction of the ranks. With DESCENDING, the largest value receives a rank of 1, the next largest value receives a rank of 2, and so on. Otherwise, values are ranked from smallest to largest.

Featured in:
Example 1 on page 955
Example 2 on page 956

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 as follows:

$$\mathrm{FLOOR}\,(rank * k/(n+1))$$

FLOOR is the FLOOR function, *rank* is the value's order rank, *k* is the value of GROUPS=, and *n* is the number of observations having nonmissing values of the ranking variable for TIES=LOW, TIES=MEAN, and TIES=HIGH. For TIES=DENSE, *n* is the number of observations that have unique nonmissing values.

If the number of observations is evenly divisible by the number of groups, each group has the same number of observations, provided there are no tied values at the boundaries of the groups. Grouping observations by a variable that has many tied values can result in unbalanced groups because PROC RANK always assigns observations with the same value to the same group.

Tip: Use DESCENDING to reverse the order of the group values.

Featured in: Example 3 on page 959

NORMAL=BLOM | TUKEY | VW
computes normal scores from the ranks. The resulting variables appear normally distributed. *n* is the number of observations that have nonmissing values of the ranking variable for TIES=LOW, TIES=MEAN, and TIES=HIGH. For TIES=DENSE, *n* is the number of observations that have unique nonmissing values. The formulas are as follows:

BLOM	$y_i=\Phi^{-1}((r_i-3/8)/(n+1/4))$
TUKEY	$y_i=\Phi^{-1}((r_i-1/3)/(n+1/3))$
VW	$y_i=\Phi^{-1}((r_i)/(n+1))$

In these formulas, Φ^{-1} is the inverse cumulative normal (PROBIT) function, r_i is the rank of the 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 score.

Restriction: Use of the NORMAL= option will prevent in-database processing.

NPLUS1
computes fractional ranks by dividing each rank by the denominator n+1, where n is the number of observations that have nonmissing values of the ranking variable for TIES=LOW, TIES=MEAN, and TIES=HIGH. For TIES=DENSE, n is the number of observations that have unique nonmissing values.

Aliases: FN1, N1

Interaction: TIES=HIGH is the default with the NPLUS1 option.

See 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.

Interaction: When in-database processing is being performed and OUT= also refers to a supported DBMS table and if both IN= and OUT= reference the same library, then all processing can occur on the DBMS with results directly populating the output table. In this case, no results will be returned to SAS.

PERCENT
divides each rank by the number of observations that have nonmissing values of the variable and multiplies the result by 100 to get a percentage. n is the number of observations that have nonmissing values of the ranking variable for TIES=LOW, TIES=MEAN, and TIES=HIGH. For TIES=DENSE, n is the number of observations that have unique nonmissing values.

Alias: P

Interaction: TIES=HIGH is the default with the PERCENT option.

Tip: You can use PERCENT to calculate cumulative percentages, but you use GROUPS=100 to compute percentiles.

SAVAGE
computes Savage (or exponential) scores from the ranks by the following formula (Lehman 1998):

$$y_i = \left[\sum_{j=n-r_i+1} \left(\frac{1}{j} \right) \right] - 1$$

Interaction: If you specify the TIES= option, then PROC RANK computes the Savage score from the ranks based on non-tied values and applies the TIES= specification to the resulting score.

TIES=HIGH | LOW | MEAN | DENSE
specifies how to compute normal scores or ranks for tied data values.

HIGH

assigns the largest of the corresponding ranks (or largest of the normal scores when NORMAL= is specified).

LOW
assigns the smallest of the corresponding ranks (or smallest of the normal scores when NORMAL= is specified).

MEAN
assigns the mean of the corresponding rank (or mean of the normal scores when NORMAL= is specified).

DENSE
computes scores and ranks by treating tied values as a single-order statistic. For the default method, ranks are consecutive integers that begin with the number one and end with the number of unique, non-missing values of the variable that is being ranked. Tied values are assigned the same rank.

Note: CONDENSE is an alias for DENSE. △

Default: MEAN (unless the FRACTION option or PERCENT option is in effect).

Interaction: If you specify the NORMAL= option, then the TIES= specification applies to the normal score, not to the rank that is used to compute the normal score.

Featured in:

Example 1 on page 955

Example 2 on page 956

See also: "Treatment of Tied Values" on page 952

BY Statement

Produces a separate set of ranks for each BY group.

Main discussion: "BY" on page 36

Featured in:

Example 2 on page 956

Example 3 on page 959

Interaction: If the NOTSORTED option is specified on a BY statement, then in-database processing cannot be performed.

Interaction: Application of a format to any BY variable of the input data set, using a FORMAT statement for example, will prevent in-database processing.

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 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.

Default: If you omit the RANKS statement, the rank values replace the original variable values in the output data set.

Requirement: If you use the RANKS statement, you must also use the VAR statement.

Featured in:
Example 1 on page 955
Example 2 on page 956

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 on.

VAR Statement

Specifies the input variables.

Default: If you omit the VAR statement, PROC RANK computes ranks for all numeric variables in the input data set.

Featured in:

Example 1 on page 955
Example 2 on page 956
Example 3 on page 959

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 corresponds to the input variables specified in the VAR statement. If you omit the RANKS statement, the rank values replace the original values in the output data set.

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).

Treatment of Tied Values

When PROC RANK ranks values, if two or more values of an analysis variable that are within a BY group are equal, then tied values are present in the data. Because the values are indistinguishable and there is usually no further obvious information on which the ranks can reasonably be based, PROC RANK does not assign different ranks to the values. Tied values could be arbitrarily assigned different ranks. But in statistical applications such as nonparametric statistical tests employing ranks, it is conventional to assign the same rank to tied values.

These statistical tests commonly assume that the data is from a continuous distribution, in which the probability of a tie is theoretically zero. In practice, whether because of inaccuracies in measurement, the finite accuracy of representation within a digital computer, or other reasons, tied values often occur. It is also conventional in these statistical tests to assign the average rank to a group of tied values. Assignment of the average rank is preferred because it preserves the sum of the ranks and, therefore, does not distort the estimate of the cumulative distribution function.

For applications within and outside of statistics, the RANK procedure provides the TIES= option to control the treatment of tied values. The default value for this option depends on the specified ranking or scoring method, which you can specify with the options of the PROC RANK statement. For ranking and scoring methods, when TIES=LOW, TIES=HIGH, or TIES=MEAN, tied values are initially treated as though they are distinguishable. These methods all begin by sorting the values of the analysis variable within a BY group, and then assigning to each nonmissing value an ordinal number that indicates its position in the sequence.

Subsequently, for non-scoring methods, PROC RANK resolves tied values by selecting the minimum with TIES-LOW, selecting the maximum with TIES=HIGH, or calculating the average of the ordinals in a group of tied values with TIES=MEAN. PROC RANK then obtains the rank from this value through one or more further transformations such as scaling, translation, and truncation.

Scoring methods include normal and Savage scoring, which are requested by the NORMAL= and SAVAGE options. Non-scoring methods include ordinal ranking, the default, and those methods that are requested by the FRACTION, NPLUS1, GROUPS=, and PERCENT options. For the scoring methods NORMAL= and SAVAGE, PROC RANK obtains the probability quantile scores with the appropriate formulas as if no tied values were present within the data. PROC RANK then resolves tied values by selecting the minimum, selecting the maximum, or calculating the average of all scores within a tied group.

For all ranking and scoring methods, when TIES=DENSE, tied values are treated as indistinguishable, and each value within a tied group is assigned the same ordinal. As with the other TIES= resolution methods, all ranking and scoring methods begin by

sorting the values of the analysis variable and then assigning ordinals. However, a group of tied values is treated as a single value. The ordinal assigned to the group differs by only +1 from the ordinal that is assigned to the value just prior to the group, if there is one. The ordinal differs by only -1 from the ordinal assigned to the value just after the group, if there is one. Therefore, the smallest ordinal within a BY group is 1, and the largest ordinal is the number of unique, nonmissing values in the BY group.

After the ordinals are assigned, PROC RANK calculates ranks and scores using the number of unique, nonmissing values instead of the number of nonmissing values for scaling. Because of its tendency to distort the cumulative distribution function estimate, dense ranking is not generally acceptable for use in nonparametric statistical tests.

Note that PROC RANK bases its computations on the internal numeric values of the analysis variables. The procedure does not format or round these values before analysis. When values differ in their internal representation, even slightly, PROC RANK does not treat them as tied values. If this is a concern for your data, then round the analysis variables by an appropriate amount before invoking PROC RANK. For information about the ROUND function, see "Round Function" in *SAS Language Reference: Dictionary*.

In-Database Processing for PROC RANK

In-database processing has several advantages over processing within SAS. These advantages include increased security, reduced network traffic, and the potential for faster processing. Increased security is possible because sensitive data does not have to be extracted from the DBMS. Faster processing is possible because data is manipulated locally, on the DBMS, using high-speed secondary storage devices instead of being transported across a relatively slow network connection, because the DBMS might have more processing resources at its disposal, and because the DBMS might be capable of optimizing a query for execution in a highly parallel and scalable fashion.

In-database processing for PROC RANK supports DB2, Oracle, and Teradata database management systems.

The presence of table statistics might affect the performance of the RANK procedure's in-database processing. If your DBMS is not configured to automatically generate table statistics, then manual generation of table statistics might be necessary to achieve acceptable in-database performance.

Note:

- For DB2, generation of table statistics (either automatic or manual) is highly recommended for all but the smallest input tables.
- The TIES=CONDENSE option is not supported for the RANK procedure's in-database processing in an Oracle DBMS. If you use this option, it will prevent SQL generation and execution of in-database processing.

△

If the RANK procedure's input data set is a table or view that resides within a database from which rows would normally be retrieved with the SAS/ACCESS interface to Teradata, then PROC RANK can perform much or all of its work within the DBMS. There are several other factors that determine whether or not such in-database processing can occur. In-database processing will not occur in the following circumstances:

- if the RENAME= data set option is specified on the input data set.
- if a WHERE statement appears in the context of the RANK procedure or a WHERE= data set option is specified on the input data set, and the WHERE

statement or option contains a reference to a SAS function that has no equivalent in the DBMS or a format that has not been installed for use by SAS within the DBMS.

- if any variable specified on a BY statement has an associated format. Formatted BY variables are not supported by PROC RANK for in-database processing.
- if a FORMAT statement appears within the procedure context and applies to a variable specified on a BY statement, then in-database processing cannot be performed. Formatted BY variables are not supported by RANK for in-database processing. With a DBMS, formats can be associated with variables only if a FORMAT or ATTRIB statement appears within the procedure context.

For more information about the settings for system options, library options, data set options, and statement options that affect in-database performance for SAS procedures, see the SQLGENERATION= LIBNAME Option and the SQLGENERATION= option in *SAS/ACCESS for Relational Databases: Reference*.

When PROC RANK can process data within the DBMS, it generates an SQL query. The structure of the SQL query that is generated during an in-database invocation of PROC RANK depends on several factors, including the ranking methods that are used, the number of variables that are ranked, the inclusion of BY and WHERE statements, and the PROC RANK options that are used, such as TIES= and DESCENDING. The SQL query expresses the required calculations and is submitted to the DBMS. The results of this query will either remain as a new table within the DBMS if the output of the RANK procedure is directed there, or it will be returned to SAS. The settings for the MSGLEVEL option and the SQLGENERATION= option determine whether messages will be printed to the SAS log, which indicates whether in-database processing was performed. Generated SQL can be examined by setting the SASTRACE= option. For more information, see the SASTRACE= option in *SAS/ACCESS for Relational Databases: Reference*.

Results: RANK Procedure

Missing Values

Missing values are not ranked and are left missing when ranks or rank scores replace the original values in the output data set.

Output Data Set

The RANK procedure creates a SAS data set containing the ranks or rank scores but does not create any printed output. You can use PROC PRINT, PROC REPORT, or another SAS reporting tool to print the output data set.

The output data set contains all the variables from the input data set plus the variables named in the RANKS statement. If you omit the RANKS statement, the rank values replace the original variable values in the output data set.

Numeric Precision

For in-database processing, the mathematical operations expressed by the RANK procedure in SQL, and the order in which they are performed, are essentially the same

as those performed within SAS. However, in-database processing might result in small numerical differences when compared to results produced directly by SAS.

Examples: RANK Procedure

Example 1: Ranking Values of Multiple Variables

Procedure features:

PROC RANK statement options:

DESCENDING
TIES=

RANKS statement

VAR statement

Other features:

PRINT procedure

This example performs the following actions:

- reverses the order of the ranks so that the highest value receives the rank of 1
- assigns the best possible rank to tied values
- creates ranking variables and prints them with the original variables

Program

Set the SAS system options. The NODATE option specifies to omit the date and time when the SAS job begins. The PAGENO= option specifies the page number for the next page of output that SAS produces. The LINESIZE= option specifies the line size. The PAGESIZE= option specifies the number of lines for a page of SAS output.

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

Create the CAKE data set. This data set contains each participant's last name, score for presentation, and score for taste in a cake-baking contest.

```
data cake;
   input Name $ 1-10 Present 12-13 Taste 15-16;
   datalines;
Davis      77 84
Orlando    93 80
Ramey      68 72
Roe        68 75
Sanders    56 79
Simms      68 77
Strickland 82 79
```

```
;
```

Generate the ranks for the numeric variables in descending order and create the output data set ORDER. DESCENDING reverses the order of the ranks so that the high score receives the rank of 1. TIES=LOW gives tied values the best possible rank. OUT= creates the output data set ORDER.

```
proc rank data=cake out=order descending ties=low;
```

Create two new variables that contain ranks. The VAR statement specifies the variables to rank. The RANKS statement creates two new variables, PresentRank and TasteRank, that contain the ranks for the variables Present and Taste, respectively.

```
   var present taste;
   ranks PresentRank TasteRank;
run;
```

Print the data set. PROC PRINT prints the ORDER data set. The TITLE statement specifies a title.

```
proc print data=order;
   title "Rankings of Participants' Scores";
run;
```

Output: Listing

Rankings of Participants' Scores 1

Obs	Name	Present	Taste	Present Rank	Taste Rank
1	Davis	77	84	3	1
2	Orlando	93	80	1	2
3	Ramey	68	72	4	7
4	Roe	68	75	4	6
5	Sanders	56	79	7	3
6	Simms	68	77	4	5
7	Strickland	82	79	2	3

Example 2: Ranking Values within BY Groups

Procedure features:

PROC RANK statement options:

DESCENDING
TIES=

BY statement
RANKS statement
VAR statement

Other features:
PRINT procedure

This example performs the following actions:
- ranks observations separately within BY groups
- reverses the order of the ranks so that the highest value receives the rank of 1
- assigns the best possible rank to tied values
- creates ranking variables and prints them with the original variables

Program

Set the SAS system options. The NODATE option specifies to omit the date and time when the SAS job begins. The PAGENO= option specifies the page number for the next page of output that SAS produces. The LINESIZE= option specifies the line size. The PAGESIZE= option specifies the number of lines for a page of SAS output.

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

Create the ELECT data set. This data set contains each candidate's last name, district number, vote total, and number of years' experience on the city council.

```
data elect;
   input Candidate $ 1-11 District 13 Vote 15-18 Years 20;
   datalines;
Cardella    1 1689 8
Latham      1 1005 2
Smith       1 1406 0
Walker      1  846 0
Hinkley     2  912 0
Kreitemeyer 2 1198 0
Lundell     2 2447 6
Thrash      2  912 2
;
```

Generate the ranks for the numeric variables in descending order and create the output data set RESULTS. DESCENDING reverses the order of the ranks so that the highest vote total receives the rank of 1. TIES=LOW gives tied values the best possible rank. OUT= creates the output data set RESULTS.

```
proc rank data=elect out=results ties=low descending;
```

Create a separate set of ranks for each BY group. The BY statement separates the rankings by values of District.

```
   by district;
```

Create two new variables that contain ranks. The VAR statement specifies the variables to rank. The RANKS statement creates the new variables, VoteRank and YearsRank, that contain the ranks for the variables Vote and Years, respectively.

```
   var vote years;
   ranks VoteRank YearsRank;
run;
```

Print the data set. PROC PRINT prints the RESULTS data set. The N option prints the number of observations in each BY group. The TITLE statement specifies a title.

```
proc print data=results n;
   by district;
   title 'Results of City Council Election';
run;
```

Output: Listing

In the 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

Obs	Candidate	Vote	Years	Vote Rank	Years Rank
1	Cardella	1689	8	1	1
2	Latham	1005	2	3	2
3	Smith	1406	0	2	3
4	Walker	846	0	4	3

N = 4

District=2

Obs	Candidate	Vote	Years	Vote Rank	Years Rank
5	Hinkley	912	0	3	3
6	Kreitemeyer	1198	0	2	3
7	Lundell	2447	6	1	1
8	Thrash	912	2	3	2

N = 4

Example 3: Partitioning Observations into Groups Based on Ranks

Procedure features:

PROC RANK statement option:

GROUPS=

BY statement

VAR statement

Other features:

PRINT procedure

SORT procedure

This example performs the following actions:

- partitions observations into groups on the basis of values of two input variables
- groups observations separately within BY groups
- replaces the original variable values with the group values

Program

Set the SAS system options. The NODATE option specifies to omit the date and time when the SAS job began. The PAGENO= option specifies the page number for the next page of output that SAS produces. The LINESIZE= option specifies the line size. The PAGESIZE= option specifies the number of lines for a page of SAS output.

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

Create the SWIM data set. This data set contains swimmers' first names and their times, in seconds, for the backstroke and the freestyle. This example groups the swimmers into pairs, within male and female classes, based on times for both strokes so that every swimmer is paired with someone who has a similar time for each stroke.

```
data swim;
   input Name $ 1-7 Gender $ 9 Back 11-14 Free 16-19;
   datalines;
Andrea  F 28.6 30.3
Carole  F 32.9 24.0
Clayton M 27.0 21.9
Curtis  M 29.0 22.6
Doug    M 27.3 22.4
Ellen   F 27.8 27.0
Jan     F 31.3 31.2
Jimmy   M 26.3 22.5
Karin   F 34.6 26.2
Mick    M 29.0 25.4
Richard M 29.7 30.2
Sam     M 27.2 24.1
```

```
Susan   F 35.1 36.1
;
```

Sort the SWIM data set and create the output data set PAIRS. PROC SORT sorts the data set by Gender. This action is required to obtain a separate set of ranks for each group. OUT= creates the output data set PAIRS.

```
proc sort data=swim out=pairs;
   by gender;
run;
```

Generate the ranks that are partitioned into three groups and create an output data set. GROUPS=3 assigns one of three possible group values (0,1,2) to each swimmer for each stroke. OUT= creates the output data set RANKPAIR.

```
proc rank data=pairs out=rankpair groups=3;
```

Create a separate set of ranks for each BY group. The BY statement separates the rankings by Gender.

```
   by gender;
```

Replace the original values of the variables with the rank values. The VAR statement specifies that Back and Free are the variables to rank. With no RANKS statement, PROC RANK replaces the original variable values with the group values in the output data set.

```
   var back free;
run;
```

Print the data set. PROC PRINT prints the RANKPAIR data set. The N option prints the number of observations in each BY group. The TITLE statement specifies a title.

```
proc print data=rankpair n;
   by gender;
   title 'Pairings of Swimmers for Backstroke and Freestyle';
run;
```

Output: Listing

The group values pair swimmers with similar times to work on each stroke. For example, Andrea and Ellen work together on the backstroke because they have the fastest times in the female class. The groups of male swimmers are unbalanced because there are seven male swimmers; for each stroke, one group has three swimmers.

```
              Pairings of Swimmers for Backstroke and Freestyle              1

---------------------------------- Gender=F ----------------------------------

                          Obs    Name      Back    Free

                            1    Andrea       0       1
                            2    Carole       1       0
                            3    Ellen        0       1
                            4    Jan          1       2
                            5    Karin        2       0
                            6    Susan        2       2

                                 N = 6


---------------------------------- Gender=M ----------------------------------

                          Obs    Name        Back    Free

                            7    Clayton       0       0
                            8    Curtis        2       1
                            9    Doug          1       0
                           10    Jimmy         0       1
                           11    Mick          2       2
                           12    Richard       2       2
                           13    Sam           1       1

                                 N = 7
```

References

Blom, G. (1958), *Statistical Estimates and Transformed Beta Variables*, New York: 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.

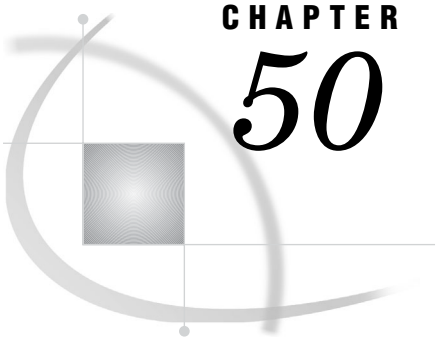
Conover, W.J. and Iman, R.L. (1981), "Rank Transformations as a Bridge between Parametric and Nonparametric Statistics," *The American Statistician*, 35, 124–129.

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

50

The REGISTRY Procedure

Overview: REGISTRY Procedure **963**
Syntax: REGISTRY Procedure **963**
PROC REGISTRY Statement **964**
Creating Registry Files with the REGISTRY Procedure **968**
Structure of a Registry File **968**
Specifying Key Names **968**
Specifying Values for Keys **969**
Sample Registry Entries **970**
Examples: REGISTRY Procedure **971**
Example 1: Importing a File to the Registry **971**
Example 2: Listing and Exporting the Registry **972**
Example 3: Comparing the Registry to an External File **973**
Example 4: Comparing Registry Files **974**
Example 5: Specifying an Entire Key Sequence with the STARTAT= Option **976**
Example 6: Displaying a List of Fonts **976**

Overview: REGISTRY Procedure

The REGISTRY procedure maintains the SAS registry. The registry consists of two parts. One part is stored in the SASHELP library, and the other part is stored in the SASUSER library.

The REGISTRY procedure enables you to do the following:

- Import registry files to populate the SASHELP and SASUSER registries.
- Export all or part of the registry to another file.
- List the contents of the registry in the SAS log.
- Compare the contents of the registry to a file.
- Uninstall a registry file.
- Deliver detailed status information when a key or value will be overwritten or uninstalled.
- Clear out entries in the SASUSER registry.
- Validate that the registry exists.
- List diagnostic information.

Syntax: REGISTRY Procedure

PROC REGISTRY <*option(s)*>;

Task	Statement
Manage registry files.	"PROC REGISTRY Statement" on page 964

PROC REGISTRY Statement

PROC REGISTRY <*option(s)*>;

Task	Option
Erase from the SASUSER registry the keys that were added by a user.	CLEARSASUSER on page 965
Compare two registry files.	COMPAREREG1= on page 965 and COMPAREREG2= on page 965
Compare the contents of a registry to a file.	COMPARETO= on page 965
Enable registry debugging.	DEBUGON on page 966
Disable registry debugging.	DEBUGOFF on page 966
Write the contents of a registry to the specified file.	EXPORT= on page 966
Provide additional information in the SAS log about the results of the IMPORT= and the UNINSTALL= options.	FULLSTATUS on page 966
Import the specified file to a registry	IMPORT= on page 966
Write the contents of the registry to the SAS log. This option is used with the STARTAT= option to list specific keys.	LIST on page 967
Write the contents of the SASHELP portion of the registry to the SAS log.	LISTHELP on page 967
Send the contents of a registry to the log.	LISTREG= on page 967
Write the contents of the SASUSER portion of the registry to the SAS log.	LISTUSER on page 967
Start exporting or writing or comparing the contents of a registry at the specified key.	STARTAT= on page 967
Delete from the specified registry all the keys and values that are in the specified file.	UNINSTALL= on page 967
Use uppercase for all incoming key names.	UPCASE on page 968

Task	Option
Use uppercase for all keys, names, and item values when you import a file.	UPCASEALL on page 968
Perform the specified operation on the SASHELP portion of the SAS registry.	USESASHELP on page 968

Options

CLEARSASUSER
erases from the SASUSER portion of the SAS registry the keys that were added by a user.

COMPAREREG1='*libname.registry-name-1*'
specifies one of two registries to compare. The results appear in the SAS log.

libname
is the name of the library in which the registry file resides.

registry-name-1
is the name of the first registry.

Requirement: Must be used with COMPAREREG2.

Interaction: To specify a single key and all of its subkeys, specify the STARTAT= option.

Featured in: Example 4 on page 974

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 974

COMPARETO=*file-specification*
compares the contents of a file that contains registry information to a registry. It returns information about keys and values that it finds in the file that are not in the registry. It reports the following items as differences:

- keys that are defined in the external file but not in the registry
- value names for a given key that are in the external file but not in the registry
- differences in the content of like-named values in like-named keys

COMPARETO= does not report as differences any keys and values that are in the registry but not in the file because the registry could easily be composed of pieces from many different files.

file-specification is one of the following:

'*external-file*'
is the path and name of an external file that contains the registry information.

fileref

is a fileref that has been assigned to an external file.

Requirement: You must have previously associated the fileref with an external file in a FILENAME statement, a FILENAME function, the Explorer window, or an appropriate operating environment command.

Interaction: By default, PROC REGISTRY compares *file-specification* to the SASUSER portion of the registry. To compare *file-specification* to the SASHELP portion of the registry, specify the option USESASHELP.

Featured in: Example 3 on page 973

See also: For information about how to structure a file that contains registry information, see "Creating Registry Files with the REGISTRY Procedure" on page 968.

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 972

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 971

See also: For information about how to structure a file that contains registry information, see "Creating Registry Files with the REGISTRY Procedure" on page 968.

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 972

STARTAT='*key-name*'
exports or writes the contents of a single key and all of its subkeys.
You must specify an entire key sequence if you want to start listing at any subkey under the root key.

Interaction: USE STARTAT= with the EXPORT=, LIST, LISTHELP, LISTUSER, COMPAREREG1=, COMPAREREG2= and the LISTREG options.

Featured in: Example 4 on page 974

UNINSTALL=*file-specification*
deletes from the specified registry all the keys and values that are in the specified file.
file-specification is one of the following:

'*external-file*'
is the name of an external file that contains the keys and values to delete.

fileref

is a fileref that has been assigned to an external file. To assign a fileref you can do the following:

- use the Explorer Window
- use the FILENAME statement (For information about the FILENAME statement, see the section on statements in *SAS 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 968.

UPCASE
uses uppercase for all incoming key names.

UPCASEALL
uses uppercase for all keys, names, and item values when you import a file.

USESASHELP
performs the specified operation on the SASHELP portion of the SAS registry.

Interaction: Use USESASHELP with the IMPORT=, EXPORT=, COMPARETO, or UNINSTALL option. To use USESASHELP with IMPORT= or UNINSTALL, you must have write permission to SASHELP.

Creating Registry Files with the REGISTRY Procedure

Structure of a Registry File

You can create registry files with the SAS Registry Editor or with any text editor.

A registry file must have a particular structure. Each entry in the registry file consists of a key name, followed on the next line by one or more values. The key name identifies the key or subkey that you are defining. Any values that follow specify the names or data to associate with the key.

Specifying Key Names

Key names are entered on a single line between square brackets ([and]). To specify a subkey, enter multiple key names between the brackets, starting with the root key. Separate the names in a sequence of key names with a backslash (\). The length of a single key name or a sequence of key names cannot exceed 255 characters (including the square brackets and the backslashes). Key names can contain any character except the backslash.

Examples of valid key name sequences follow. These sequences are typical of the SAS registry:

[CORE\EXPLORER\MENUS\ENTRIES\CLASS]

[CORE\EXPLORER\NEWMEMBER\CATALOG]
[CORE\EXPLORER\NEWENTRY\CLASS]
[CORE\EXPLORER\ICONS\ENTRIES\LOG]

Specifying Values for Keys

Enter each value on the line that follows the key name that it is associated with. You can specify multiple values for each key, but each value must be on a separate line.

The general form of a value is:

value-name=value-content

A *value-name* can be an at sign (@), which indicates the default value name, or it can be any text string in double quotation marks. If the text string contains an ampersand (&), then the character (either uppercase or lowercase) that follows the ampersand is a shortcut for the value name. See "Sample Registry Entries" on page 970.

The entire text string cannot contain more than 255 characters (including quotation marks and ampersands). It can contain any character except a backslash (\).

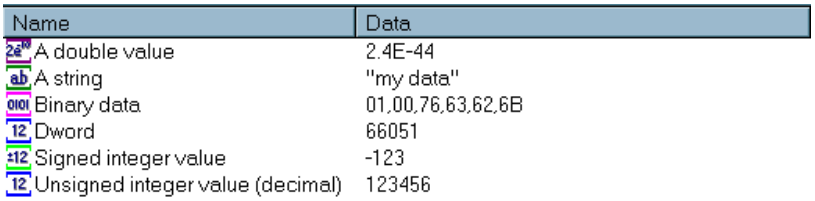
Value-content can be any of the following:

- the string **`double:`** followed by a numeric value.
- a string. You can put anything inside the quotes, including nothing ("").

  Note: To include a backslash in the quoted string, use two adjacent backslashes. To include a double quotation mark, use two adjacent double quotation marks. △
- the string **`hex:`** followed by any number of hexadecimal characters, up to the 255-character limit, separated by commas. If you extend the hexadecimal characters beyond a single line, then end the line with a backslash to indicate that the data continues on the next line. Hexadecimal values can also be referred to as "binary values" in the Registry Editor.
- the string **`dword:`** followed by an unsigned long hexadecimal value.
- the string **`int:`** followed by a signed long integer value.
- the string **`uint:`** followed by an unsigned long integer value.

The following display shows how the different types of values that are described above appear in the Registry Editor:

Display 50.1 Types of Registry Values, Displayed in the Registry Editor



The following list contains a sample of valid registry values:

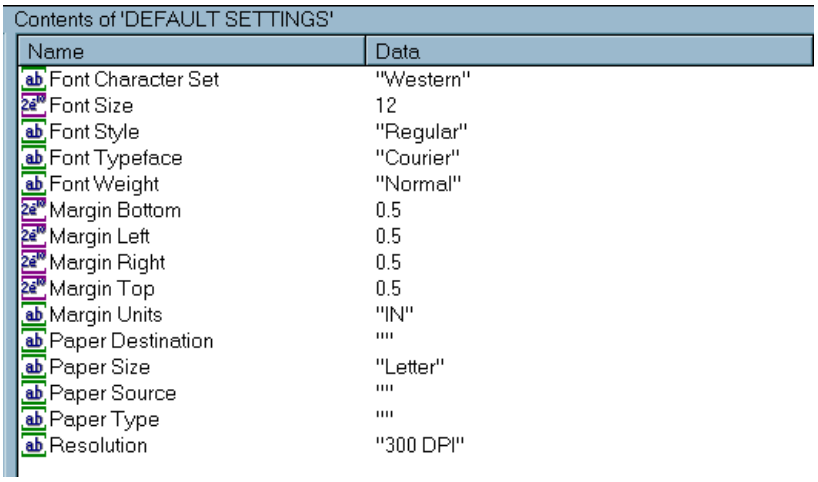
- a double value=double:2.4E-44
- a string="my data"
- binary data=hex: 01,00,76,63,62,6B
- dword=dword:00010203
- signed integer value=int:-123

- 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 50.2 Portion of a Registry Editor Showing Settings for a PostScript Printer



To see what the actual registry text file looks like, you can use PROC REGISTRY to write the contents of the registry key to the SAS log, using the LISTUSER and STARTAT= options.

The following example shows the syntax for sending a SASUSER registry entry to the log:

```
proc registry
     listuser
     startat='sasuser-registry-key-name';
run;
```

The following example shows a value for the STARTAT= option:

```
proc registry
     listuser
     startat='HKEY_SYSTEM_ROOT\CORE\PRINTING\PRINTERS\PostScript\DEFAULT SETTINGS';
run;
```

In the following example, the list of subkeys begins at the CORE\PRINTING\PRINTERS\PostScript\DEFAULT SETTINGS key.

Output 50.1 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;
```

Output: SAS Log

Output 50.2 Output From Importing a File into the SAS Registry

```
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: The file is usually very large. 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;
```

Output: SAS Log

Output 50.3 Output From Exporting a File From the SAS Registry

```
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 one are useful if you want to know the difference between a backup file that was saved with a .txt file extension and the current registry file.

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;
```

Output: SAS Log

Output 50.4 Output From Comparing the Registry to an External File

```
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.
```

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".

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-library';
```

Start PROC REGISTRY and specify the first registry file to be used in the comparison.

```
proc registry compareregl='sasuser.regstry'
```

Limit the comparison to the registry keys including and following the specified registry key. The STARTAT= option limits the scope of the comparison to the EXPLORER subkey under the CORE key. By default the comparison includes the entire contents of both registries.

```
    startat='CORE\EXPLORER'
```

Specify the second registry file to be used in the comparison.

```
   comparereg2='proclib.regstry';
run;
```

Output: SAS Log

Output 50.5 Output From Comparing Registry Files

```
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
```

Example 5: Specifying an Entire Key Sequence with the STARTAT= Option

Procedure features

EXPORT option

STARTAT= option

The following example shows how to use the STARTAT= option. You must specify an entire key sequence if you want to start listing any subkey under the root key. The root key is optional.

Program

```
proc registry export = my-fileref startat='core\explorer\icons';
run;
```

Example 6: Displaying a List of Fonts

Procedure features

LISTHELP option

STARTAT option

The following example writes a list of ODS fonts to the SAS log.

Program

```
proc registry clearsasuser; run;
proc registry listhelp startat='ods\fonts'; run;
proc registry clearsasuser; run;
```

Output: SAS Log

Output 50.6 Partial Log Output from Displaying a List of Fonts

```
NOTE: Contents of SASHELP REGISTRY starting at subkey [ods\fonts]
[  ods\fonts]
  dings="Wingdings"
  monospace="Courier New"
  MTdings="Monotype Sorts"
  MTmonospace="Cumberland AMT"
  MTsans-serif="Albany AMT"
  MTsans-serif-unicode="Monotype Sans WT J"
  MTserif="Thorndale AMT"
  MTserif-unicode="Thorndale Duospace WT J"
  MTsymbol="Symbol MT"
  sans-serif="Arial"
  serif="Times New Roman"
  symbol="Symbol"
[    ja_JP]
    dings="Wingdings"
    monospace="MS Gothic"
    MTdings="Wingdings"
    MTmonospace="MS Gothic"
    MTsans-serif="MS PGothic"
    MTsans-serif-unicode="MS PGothic"
    MTserif="MS PMincho"
    MTserif-unicode="MS PMincho"
    MTsymbol="Symbol"
    sans-serif="MS PGothic"
    serif="MS PMincho"
    symbol="Symbol"
[    ko_KR]
    dings="Wingdings"
    monospace="GulimChe"
    MTdings="Wingdings"
    MTmonospace="GulimChe"
    MTsans-serif="Batang"
    MTsans-serif-unicode="Batang"
    MTserif="Gulim"
    MTserif-unicode="Gulim"
    MTsymbol="Symbol"
    sans-serif="Batang"
    serif="Gulim"
    symbol="Symbol"
[    th_TH]
    dings="Wingdings"
    monospace="Thorndale Duospace WT J"
    MTdings="Monotype Sorts"
    MTmonospace="Cumberland AMT"
    MTsans-serif="Monotype Sans WT J"
    MTsans-serif-unicode="Thorndale Duospace WT J"
    MTserif="Thorndale Duospace WT J"
    MTserif-unicode="Monotype Sans WT J"
    MTsymbol="Symbol MT"
    sans-serif="Angsana New"
    serif="Thorndale Duospace WT J"
    symbol="Symbol"
```

See Also

SAS registry section in *SAS Language Reference: Concepts*

CHAPTER

51

The REPORT Procedure

Overview: REPORT Procedure **981**
- *What Does the REPORT Procedure Do?* **981**
- *What Types of Reports Can PROC REPORT Produce?* **981**
- *What Do the Various Types of Reports Look Like?* **981**

Concepts: REPORT Procedure **986**
- *Laying Out a Report* **986**
  - *Planning the Layout* **986**
  - *Usage of Variables in a Report* **987**
  - *Display Variables* **987**
  - *Order Variables* **987**
  - *Group Variables* **988**
  - *Analysis Variables* **988**
  - *Across Variables* **989**
  - *Computed Variables* **989**
  - *Interactions of Position and Usage* **989**
  - *Statistics That Are Available in PROC REPORT* **991**
- *Using Compute Blocks* **992**
  - *What Is a Compute Block?* **992**
  - *The Purpose of Compute Blocks* **992**
  - *The Contents of Compute Blocks* **992**
  - *Four Ways to Reference Report Items in a Compute Block* **993**
  - *Compute Block Processing* **994**
- *Using Break Lines* **995**
  - *What Are Break Lines?* **995**
  - *Creating Break Lines* **995**
  - *Order of Break Lines* **995**
  - *The Automatic Variable _BREAK_* **995**
- *Using Compound Names* **996**
- *Using Style Elements in PROC REPORT* **997**
  - *Using the STYLE= Option* **997**
  - *Using a Format to Assign a Style Attribute Value* **1000**
  - *Controlling the Spacing between Rows* **1001**
- *Printing a Report* **1001**
  - *Printing with ODS* **1001**
  - *Printing from the REPORT Window* **1001**
  - *Printing with a Form* **1001**
  - *Printing from the Output Window* **1002**
  - *Printing from Noninteractive or Batch Mode* **1002**
  - *Printing from Interactive Line Mode* **1002**
  - *Using PROC PRINTTO* **1002**
- *Storing and Reusing a Report Definition* **1002**

ODS Destinations Supported by PROC REPORT **1003**
In-Database Processing for PROC REPORT **1003**
Syntax: REPORT Procedure **1004**
PROC REPORT Statement **1005**
BREAK Statement **1021**
BY Statement **1027**
CALL DEFINE Statement **1028**
COLUMN Statement **1030**
COMPUTE Statement **1032**
DEFINE Statement **1035**
ENDCOMP Statement **1044**
FREQ Statement **1045**
LINE Statement **1046**
RBREAK Statement **1047**
WEIGHT Statement **1051**
REPORT Procedure Windows **1052**
BREAK **1052**
COMPUTE **1055**
COMPUTED VAR **1056**
DATA COLUMNS **1056**
DATA SELECTION **1057**
DEFINITION **1057**
DISPLAY PAGE **1063**
EXPLORE **1063**
FORMATS **1064**
LOAD REPORT **1065**
MESSAGES **1065**
PROFILE **1066**
PROMPTER **1066**
REPORT **1067**
ROPTIONS **1068**
SAVE DATA SET **1072**
SAVE DEFINITION **1073**
SOURCE **1073**
STATISTICS **1073**
WHERE **1074**
WHERE ALSO **1074**
How PROC REPORT Builds a Report **1075**
Sequence of Events **1075**
Construction of Summary Lines **1076**
Report-Building Examples **1076**
Building a Report That Uses Groups and a Report Summary **1076**
Building a Report That Uses Temporary Variables **1081**
Examples: REPORT Procedure **1087**
Example 1: Selecting Variables for a Report **1087**
Example 2: Ordering the Rows in a Report **1090**
Example 3: Using Aliases to Obtain Multiple Statistics for the Same Variable **1093**
Example 4: Consolidating Multiple Observations into One Row of a Report **1097**
Example 5: Creating a Column for Each Value of a Variable **1099**
Example 6: Displaying Multiple Statistics for One Variable **1103**
Example 7: Storing and Reusing a Report Definition **1105**
Example 8: Condensing a Report into Multiple Panels **1108**
Example 9: Writing a Customized Summary on Each Page **1110**
Example 10: Calculating Percentages **1114**

Example 11: How PROC REPORT Handles Missing Values **1117**
Example 12: Creating and Processing an Output Data Set **1120**
Example 13: Storing Computed Variables as Part of a Data Set **1122**
Example 14: Using a Format to Create Groups **1126**
Example 15: Specifying Style Elements for ODS Output in the PROC REPORT Statement **1129**
Example 16: Specifying Style Elements for ODS Output in Multiple Statements **1134**

Overview: REPORT Procedure

What Does the REPORT Procedure Do?

The REPORT procedure combines features of the PRINT, MEANS, and TABULATE procedures with features of the DATA step in a single report-writing tool that can produce a variety of reports. You can use PROC REPORT in three ways:

- in a nonwindowing environment. In this case, you submit a series of statements with the PROC REPORT statement, just as you do in other SAS procedures. You can submit these statements from the Program Editor with the NOWINDOWS option in the PROC REPORT statement, or you can run SAS in batch, noninteractive, or interactive line mode. (See the information about running SAS in *SAS Language Reference: Concepts*.)
- in an interactive report window environment with a prompting facility that guides you as you build a report.
- in an interactive report window environment without the prompting facility.

This documentation provides reference information about using PROC REPORT in a windowing or nonwindowing environment. For task-oriented documentation for the nonwindowing environment, see SAS Technical Report P-258, *Using the REPORT Procedure in a Nonwindowing Environment, Release 6.07*.

What Types of Reports Can PROC REPORT Produce?

A *detail report* contains one row for every observation selected for the report. Each of these rows is a report row, a *detail report row*. A *summary report* consolidates data so that each row represents multiple observations. Each of these rows is also called a detail row, a *summary report row*.

Both detail and summary reports can contain *summary report lines*(break lines) as well as report rows. A summary line summarizes numerical data for a set of detail rows or for all detail rows. PROC REPORT provides both default and customized summaries. (See "Using Break Lines" on page 995.)

This overview illustrates the types of reports that PROC REPORT can produce. The statements that create the data sets and formats used in these reports are in Example 1 on page 1087. The formats are stored in a permanent SAS library. See "Examples: REPORT Procedure" on page 1087 for more reports and for the statements that create them.

What Do the Various Types of Reports Look Like?

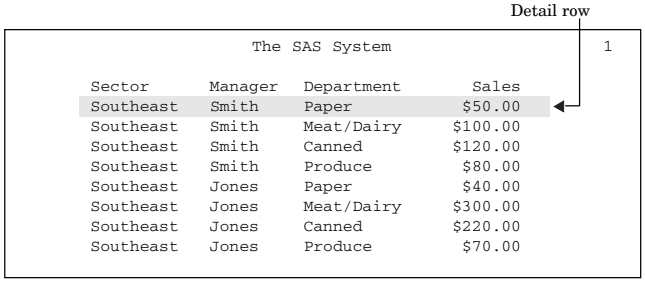
The data set that these reports use contains one day's sales figures for eight stores in a chain of grocery stores.

A simple PROC REPORT step produces a report similar to one produced by a simple PROC PRINT step. Figure 51.1 on page 982 illustrates the simplest type of report that

you can produce with PROC REPORT. The statements that produce the report follow. The data set and formats that the program uses are created in Example 1 on page 1087. Although the WHERE and FORMAT statements are not essential, here they limit the amount of output and make the values easier to understand.

```
libname proclib 'SAS-library';


options nodate pageno=1 linesize=64 pagesize=60
        fmtsearch=(proclib);

proc report data=grocery nowd;
   where sector='se';
   format sector $sctrfmt. manager $mgrfmt.
          dept $deptfmt. sales dollar10.2;
run;
```

Figure 51.1 Simple Detail Report with a Detail Row for Each Observation

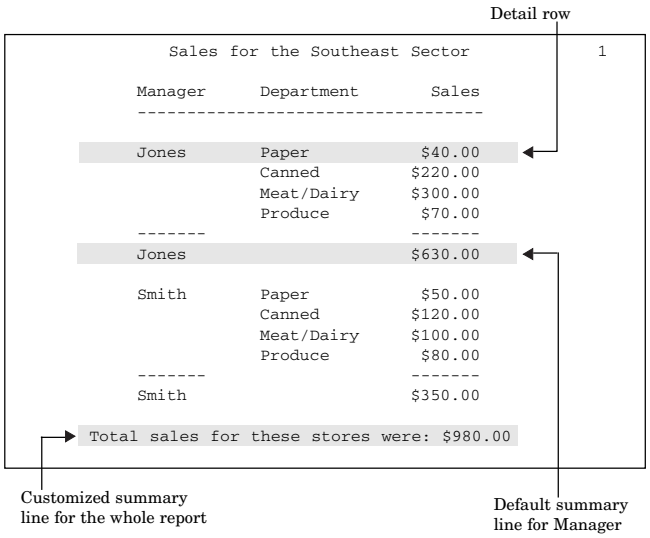


The report in the following figure uses the same observations as the above figure. However, the statements that produce this report

- order the rows by the values of Manager and Department
- create a default summary line for each value of Manager
- create a customized summary line for the whole report. A customized summary lets you control the content and appearance of the summary information, but you must write additional PROC REPORT statements to create one.

For an explanation of the program that produces this report, see Example 2 on page 1090.

Figure 51.2 Ordered Detail Report with Default and Customized Summaries

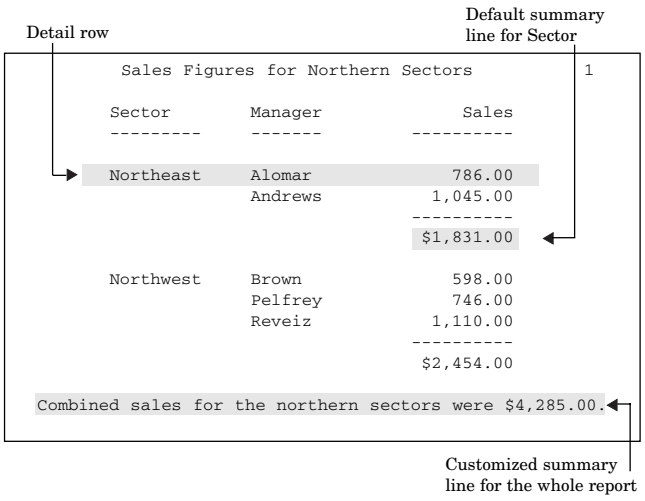


The summary report in the following figure contains one row for each store in the northern sector. Each detail row represents four observations in the input data set, one observation for each department. Information about individual departments does not appear in this report. Instead, the value of Sales in each detail row is the sum of the values of Sales in all four departments. In addition to consolidating multiple observations into one row of the report, the statements that create this report

- customize the text of the column headings
- create default summary lines that total the sales for each sector of the city
- create a customized summary line that totals the sales for both sectors.

For an explanation of the program that produces this report, see Example 4 on page 1097.

Figure 51.3 Summary Report with Default and Customized Summaries

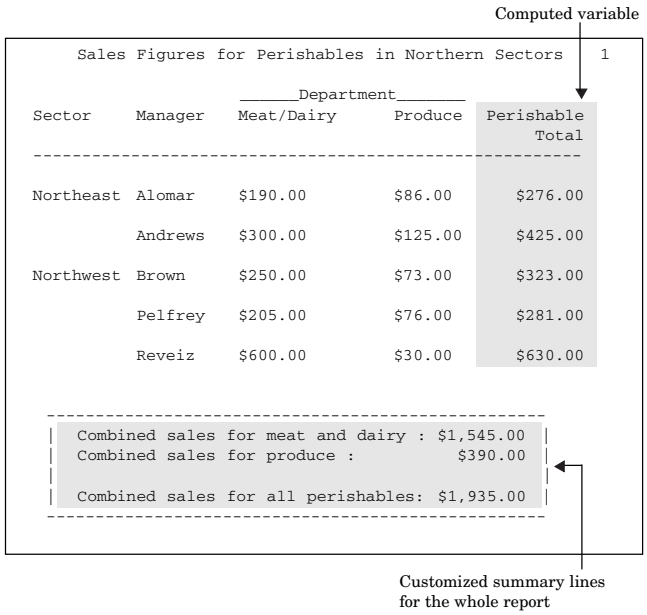


The summary report in the following figure is similar to the above figure. The major difference is that it also includes information for individual departments. Each selected value of Department forms a column in the report. In addition, the statements that create this report

- compute and display a variable that is not in the input data set
- double-space the report
- put blank lines in some of the column headings.

For an explanation of the program that produces this report, see Example 5 on page 1099.

Figure 51.4 Summary Report with a Column for Each Value of a Variable

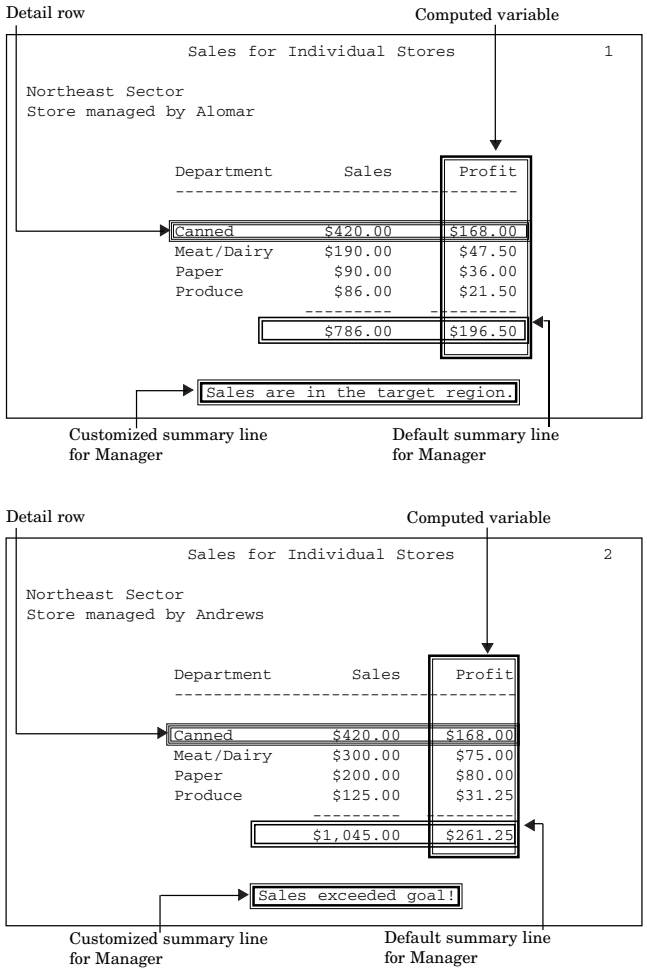


The customized report in the following figure shows each manager's store on a separate page. Only the first two pages appear here. The statements that create this report create

- a customized heading for each page of the report
- a computed variable (Profit) that is not in the input data set
- a customized summary with text that is dependent on the total sales for that manager's store.

For an explanation of the program that produces this report, see Example 9 on page 1110.

Figure 51.5 Customized Summary Report



The report in the following figure uses customized style elements to control things like font faces, font sizes, and justification, as well as the width of the border of the table and the width of the spacing between cells. This report was created by using the HTML destination of the Output Delivery System (ODS) and the STYLE= option in several statements in the procedure.

For an explanation of the program that produces this report, see Example 16 on page 1134. For information about ODS, see “Output Delivery System” on page 33.

Figure 51.6 HTML Output

Sales for the Southeast Sector

Manager	Department	Sales
Jones	Paper	40
	Canned	220
	Meat/Dairy	300
	Produce	70
Jones		630
Subtotal for Jones is $630.00.		
Smith	Paper	50
	Canned	120
	Meat/Dairy	100
	Produce	80
Smith		350
Subtotal for Smith is $350.00.		
Total for all departments is: $980.00.		

Concepts: REPORT Procedure

Laying Out a Report

Planning the Layout

Report writing is simplified if you approach it with a clear understanding of what you want the report to look like. The most important thing to determine is the layout of the report. To design the layout, ask yourself the following types of questions:

- What do I want to display in each column of the report?
- In what order do I want the columns to appear?
- Do I want to display a column for each value of a particular variable?
- Do I want a row for every observation in the report, or do I want to consolidate information for multiple observations into one row?
- In what order do I want the rows to appear?

When you understand the layout of the report, use the COLUMN and DEFINE statements in PROC REPORT to construct the layout.

The COLUMN statement lists the items that appear in the columns of the report, describes the arrangement of the columns, and defines headings that span multiple columns. A report item can be

- a data set variable
- a statistic calculated by the procedure
- a variable that you compute from other items in the report.

Omit the COLUMN statement if you want to include all variables in the input data set in the same order as they occur in the data set.

Note: If you start PROC REPORT in the interactive report window environment without the COLUMN statement, then the initial report includes only as many variables as will fit on one page. △

The DEFINE statement (or, in the interactive report window environment, the DEFINITION window) defines the characteristics of an item in the report. These characteristics include how PROC REPORT uses the item in the report, the text of the column heading, and the format to use to display values.

Usage of Variables in a Report

Much of a report's layout is determined by the usages that you specify for variables in the DEFINE statements or DEFINITION windows. For data set variables, these usages are

DISPLAY

ORDER

ACROSS

GROUP

ANALYSIS

A report can contain variables that are not in the input data set. These variables must have a usage of COMPUTED.

Display Variables

A report that contains one or more display variables has a row for every observation in the input data set. Display variables do not affect the order of the rows in the report. If no order variables appear to the left of a display variable, then the order of the rows in the report reflects the order of the observations in the data set. By default, PROC REPORT treats all character variables as display variables.

Featured in: Example 1 on page 1087

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 1090

Group Variables

If a report contains one or more group variables, then PROC REPORT tries to consolidate into one row all observations from the data set that have a unique combination of formatted values for all group variables.

When PROC REPORT creates groups, it orders the detail rows by the ascending, formatted values of the group variable. You can change the default order with ORDER= and DESCENDING in the DEFINE statement or with the DEFINITION window.

If the report contains multiple group variables, then the REPORT procedure establishes the order of the detail rows by sorting these variables from left to right in the report. PROC REPORT does not repeat the values of a group variable from one row to the next if the value does not change, unless a group variable to its left changes values.

If you are familiar with procedures that use class variables, then you will see that group variables are class variables that are used in the row dimension in PROC TABULATE.

Note: You cannot always create groups. PROC REPORT cannot consolidate observations into groups if the report contains any order variables or any display variables that do not have one or more statistics associated with them. (See "COLUMN Statement" on page 1030.) In the interactive report window environment, if PROC REPORT cannot immediately create groups, then the procedure changes all display and order variables to group variables so that it can create the group variable that you requested. In the nonwindowing environment, it returns to the SAS log a message that explains why it could not create groups. Instead, it creates a detail report that displays group variables the same way as it displays order variables. Even when PROC REPORT creates a detail report, the variables that you define as group variables retain that usage in their definitions. △

Featured in: Example 4 on page 1097

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 1021 and "RBREAK Statement" on page 1047

Featured in: Example 2 on page 1090, Example 3 on page 1093, Example 4 on page 1097, and Example 5 on page 1099

Note: Be careful when you use SAS dates in reports that contain summary lines. SAS dates are numeric variables. Unless you explicitly define dates as some other type of variable, PROC REPORT summarizes them. △

Across Variables

PROC REPORT creates a column for each value of an across variable. PROC REPORT orders the columns by the ascending, formatted values of the across variable. You can change the default order with ORDER= and DESCENDING in the DEFINE statement or with the DEFINITION window. If no other variable helps define the column. (See "COLUMN Statement" on page 1030), 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 like class variables that are used in the column dimension with PROC TABULATE. Generally, you use Across variables in conjunction with order or group variables.

Featured in: Example 5 on page 1099

Computed Variables

Computed variables are variables that you define for the report. They are not in the input data set, and PROC REPORT does not add them to the input data set. However, computed variables are included in an output data set if you create one.

In the interactive report window environment, you add a computed variable to a report from the COMPUTED VAR window.

In the nonwindowing environment, you add a computed variable by

- including the computed variable in the COLUMN statement
- defining the variable's usage as COMPUTED in the DEFINE statement
- computing the value of the variable in a compute block associated with the variable.

Featured in: Example 5 on page 1099, Example 10 on page 1114, and Example 13 on page 1122

Interactions of Position and Usage

The position and usage of each variable in the report determine the report's structure and content. PROC REPORT orders the rows of the report according to the values of order and group variables, considered from left to right as specified in the report window or the COLUMN statement. Similarly, PROC REPORT orders columns for an across variable from left to right, according to the values of the variable.

Several items can collectively define the contents of a column in a report. For example, in the following figure, the values that appear in the third and fourth columns

are collectively determined by Sales, an analysis variable, and by Department, an across variable. You create this type of report with the COLUMN statement or, in the interactive report window environment, by placing report items above or below each other. This arrangement is called stacking items in the report because each item generates a heading, and the headings are stacked one above the other.

Figure 51.7 Stacking Department and Sales

Sales Figures for Perishables in Northern Sectors

Sector	Manager	Department: Meat/Dairy	Department: Produce	Perishable Total
Northeast	Alomar	$190.00	$86.00	$276.00
	Andrews	$300.00	$125.00	$425.00
Northwest	Brown	$250.00	$73.00	$323.00
	Pelfrey	$205.00	$76.00	$281.00
	Reveiz	$600.00	$30.00	$630.00

When you use multiple items to define the contents of a column, at most one of the following can be in a column:

- a display variable with or without a statistic above or below it
- an analysis variable with or without a statistic above or below it
- an order variable
- a group variable
- a computed variable.

More than one of these items in a column creates a conflict for PROC REPORT about which values to display.

The following table shows which report items can share a column.

Note: You cannot stack order variables with other report items. △

Table 51.1 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	STD
MEAN	STDERR
MIN	SUM
MODE	SUMWGT
N	USS
NMISS	VAR
PCTN	
Quantile statistic keywords	
MEDIAN | P50	Q3 | P75
P1	P90
P5	P95
P10	P99
Q1 | P25	QRANGE
Hypothesis testing keyword	
PRT | PROBT	T

These statistics, the formulas that are used to calculate them, and their data requirements are discussed in "Keywords and Formulas" on page 1536.

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 995.)

Note: When you use the COMPUTE statement, you do not have to use a corresponding BREAK or RBREAK statement. (See Example 2 on page 1090, 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 1110, 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 1028.)
- define or change the value for a report item, such as showing the word "Total" on a summary line.

A compute block that is associated with a location can write a customized summary.

In addition, all compute blocks can use most SAS language elements to perform calculations. (See "The Contents of Compute Blocks" on page 992.) A PROC REPORT step can contain multiple compute blocks, but they cannot be nested.

The Contents of Compute Blocks

In the interactive report window environment, a compute block is in a COMPUTE window. In the nonwindowing environment, a compute block begins with a COMPUTE statement and ends with an ENDCOMP statement. Within a compute block, you can use these SAS language elements:

- %INCLUDE statement
- these DATA step statements:

ARRAY	END
array-reference	IF-THEN/ELSE
assignment	LENGTH
CALL	RETURN
CONTINUE	SELECT
DO (all forms)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 1046.)
- 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 1028.)
- Any compute block can reference the automatic variable _BREAK_ . (See "The Automatic Variable _BREAK_" on page 995.)

Four Ways to Reference Report Items in a Compute Block

A compute block can reference any report item that forms a column in the report (whether the column is visible). You reference report items in a compute block in one of four ways:

- by name.
- by a compound name that identifies both the variable and the name of the statistic that you calculate with it. A compound name has this form

  variable-name.statistic

- by an alias that you create in the COLUMN statement or in the DEFINITION window.
- by column number, in the form

  '_C*n*_'

  where *n* is the number of the column (from left to right) in the report.

  Note: The only time a column number is necessary is when a COMPUTED variable is sharing a column with an ACROSS variable. △

  Note: Even though the columns that you define with NOPRINT and NOZERO do not appear in the report, you must count them when you are referencing columns by number. See the discussion of NOPRINT on page 1041 and NOZERO on page 1041. △

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 1093, which references analysis variables by their aliases; Example 5 on page 1099, which references variables by column number; and Example 10 on page 1114, which references group variables and computed variables by name.

Compute Block Processing

PROC REPORT processes compute blocks in two different ways.

- If a compute block is associated with a location, then PROC REPORT executes the compute block only at that location. Because PROC REPORT calculates statistics for groups before it actually constructs the rows of the report, statistics for sets of report rows are available before or after the rows are displayed, as are values for any variables based on these statistics.
- If a compute block is associated with a report item, then PROC REPORT executes the compute block on every row of the report when it comes to the column for that item. The value of a computed variable in any row of a report is the last value assigned to that variable during that execution of the DATA step statements in the compute block. PROC REPORT assigns values to the columns in a row of a report from left to right. Consequently, you cannot base the calculation of a computed variable on any variable that appears to its right in the report.

Note: PROC REPORT recalculates computed variables at breaks. For details about compute block processing see "How PROC REPORT Builds a Report" on page 1075. △

Using Break Lines

What Are Break Lines?

Break lines are lines of text (including blanks) that appear at particular locations, called *breaks*, in a report. A report can contain multiple breaks. Generally, break lines are used to visually separate parts of a report, to summarize information, or both. They can occur

- at the beginning or end of a report
- at the top or bottom of each page
- between sets of observations (whenever the value of a group or order variable changes).

Break lines can contain

- text
- values calculated for either a set of rows or for the whole report.

Creating Break Lines

There are two ways to create break lines. The first way is simpler. It produces a default summary. The second way is more flexible. It produces a customized summary and provides a way to slightly modify a default summary. Default summaries and customized summaries can appear at the same location in a report.

Default summaries are produced with the BREAK statement, the RBREAK statement, or the BREAK window. You can use default summaries to visually separate parts of the report, to summarize information for numeric variables, or both. Options provide some control over the appearance of the break lines, but if you choose to summarize numeric variables, then you have no control over the content and the placement of the summary information. (A break line that summarizes information is a summary line.)

Customized summaries are produced in a compute block. You can control both the appearance and content of a customized summary, but you must write the code to do so.

Order of Break Lines

You control the order of the lines in a customized summary. However, PROC REPORT controls the order of lines in a default summary and the placement of a customized summary relative to a default summary. When a default summary contains multiple break lines, the order in which the break lines appear is

1. overlining or double overlining (in traditional SAS monospace output only)
2. summary line
3. underlining or double underlining
4. blank line (in traditional SAS monospace output only)
5. page break.

In traditional SAS monospace output only, if you define a customized summary for the same location, then customized break lines appear after underlining or double underlining.

The Automatic Variable _BREAK_

PROC REPORT automatically creates a variable called _BREAK_. This variable contains

- a blank if the current line is not part of a break
- the value of the break variable if the current line is part of a break between sets of observations
- the value **_RBREAK_** if the current line is part of a break at the beginning or end of the report
- the value **_PAGE_** if the current line is part of a break at the beginning or end of a page.

Using Compound Names

When you use a statistic in a report, you generally refer to it in compute blocks by a compound name like Sales.sum. However, in different parts of the report, that same name has different meanings. Consider the report in the following output. The statements that create the output follow. The user-defined formats that are used are created by a PROC FORMAT step on page 1089.

```
libname proclib 'SAS-library';


options nodate pageno=1 linesize=64
        pagesize=60 fmtsearch=(proclib);
proc report data=grocery nowindows;
   column sector manager sales;
   define sector / group format=$sctrfmt.;
   define sales  / analysis sum
                   format=dollar9.2;
   define manager / group format=$mgrfmt.;
   break after sector / summarize skip ol;
   rbreak after / summarize dol dul;
   compute after;
      sector='Total:';
   endcomp;
run;
```

Output 51.1 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: When you refer in a compute block to a statistic that has an alias, do not use a compound name. Generally, you must use the alias. However, if the statistic shares a column with an across variable, then you must reference it by column number. (See "Four Ways to Reference Report Items in a Compute Block" on page 993.) △

Using Style Elements in PROC REPORT

Using the STYLE= Option

If you use the Output Delivery System to create HTML, RTF, or Printer output from PROC REPORT, then you can use the STYLE= option to specify style elements for the procedure to use in various parts of the report. Style elements determine presentation attributes like font type, font weight, color, and so on. For information about the style attributes and their values, 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 51.2 Location Values

Location Value	Part of Report Affected
CALLDEF	Cells identified by a CALL DEFINE statement
COLUMN	Column cells
HEADER|HDR	Column headings
LINES	Lines generated by LINE statements
REPORT	Report as a whole
SUMMARY	Summary lines

The valid and default values for *location* vary by what statement the STYLE= option appears in. The following table shows valid and default 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. Refer to "ODS Style Elements" for a list of SAS provided style elements. 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 51.3 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 PROC REPORT. Note that not all style 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.

Table 51.4 Style Attributes for PROC REPORT and PROC TABULATE

Attribute	PROC REPORT STATEMENT: REPORT Area	PROC REPORT Areas: CALLDEF, COLUMN, HEADER, LINES, SUMMARY	PROC TABULATE STATEMENT: TABLE	PROC TABULATE STATEMENTS: VAR, CLASS, BOX Opt, CLASSLEV, KEYWORD
ASIS=	X	X		X
BACKGOUNDCOLOR=	X	X	X	X
BACKGOUNDIMAGE=	X	X	X	X
BORDERBOTTOMCOLOR=	X	X		X
BORDERBOTTOMSTYLE=	X	X	X	X
BORDERBOTTOMWIDTH=	X	X	X	X
BORDERCOLOR=	X	X		X
BORDERCOLORDARK=	X	X	X	X
BORDERCOLORLIGHT=	X	X	X	X
BORDERTOPCOLOR=	X	X		X
BORDERTOPSTYLE=	X	X	X	X
BORDERTOPWIDTH=	X	X	X	X
BORDERWIDTH=	X	X	X	X
CELLPADDING=	X		X	
CELLSPACING=	X		X	
CLASS=	X	X	X	X
COLOR=	X	X	X	
FLYOVER=	X	X		X
FONT=	X	X	X	X
FONTFAMILY=	X	X	X	X
FONTSIZE=	X	X	X	X
FONTSTYLE=	X	X	X	X
FONTWEIGHT=	X	X	X	X
FONTWIDTH=	X	X	X	X
FRAME=	X		X	
HEIGHT=	X	X		X
HREFTARGET=		X		X

Attribute	PROC REPORT STATEMENT: REPORT Area	PROC REPORT Areas: CALLDEF, COLUMN, HEADER, LINES, SUMMARY	PROC TABULATE STATEMENT: TABLE	PROC TABULATE STATEMENTS: VAR, CLASS, BOX Opt, CLASSLEV, KEYWORD
HTMLSTYLE=	X	X	X	X
NOBREAKSPACE=	X	X		X
POSTHTML=	X	X	X	X
POSTIMAGE=	X	X	X	X
POSTTEXT=	X	X	X	X
PREHTML=	X	X	X	X
PREIMAGE=	X	X	X	X
PRETEXT=	X	X	X	X
PROTECTSPECIALCHARS=		X		X
RULES=	X		X	
TAGATTR=	X	X		X
TEXTALIGN=	X	X	X	X
URL=		X		X
VERTICALALIGN=		X		X
WIDTH=	X	X	X	X

Specifications in a PROC REPORT statement other than the PROC REPORT location override the same specification in the PROC REPORT statement. However, any style attributes that you specify in the PROC REPORT statement and do not override in another PROC REPORT statement are inherited. For example, if you specify a blue background and a white foreground for all column headings in the PROC REPORT statement, and you specify a gray background for the column headings of a variable in the PROC REPORT DEFINE statement, then the background for that particular column heading is gray, and the foreground is white (as specified in the PROC REPORT statement).

Using a Format to Assign a Style Attribute Value

You can use a format to assign a style attribute value. For example, the following code assigns a red background color to cells in the Profit column for which the value is negative, and a green background color where the values are positive:

```
proc format;
   value proffmt low-<0='red'
                 0-high='green';
run;
ods html body='external-HTML-file';
proc report data=profits nowd;
   title 'Profits for Individual Stores';
   column Store Profit;
   define Store / display 'Store';
```

```
   define Profit / display 'Profit' style=[backgroundcolor=proffmt.];
run;
ods html close;
```

Controlling the Spacing between Rows

Users frequently need to "shrink" a report to fit more rows on a page. Shrinking a report involves changing both the font size and the spacing between the rows. In order to give maximum flexibility to the user, ODS uses the font size that is specified for the REPORT location to calculate the spacing between the rows. Therefore, to shrink a table, change the font size for both the REPORT location and the COLUMN location. Here is an example:

```
proc report nowindows data=libref.data---set-name
            style(report)=[fontsize=8pt]
            style(column)=[font=(Arial, 8pt)];
```

Printing a Report

Printing with ODS

Printing reports with the Output Delivery System is much simpler and provides more attractive output than the older methods of printing that are documented here. For best results, use an output destination in the ODS printer family or RTF. For details about these destinations and on using the ODS statements, see *SAS Output Delivery System: User's Guide*.

Printing from the REPORT Window

By default, if you print from the REPORT window, then the report is routed directly to your printer. If you want, you can specify a form to use for printing. (See "Printing with a Form" on page 1001.) Forms specify things like the type of printer that you are using, text format, and page orientation.

Note: Forms are available only when you run SAS from an interactive report window environment. △

Operating Environment Information: Printing is implemented differently in different operating environments. For information related to printing, consult *SAS Language Reference: Concepts*. Additional information might be available in the SAS documentation for your operating environment. △

Printing with a Form

To print with a form from the REPORT window:

1. Specify a form. You can specify a form with the FORMNAME command or, in some cases, through the **File** menu.
2. Specify a print file if you want the output to go to a file instead of directly to the printer. You can specify a print file with the PRTFILE command or, in some cases, through the **File** menu.
3. Issue the PRINT or PRINT PAGE command from the command line or from the **File** menu.

4 If you specified a print file, then do the following:

   a Free the print file. You can free a file with the FREE command or, in some cases, through **Print utilities** in the **File** menu. You cannot view or print the file until you free it.
   b Use operating environment commands to send the file to the printer.

Printing from the Output Window

If you are running PROC REPORT with the NOWINDOWS option, then the default destination for the output is the Output window. Use the commands in the **File** menu to print the report.

Printing from Noninteractive or Batch Mode

If you use noninteractive or batch mode, then SAS writes the output either to the display or to external files, depending on the operating environment and on the SAS options that you use. Refer to the SAS documentation for your operating environment for information about how these files are named and where they are stored.

You can print the output file directly or use PROC PRINTTO to redirect the output to another file. In either case, no form is used, but carriage control characters are written if the destination is a print file.

Use operating environment commands to send the file to the printer.

Printing from Interactive Line Mode

If you use interactive line mode, then by default the output and log are displayed on the screen immediately following the programming statements. Use PROC PRINTTO to redirect the output to an external file. Then use operating environment commands to send the file to the printer.

Using PROC PRINTTO

PROC PRINTTO defines destinations for the SAS output and the SAS log. (See Chapter 44, “The PRINTTO Procedure,” on page 887.)

PROC PRINTTO does not use a form, but it does write carriage control characters if you are writing to a print file.

Note: You need two PROC PRINTTO steps. The first PROC PRINTTO step precedes the PROC REPORT step. It redirects the output to a file. The second PROC PRINTTO step follows the PROC REPORT step. It reestablishes the default destination and frees the output file. You cannot print the file until PROC PRINTTO frees it. △

Storing and Reusing a Report Definition

The OUTREPT= option in the PROC REPORT statement stores a report definition in the specified catalog entry. If you are working in the nonwindowing environment, then the definition is based on the PROC REPORT step that you submit. If you are in the interactive report window environment, then the definition is based on the report that is in the REPORT window when you end the procedure. SAS assigns an entry type of REPT to the entry.

In the interactive report window environment, you can save the definition of the current report by selecting **File ▶ Save Report**. A report definition might differ from the SAS program that creates the report. See the discussion of OUTREPT= on page 1015.

You can use a report definition to create an identically structured report for any SAS data set that contains variables with the same names as the ones that are used in the report definition. Use the REPORT= option in the PROC REPORT statement to load a report definition when you start PROC REPORT. In the interactive report window environment, load a report definition from the LOAD REPORT window by selecting **File ► Open Report**

ODS Destinations Supported by PROC REPORT

Before SAS 9.2, the ODS DOCUMENT and the ODS OUTPUT destinations were unsupported by PROC REPORT. Now, PROC REPORT supports all ODS destinations. Refer to Understanding ODS Destinations in *SAS Output Delivery System: User's Guide* for information about all of the ODS destinations.

The DOCUMENT destination enables you to restructure, navigate, and replay your data in different ways and to different destinations without rerunning your analysis or repeating your database query. The DOCUMENT destination makes your entire output stream available in "raw" form and accessible to you to customize. The output is kept in the original internal representation as a data component plus a table definition. When the output is in a DOCUMENT form, it is possible to rearrange, restructure, and reformat without rerunning your analysis. Refer to the ODS DOCUMENT procedure in the *SAS Output Delivery System: User's Guide* for additional information.

The OUTPUT destination produces SAS output data sets. Because ODS already knows the logical structure of the data and its native form, ODS can output a SAS data set that represents exactly the same resulting data set that the procedure worked with internally. Refer to the ODS OUTPUT statement in the *SAS Output Delivery System: User's Guide* for additional information.

In-Database Processing for PROC REPORT

When the DATA= input data set is stored as a table or view in a database management system (DBMS), the PROC REPORT procedure can use in-database processing to perform most of its work within the database. In-database processing can provide the advantages of faster processing and reduced data transfer between the database and SAS software.

PROC REPORT performs in-database processing by using SQL implicit pass-through. The procedure generates SQL queries that are based on the statements and the PROC REPORT options that are used as well as the output statistics that are specified in the procedure. The database executes these SQL queries and the results of the query are then transmitted to PROC REPORT. To examine the generated SQL, set the SASTRACE= option.

If the SAS format definitions have not been deployed in the database, the in-database aggregation occurs on the raw values, and the relevant formats are applied by SAS as the results' set is merged into the PROC REPORT internal structures. For more information, see the section "Deploying and Using SAS Formats" in *SAS/ACCESS for Relational Databases: Reference*.

In-database processing will not occur if the PROC REPORT step contains variables with usage types DISPLAY or ORDER.

The following statistics are supported for in-database processing: N, NMISS, MIN, MAX, MEAN, RANGE, SUM, SUMWGT, CSS, USS, VAR, STD, STDERR, and CV.

Weighting for in-database processing is supported only for N, NMISS, MIN, MAX, RANGE, SUM, SUMWGT, and MEAN.

The SQLGENERATION system option or LIBNAME statement option controls whether and how in-database procedures are run inside the database. By default, the

in-database procedures are run inside the database when possible. There are many data set options that will prevent in-database processing. For a complete listing, refer to "Overview of In-Database Procedures" in *SAS/ACCESS for Relational Databases: Reference*.

For more information about in-database processing, see *SAS/ACCESS for Relational Databases: Reference*.

Syntax: REPORT Procedure

Tip: Supports the Output Delivery System. See "Output Delivery System: Basic Concepts in *SAS Output Delivery System: User's Guide* for details.

Tip: You can use the ATTRIB, FORMAT, LABEL, and WHERE statements. See Chapter 3, "Statements with the Same Function in Multiple Procedures," on page 35 for details. You can also use any global statements. See "Global Statements" on page 20 for a list.

PROC 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*;

Task	Statement
Produce a summary or detail report	"PROC REPORT Statement" on page 1005
Produce a default summary at a change in the value of a group or order variable	"BREAK Statement" on page 1021
Create a separate report for each BY group	"BY Statement" on page 1027
Set the value of an attribute for a particular column in the current row	"CALL DEFINE Statement" on page 1028
Describe the arrangement of all columns and of headings that span more than one column	"COLUMN Statement" on page 1030
Specify one or more programming statements that PROC REPORT executes as it builds the report	"COMPUTE Statement" on page 1032 and "ENDCOMP Statement" on page 1044
Describe how to use and display a report item	"DEFINE Statement" on page 1035
Treat observations as if they appear multiple times in the input data set	"FREQ Statement" on page 1045
Provide a subset of features of the PUT statement for writing customized summaries	"LINE Statement" on page 1046
Produce a default summary at the beginning or end of a report or at the beginning and end of each BY group	"RBREAK Statement" on page 1047
Specify weights for analysis variables in the statistical calculations	"WEIGHT Statement" on page 1051

PROC REPORT Statement

PROC REPORT <*option(s)*>;

Task	Option
Specify the input data set	DATA=
Specify the output data set	OUT=
Override the SAS system option THREADS | NOTHREADS	THREADS | NOTHREADS
Select the interactive report window or the nonwindowing environment	WINDOWS | NOWINDOWS
Use a report that was created before compute blocks required aliases (before Release 6.11)	NOALIAS
Control the statistical analysis	

Task	Option
Specify the divisor to use in the calculation of variances	VARDEF=
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=
Exclude observations with nonpositive weight values from the analysis.	EXCLNPWGT
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	
Resets the page number between BY groups	BYPAGENO
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. Use for monospace output only.	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=*

Task	Option
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 headings	
Underline all column headings and the spaces between them	HEADLINE*
Write a blank line beneath all column headings	HEADSKIP*
Suppress column headings	NOHEADER
Write *name*= in front of each value in the report, where *name*= is the column heading for the value	NAMED
Specify the split character	SPLIT=
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=
Specify that a single cell will occupy the column in all the rows for which the value is the same. Only applies to ODS MARKUP, PDF, RTF, and PRINTER destinations.	SPANROWS
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 interactive report window 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 headings from the body of the report
- separate rows and columns from each other
- separate values in a summary line from other values in the same columns
- separate a customized summary from the rest of the report.

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Interaction: You cannot use BOX if you use WRAP in the PROC REPORT statement or in the ROPTIONS window or if you use FLOW in any item definition.

See also: the discussion of FORMCHAR= on page 1010

Featured in: Example 12 on page 1120

BYPAGENO=*number*
If a BY statement is present, specifies the listing page number at the start of each BY group.

Range: any positive integer greater than 0.

Restriction: This option has no effect if a BY statement is not present.

CENTER | NOCENTER
specifies whether to center or left-justify the report and summary text (customized break lines).

PROC REPORT honors the first of these centering specifications that it finds:

- the CENTER or NOCENTER option in the PROC REPORT statement or the CENTER toggle in the ROPTIONS window
- the CENTER or NOCENTER option stored in the report definition that is loaded with REPORT= in the PROC REPORT statement
- the SAS system option CENTER or NOCENTER.

Interaction: When CENTER is in effect, PROC REPORT ignores spacing that precedes the leftmost variable in the report.

COLWIDTH=*column-width*
specifies the default number of characters for columns containing computed variables or numeric data set variables.

Default: 9

Range: 1 to the line size

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 1040.)

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 1090

COMMAND
displays command lines rather than menu bars in all REPORT windows.

After you have started PROC REPORT in the interactive report window environment, you can display the menu bars in the current window by issuing the COMMAND command. You can display the menu bars in all PROC REPORT windows by issuing the PMENU command. The PMENU command affects all the windows in your SAS session. Both of these commands are toggles.

You can store a setting of COMMAND in your report profile. PROC REPORT honors the first of these settings that it finds:

- the COMMAND option in the PROC REPORT statement
- the setting in your report profile.

Restriction: This option has no effect in the nonwindowing environment.

COMPLETECOLS | NOCOMPLETECOLS
creates all possible combinations for the values of the across variables even if one or more of the combinations do not occur within the input data set. Consequently, the column headings are the same for all logical pages of the report within a single BY group.

Default: COMPLETECOLS

Interaction: The PRELOADFMT option in the DEFINE statement ensures that PROC REPORT uses all user-defined format ranges for the combinations of across variables, even when a frequency is zero.

COMPLETEROWS | NOCOMPLETEROWS
displays all possible combinations of the values of the group variables, even if one or more of the combinations do not occur in the input data set. Consequently, the row headings are the same for all logical pages of the report within a single BY group.

Default: NOCOMPLETEROWS

Interaction: The PRELOADFMT option in the DEFINE statement ensures that PROC REPORT uses all user-defined format ranges for the combinations of group variables, even when a frequency is zero.

CONTENTS='*link-text*'
specifies the text for the entries in the HTML contents file or PDF table of contents for the output that is produced by PROC REPORT. For information about HTML and PDF output, see "Output Delivery System" on page 33.

Restriction: For HTML output, the CONTENTS= option has no effect in the HTML body file. It affects only the HTML contents file.

Interaction: For RTF output, the CONTENTS= option has no effect on the RTF body file unless you turn on the CONTENTS=YES option in the ODS RTF statement. In that case, a Table of Contents page is inserted at the front of your RTF output file. Your CONTENTS= option text from PROC REPORT will then show up in this separate Table of Contents page.

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

Main discussion: "Input Data Sets" on page 20

EXCLNPWGT
excludes observations with nonpositive weight values (zero or negative) from the analysis. By default, PROC REPORT treats observations with negative weights like observations with zero weights and counts them in the total number of observations.

Alias: EXCLNPWGTS

Requirement: You must use a WEIGHT statement.

See also: "WEIGHT Statement" on page 1051

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 51.5 on page 1011 shows the formatting characters that PROC REPORT uses. Figure 51.8 on page 1012 illustrates the use of some commonly used formatting character in the output from PROC REPORT.

formatting-character(s)
lists the characters to use for the specified positions. PROC REPORT assigns characters in *formatting-character(s)* to *position(s)*, in the order in which they are listed. For example, the following option assigns the asterisk (*) to the third formatting character, the pound sign (#) to the seventh character, and does not alter the remaining characters:

```
formchar(3,7)='*#'
```

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

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

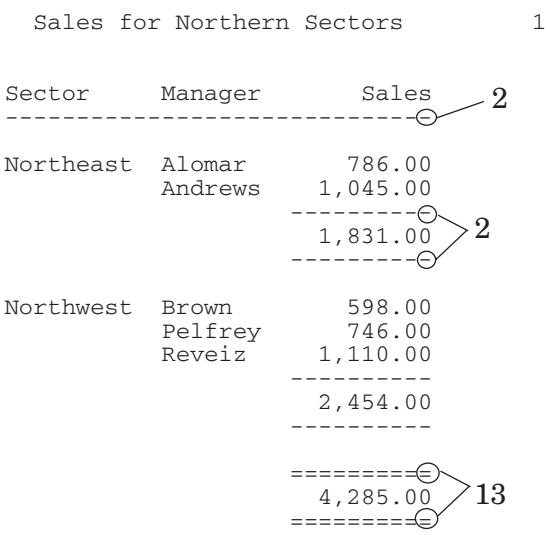
Tip: You can use any character in *formatting-characters*, including hexadecimal characters. If you use hexadecimal characters, then you must put **x** after the closing quotation mark. For example, the following option assigns the hexadecimal character 2D to the third formatting character, the hexadecimal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

```
formchar(3,7)='2D7C'x
```

Table 51.5 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 51.8 Formatting Characters in PROC REPORT Output



HEADLINE
underlines all column headings and the spaces between them at the top of each page of the report.

The HEADLINE option underlines with the second formatting character. (See the discussion of FORMCHAR= on page 1010 .)

Default: hyphen (-)

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Tip: In traditional (monospace) SAS output, you can underline column headings without underlining the spaces between them, by using two hyphens **('--')** as the last line of each column heading instead of using HEADLINE.

Featured in: Example 2 on page 1090 and Example 8 on page 1108

HEADSKIP
writes a blank line beneath all column headings (or beneath the underlining that the HEADLINE option writes) at the top of each page of the report.

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Featured in: Example 2 on page 1090

HELP=*libref.catalog*
identifies the library and catalog containing user-defined help for the report. This help can be in CBT or HELP catalog entries. You can write a CBT or HELP entry for each item in the report with the BUILD procedure in SAS/AF software. Store all such entries for a report in the same catalog.

Specify the entry name for help for a particular report item in the DEFINITION window for that report item or in a DEFINE statement.

Restriction: This option only works in the Report Window.

LIST
writes to the SAS log the PROC REPORT code that creates the current report. This listing might differ in these ways from the statements that you submit:

- It shows some defaults that you might not have specified.
- It omits some statements that are not specific to the REPORT procedure, whether you submit them with the PROC REPORT step or had previously submitted them. These statements include

  BY

  FOOTNOTE

  FREQ

  TITLE

  WEIGHT

  WHERE
- It omits these PROC REPORT statement options:

  LIST

  OUT=

  OUTREPT=

  PROFILE=

  REPORT=

  WINDOWS|NOWINDOWS
- It omits SAS system options.
- It resolves automatic macro variables.

Restriction: This option has no effect in the interactive report window environment. In the interactive report window environment, you can write the report definition for the report that is currently in the REPORT window to the SOURCE window by selecting **Tools ▶ Report Statements**

LS=*line-size*

specifies the length of a line of the report.

PROC REPORT honors the line size specifications that it finds in the following order of precedence:

- the LS= option in the PROC REPORT statement or LINESIZE= in the ROPTIONS window
- the LS= setting stored in the report definition loaded with REPORT= in the PROC REPORT statement
- the SAS system option LINESIZE=.

Note: The PROC REPORT LS= option takes precedence over all other line size options. △

Range: 64-256 (integer)

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Featured in: Example 6 on page 1103 and Example 8 on page 1108

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: The missing values section in *SAS Language Reference: Concepts*.

Featured in: Example 11 on page 1117

NAMED
writes *name*= in front of each value in the report, where *name* is the column heading for the value.

Interaction: When you use the NAMED option, PROC REPORT automatically uses the NOHEADER option.

Tip: Use NAMED in conjunction with the WRAP option to produce a report that wraps all columns for a single row of the report onto consecutive lines rather than placing columns of a wide report on separate pages.

Featured in: Example 7 on page 1105

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 1008.

NOCOMPLETECOLS
See COMPLETECOLS | NOCOMPLETECOLS on page 1009.

NOCOMPLETEROWS
See COMPLETEROWS | NOCOMPLETEROWS on page 1009.

NOEXEC
suppresses the building of the report. Use NOEXEC with OUTREPT= to store a report definition in a catalog entry. Use NOEXEC with LIST and REPORT= to display a listing of the specified report definition.

Alias: NOEXECUTE

NOHEADER
suppresses column headings, including headings that span multiple columns.

When you suppress the display of column headings in the interactive report window environment, you cannot select any report items.

NOTHREADS
See THREADS | NOTHREADS on page 1020.

NOWINDOWS

Alias: NOWD

See WINDOWS | NOWINDOWS on page 1020.

OUT=*SAS-data-set*
names the output data set. If this data set does not exist, then PROC REPORT creates it. The data set contains one observation for each report row and one observation for each unique summary line. If you use both customized and default summaries at the same place in the report, then the output data set contains only one observation because the two summaries differ only in how they present the data. Information about customization (underlining, color, text, and so on) is not data and is not saved in the output data set.

The output data set contains one variable for each column of the report. PROC REPORT tries to use the name of the report item as the name of the corresponding variable in the output data set. However, it cannot perform this substitution if a data set variable is under or over an across variable or if a data set variable appears multiple times in the COLUMN statement without aliases. In these cases, the name of the variable is based on the column number (_C1_, _C2_, and so on).

Output data set variables that are derived from input data set variables retain the formats of their counterparts in the input data set. PROC REPORT derives labels for these variables from the corresponding column headings in the report unless the only item defining the column is an across variable. In that case, the variables have no label. If multiple items are stacked in a column, then the labels of the corresponding output data set variables come from the analysis variable in the column.

The output data set also contains a character variable named _BREAK_. If an observation in the output data set derives from a detail row in the report, then the value of _BREAK_ is missing or blank. If the observation derives from a summary line, then the value of _BREAK_ is the name of the break variable that is associated with the summary line, or **_RBREAK_**. If the observation derives from a COMPUTE BEFORE _PAGE_ or COMPUTE AFTER _PAGE_ statement, then the value of _BREAK_ is **_PAGE_**. Note, however, that for COMPUTE BEFORE _PAGE_ and COMPUTE AFTER _PAGE_, the **_PAGE_** value is written to the output data set only; it is not available as a value of the automatic variable _BREAK_ during execution of the procedure.

Tip: An output data set can be created by using an ODS OUTPUT statement. The data set created by ODS OUTPUT is the same as the one created by the OUT= option. Refer to the ODS OUTPUT statement in *SAS Output Delivery System: User's Guide*.

Featured in: Example 12 on page 1120 and Example 13 on page 1122

OUTREPT=*libref.catalog.entry*

stores in the specified catalog entry the REPORT definition that is defined by the PROC REPORT step that you submit. PROC REPORT assigns the entry a type of REPT.

The stored report definition might differ in these ways from the statements that you submit:

- It omits some statements that are not specific to the REPORT procedure, whether you submit them with the PROC REPORT step or whether they are already in effect when you submit the step. These statements include

  BY

  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: PROC REPORTversion 7 and later cannot read entries created with SAS versions before Version 7. △

Featured in: Example 7 on page 1105

PANELS=*number-of-panels*
specifies the number of panels on each page of the report. If the width of a report is less than half of the line size, then you can display the data in multiple sets of columns so that rows that would otherwise appear on multiple pages appear on the same page. Each set of columns is a *panel*. A familiar example of this type of report is a telephone book, which contains multiple panels of names and telephone numbers on a single page.

When PROC REPORT writes a multipanel report, it fills one panel before beginning the next.

The number of panels that fits on a page depends on the

- width of the panel
- space between panels
- line size.

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output. However, the COLUMNS= option in the ODS PRINTER, ODS PDF, and ODS RTF statements produces similar results. For details, see the 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 1017 and the discussion of LS= on page 1013.

Featured in: Example 8 on page 1108

PCTLDEF=
See QNTLDEF= on page 1018.

PROFILE=*libref.catalog*
identifies the report profile to use. A profile

- specifies the location of menus that define alternative menu bars and menus for the REPORT and COMPUTE windows.
- sets defaults for WINDOWS, PROMPT, and COMMAND.

PROC REPORT uses the entry REPORT.PROFILE in the catalog that you specify as your profile. If no such entry exists, or if you do not specify a profile, then PROC REPORT uses the entry REPORT.PROFILE in SASUSER.PROFILE. If you have no profile, then PROC REPORT uses default menus and the default settings of the options.

You create a profile from the PROFILE window while using PROC REPORT in an interactive report window environment. To create a profile

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 about report profiles, see "PROFILE" on page 1066.

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 1103 and Example 8 on page 1108

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 1108

QMARKERS=*number*
specifies the default number of markers to use for the P^2 estimation method. The number of markers controls the size of fixed memory space.

Default: The default value depends on which quantiles you request. For the median (P50), *number* is 7. For the quartiles (P25 and P75), *number* is 25. For the quantiles P1, P5, P10, P90, P95, or P99, *number* is 105. If you request several quantiles, then PROC REPORT uses the largest default value of *number*.

Range: any odd integer greater than 3

Tip: Increase the number of markers above the default settings to improve the accuracy of the estimates; you can reduce the number of markers to conserve computing resources.

QMETHOD=OS|P2
specifies the method that PROC REPORT uses to process the input data when it computes quantiles. If the number of observations is less than or equal to the value of the QMARKERS= option, and the value of the QNTLDEF= option is 5, then both methods produce the same results.

OS
uses order statistics. PROC UNIVARIATE uses this technique.

Note: This technique can be very memory intensive. △

P2
uses the P^2 method to approximate the quantile.

Default: OS

Restriction: When QMETHOD=P2, PROC REPORT will not compute the following:

- MODE
- weighted quantiles

Tip: When QMETHOD=P2, reliable estimates of some quantiles (P1, P5, P95, P99) might not be possible for some data sets such as data sets with heavily tailed or skewed distributions.

QNTLDEF=1|2|3|4|5
specifies the mathematical definition that the procedure uses to calculate quantiles when the value of the QMETHOD= option is OS. When QMETHOD=P2, you must use QNTLDEF=5.

Default: 5

Alias: PCTLDEF=

Main discussion: "Quantile and Related Statistics" on page 1541

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 1015

Featured in: Example 7 on page 1105

SHOWALL
overrides options in the DEFINE statement that suppress the display of a column.

See also: NOPRINT and NOZERO in "DEFINE Statement" on page 1035

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 1090

SPANROWS
specifies that when the value of a GROUP or ORDER column is the same in multiple rows, the value will be displayed in a single cell that occupies that column in all the rows for which the value is the same. A box is essentially created for that part of the column, and no rows appear in that box.

Restriction: This option is supported only for the ODS MARKUP, RTF, PRINTER, and HTML destinations. It has no effect on the Report Window, the listing, or the data set destinations.

Tip: The SPANROWS option also allows GROUP and ORDER variables values to repeat when the values break across pages in PDF, PS, and RTF destinations.

Tip: If a summary row appears in the middle of a set of rows that would otherwise be spanned by a single cell, the summary row introduces its own cell in that column. This action breaks the spanning cell into two cells even when the value of the GROUP or ORDER variable that comes after the summary row is unchanged.

Tip: In order to produce PROC REPORT output that is compliant with section 508, you need to specify the SPANROWS option in PROC REPORT and specify the HEADER_DATA_ASSOCIATIONS=yes OPTIONS option in the HTML statement. Section 508 is the accessibility standards for electronic information technology adopted by the U.S. Government under Section 508 of the U.S. Rehabilitation Act of 1973.

Here is example code:

```
ods html file="sec508.html" options(header_data_associations="yes");
proc report data=energy headline headskip nowd spanrows;
```

SPLIT='*character*'
specifies the split character. PROC REPORT breaks a column heading when it reaches that character and continues the heading on the next line. The split character itself is not part of the column heading although each occurrence of the split character counts toward the 256-character maximum for a label.

Default: slash (/)

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output. However, in these ODS destinations, the SPLIT= option works in the column heading.

Interaction: The FLOW option in the DEFINE statement honors the split character.

Featured in: Example 5 on page 1099

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 997 for details.

Restriction: This option affects only the HTML, RTF, and Printer output.

Tip: FONT names that contain characters other than letters or underscores must be enclosed by quotation marks.

Featured in: Example 15 on page 1129 and Example 16 on page 1134

THREADS | NOTHREADS

enables or disables parallel processing of the input data set. This option overrides the SAS system option THREADS | NOTHREADS unless the system option is restricted. (See Restriction.) See "Support For Parallel Processing" in *SAS Language Reference: Concepts* for more information.

Default: value of SAS system option THREADS | NOTHREADS.

Restriction: Your site administrator can create a restricted options table. A restricted options table specifies SAS system option values that are established at startup and cannot be overridden. If the THREADS | NOTHREADS system option is listed in the restricted options table, any attempt to set these system options is ignored and a warning message is written to the SAS log.

Interaction: PROC REPORT uses the value of the SAS system option THREADS except when a BY statement is specified or the value of the SAS system option CPUCOUNT is less than 2. You can specify the THREADS option in the PROC REPORT statement to force PROC REPORT to use parallel processing in these situations.

Note: When multi-threaded processing, also known as parallel processing, is in effect, observations might be returned in an unpredictable order. However, the observations are sorted correctly when a BY statement is specified. △

VARDEF=*divisor*

specifies the divisor to use in the calculation of the variance and standard deviation. The following table shows the possible values for *divisor* and associated divisors.

Table 51.6 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 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 41

WINDOWS | NOWINDOWS

selects an interactive report window or nonwindowing environment.

When you use WINDOWS, SAS opens the REPORT window for the interactive report interface, which enables you to modify a report repeatedly and to see the modifications immediately. When you use NOWINDOWS, PROC REPORT runs without the REPORT window and sends its output to the open output destinations.

Alias: WD|NOWD

Restriction: When you use the WINDOWS option, you can send the output only to a SAS data set or to a Printer destination.

Tip: You can store a setting of WINDOWS in your report profile, if you have one. If you do not specify WINDOWS or NOWINDOWS in the PROC REPORT statement, then the procedure uses the setting in your report profile. If you do not have a report profile, then PROC REPORT looks at the setting of the SAS system option DMS. If DMS is ON, then PROC REPORT uses the interactive report window environment. If DMS is OFF, then it uses the nonwindowing environment.

See also: For a discussion of the report profile see the discussion of PROFILE= on page 1016.

Featured in: Example 1 on page 1087

WRAP
displays one value from each column of the report, on consecutive lines if necessary, before displaying another value from the first column. By default, PROC REPORT displays values for only as many columns as it can fit on one page. It fills a page with values for these columns before starting to display values for the remaining columns on the next page.

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Interaction: When WRAP is in effect, PROC REPORT ignores PAGE in any item definitions.

Tip: Typically, you use WRAP in conjunction with the NAMED option in order to avoid wrapping column headings.

Featured in: Example 7 on page 1105

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 1097

Example 5 on page 1099

BREAK *location break-variable</ option(s)>*;

Tasks	Option
Specify the color of the break lines in the REPORT window	COLOR=
Specify the link text used in the Table of Contents	CONTENTS=
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 interactive report window environment only.

Note: Not all operating environments and devices support all colors, and on some operating systems and devices, one color might map to another color. For example, if the DEFINITION window displays the word BROWN in yellow characters, then selecting BROWN results in a yellow item. △

CONTENTS='*link-text*'
specifies the text for the entries in the HTML contents file or PDF table of contents for the output that is produced by PROC REPORT. If the PAGE= option and the CONTENTS= option with *link-text* is specified, PROC REPORT uses the value of *link-text* as a link for tables created in the Table of Contents.

For information about HTML and PDF output, see "Output Delivery System" on page 33.

Default: If a BREAK BEFORE statement is present and the PAGE option is specified but no CONTENTS= option is specified, the default link text will be the location variable plus the value of the location variable. The location variable is associated with the BREAK variable. The value is the BREAK variable value. As shown in the following code, the value is rep and the location is before rep.

```
break before rep / summarize page;
```

If the BREAK AFTER statement does not have a CONTENTS= option specified, but does have the PAGE option specified, the default link text in the TOC is "Table N" where N is an integer.

Restriction: For HTML output, the CONTENTS= option has no effect in the HTML body file. It affects only the HTML contents file.

Restriction: If CONTENTS= is specified, but no PAGE option is specified, PROC REPORT generates a warning message in the SAS Log file.

Interaction: If the DEFINE statement has a page option and there is a BREAK BEFORE statement with a PAGE option and the CONTENTS= option has a value other than empty quotes specified, PROC REPORT adds a directory to the TOC and puts links to the tables in that directory. See the CONTENTS= option in the DEFINE statement"DEFINE Statement" on page 1035 for more information about this interaction.

Interaction: If there is a BREAK BEFORE statement specified and a CONTENTS=' 'option and a PAGE= option specified, PROC REPORT does not create a directory in the TOC. Instead, PROC REPORT uses the CONTENTS= value from the DEFINE statement to create links to the TOC. If there is no CONTENTS= option in the DEFINE statement, PROC REPORT creates links using the default text described in the DEFINE statement. Refer to the "DEFINE Statement" on page 1035 CONTENTS= option for an explanation of the default text information. .

Interaction: For RTF output, the CONTENTS= option has no effect on the RTF body file unless you turn on the CONTENTS=YES option in the ODS RTF statement. In that case, a Table of Contents page is inserted at the front of your RTF output file. Your CONTENTS= option text from PROC REPORT will then show up in this separate Table of Contents page.

Tip: If the CONTENTS= option is specified where the value is empty quotation marks, no table link will be created in the Table of Contents. An example of this code is

```
CONTENTS=' '
```

Tip: If there are multiple BREAK BEFORE statements, the link text is the concatenation of all of the CONTENTS= values or of all the default values.

DOL
(for double overlining) uses the 13th formatting character to overline each value

- that appears in the summary line
- that would appear in the summary line if you specified the SUMMARIZE option.

Default: equal sign (=)

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Interaction: If you specify both the OL and DOL options, then PROC REPORT honors only OL.

See also: the discussion of FORMCHAR= on page 1010.

DUL
(for double underlining) uses the 13th formatting character to underline each value

- that appears in the summary line
- that would appear in the summary line if you specified the SUMMARIZE option.

Default: equal sign (=)

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Interaction: If you specify both the UL and DUL options, then PROC REPORT honors only UL.

See also: the discussion of FORMCHAR= on page 1010.

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 1010.

Featured in: Example 2 on page 1090 and Example 9 on page 1110

PAGE
in monospace output, starts a new page. In HTML and PRINTER destinations, the PAGE option starts a new table.

Restriction In the OUTPUT destination, this option has no effect.

Interaction: If you use PAGE in the BREAK statement and you create a break at the end of the report, then the summary for the whole report appears on a separate page.

Featured in: Example 9 on page 1110

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 1090, Example 4 on page 1097, Example 5 on page 1099, and Example 8 on page 1108

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 997 for details.

Restriction: This option affects only the HTML, RTF, and Printer output.

Tip: FONT names that contain characters other than letters or underscores must be enclosed by quotation marks.

SUMMARIZE
writes a summary line in each group of break lines. A summary line for a set of observations contains values for

- the break variable (which you can suppress with the SUPPRESS option)
- other group or order variables to the left of the break variable
- statistics
- analysis variables
- computed variables.

The following table shows how PROC REPORT calculates the value for each type of report item in a summary line that is created by the BREAK statement:

Report Item	Value
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*
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 1032.)

* 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 1090, Example 4 on page 1097, and Example 9 on page 1110

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 1032.)

Featured in: Example 4 on page 1097

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 1010.

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 1032 and "LINE Statement" on page 1046. △

BY Statement

Creates a separate report on a separate page for each BY group.

Restriction: If you use the BY statement, then you must use the NOWINDOWS option in the PROC REPORT statement.

Interaction: If you use the RBREAK statement in a report that uses BY processing, then PROC REPORT creates a default summary for each BY group. In this case, you cannot summarize information for the whole report.

Tip: Using the BY statement does not make the FIRST. and LAST. variables available in compute blocks.

Main discussion: "BY" on page 36

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. For example, the data are grouped in chronological order.

The requirement for ordering or indexing observations according to the values of BY variables is suspended for BY-group processing when you use the NOTSORTED option. In fact, the procedure does not use an index if you specify NOTSORTED. The procedure defines a BY group as a set of contiguous observations that have the same values for all BY variables. If observations with the same values for the BY variables are not contiguous, then the procedure treats each contiguous set as a separate BY group.

CALL DEFINE Statement

Sets the value of an attribute for a particular column in the current row.

Restriction: Valid only in a compute block that is attached to a report item.

Featured in: Example 4 on page 1097

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 an interactive report window environment. Only the FORMAT, URL, URLBP, and URLP attributes have an effect in the nonwindowing environment. In fact, URL, URLBP, and URLP are effective only in the nonwindowing environment. The STYLE= and URL attributes are effective only when you are using the Output Delivery System to create HTML, RTF, or Printer output. (See Table 51.7 on page 1028 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 51.7 on page 1028.

ROW
is an automatic variable that indicates the entire current row.

value
sets the value for the attribute. For values for each attribute, refer to the following table.

Table 51.7 Attribute Descriptions

Attribute	Description	Values	Affects
BLINK	Controls blinking of current value	1 turns blinking on; 0 turns it off	interactive report window environment
COLOR	Controls the color of the current value in the REPORT window	'blue', 'red', 'pink', 'green', 'cyan', 'yellow', 'white', 'orange', 'black', 'magenta', 'gray', 'brown'	interactive report window environment

Attribute	Description	Values	Affects
COMMAND	Specifies that a series of commands follows	a quoted string of SAS commands to submit to the command line	interactive report window environment
FORMAT	Specifies a format for the column	a SAS format or a user-defined format	interactive report window and nonwindowing environments
HIGHLIGHT	Controls highlighting of the current value	1 turns highlighting on; 0 turns it off	interactive report window environment
RVSVIDEO	Controls display of the current value	1 turns reverse video on; 0 turns it off	interactive report window environment
URL	Makes the contents of each cell of the column a link to the specified Uniform Resource Locator (URL)	a quoted URL (either single or double quotation marks can be used)	HTML, RTF, and PDF output
URLBP	Makes the contents of each cell of the column a link. The link points to a Uniform Resource Locator that is a concatenation of **1** the string that is specified by the BASE= option in the ODS HTML statement **2** the string that is specified by the PATH= option in the ODS HTML statement **3** the value of the URLBP attribute[*,#]	a quoted URL (either single or double quotation marks can be used)	HTML output
URLP	Makes the contents of each cell of the column a link. The link points to a Uniform Resource Locator that is a concatenation of **1** the string that is specified by the PATH= option in the ODS HTML statement **2** the value of the URLP attribute[*,#]	a quoted URL (either single or double quotation marks can be used)	HTML output

For information about the BASE= and PATH= options, see the documentation for the ODS HTML Statement in *SAS Output Delivery System: User's Guide*.

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= value functions like the STYLE= option in other statements in PROC REPORT. However, instead of acting as an option in a statement, it becomes the value for the STYLE attribute. For example, the following CALL DEFINE statement sets the background color to yellow and the font size to 7 for the specified column:

```
call define(_col_, "style",
            "style=[backgroundcolor=yellow fontsize=7]");
```

In SAS 9.2, the STYLE and STYLE/REPLACE attributes specify the style element to be used for the Output Delivery System. If a style already exists for this cell or row, these STYLE attributes tell CALL DEFINE to replace the style specified by the STYLE= value. The STYLE/MERGE attribute tells CALL DEFINE to merge the style specified by the STYLE= value with the existing style attributes that are in the same cell or row. If there is no previously existing STYLE= value to merge, STYLE/MERGE acts the same as the STYLE or STYLE/REPLACE attributes. See "Using Style Elements in PROC REPORT" on page 997 for more details.

Restriction: This option affects only the HTML, RTF, Printer destinations.

Interaction: If you set a style element for the CALLDEF location in the PROC REPORT statement and you want to use that exact style element in a CALL DEFINE statement, then use an empty string as the value for the STYLE attribute, as shown here:

```
call define (_col_, "STYLE", "" );
```

Tip: FONT names that contain characters other than letters or underscores must be enclosed by quotation marks.

Featured in: Example 16 on page 1134

COLUMN Statement

Describes the arrangement of all columns and of headings that span more than one column.

Restriction: You cannot use the COLUMN statement if you use REPORT= in the PROC REPORT statement.

Featured in:

Example 1 on page 1087
Example 3 on page 1093
Example 5 on page 1099
Example 6 on page 1103
Example 10 on page 1114
Example 11 on page 1117

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 991 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 1087 and Example 11 on page 1117

report-item-1, *report-item-2* <. . . , *report-item-n*>
identifies report items that collectively determine the contents of the column or columns. These items are said to be stacked in the report because each item generates a heading, and the headings are stacked one above the other. The heading for the leftmost item is on top. If one of the items is an analysis variable, a computed variable, a group variable, or a statistic, then its values fill the cells in that part of the report. Otherwise, PROC REPORT fills the cells with frequency counts.

If you stack a statistic with an analysis variable, then the statistic that you name in the column statement overrides the statistic in the definition of the analysis variable. For example, the following PROC REPORT step produces a report that contains the minimum value of Sales for each sector:

```
proc report data=grocery;
   column sector sales,min;
   define sector/group;
   define sales/analysis sum;
run;
```

If you stack a display variable under an across variable, then all the values of that display variable appear in the report.

Interaction: A series of stacked report items can include only one analysis variable or statistic. If you include more than one analysis variable or statistic, then PROC REPORT returns an error because it cannot determine which values to put in the cells of the report.

Tip: You can use parentheses to group report items whose headings should appear at the same level rather than stacked one above the other.

Featured in: Example 5 on page 1099, Example 6 on page 1103, and Example 10 on page 1114

('*header-1* '<… '*header-n* '> *report-item(s)*)
creates one or more headings that span multiple columns.

header
is a string of characters that spans one or more columns in the report. PROC REPORT prints each heading on a separate line. You can use split characters in a heading to split one heading over multiple lines. See the discussion of SPLIT= on page 1019.

In traditional (monospace) SAS output, if the first and last characters of a heading are one of the following characters, then PROC REPORT uses that character to expand the heading to fill the space over the column or columns. Note that the <> and the >< must be paired.

```
- = . _ * + <> ><
```

Similarly, if the first character of a heading is < and the last character is >, or vice versa, then PROC REPORT expands the heading to fill the space over the column by repeating the first character before the text of the heading and the last character after it.

Note: The use of expanding characters is supported only in monospace destinations. Therefore, PROC REPORT simply removes the expanding characters when the output is directed to an ODS MARKUP, HTML, RTF, or PRINTER destination. Refer to Understanding ODS Destinations in *SAS Output Delivery System: User's Guide* for more information. △

report-item(s)
specifies the columns to span.

Featured in: Example 10 on page 1114

report-item=name
specifies an alias for a report item. You can use the same report item more than once in a COLUMN statement. However, you can use only one DEFINE statement for any given name. (The DEFINE statement designates characteristics such as formats and customized column headings. If you omit a DEFINE statement for an item, then the REPORT procedure uses defaults.) Assigning an alias in the COLUMN statement does not by itself alter the report. However, it does enable you to use separate DEFINE statements for each occurrence of a variable or statistic.

Featured in: Example 3 on page 1093

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 993.) △

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 1090
Example 3 on page 1093
Example 4 on page 1097
Example 5 on page 1099
Example 9 on page 1110
Example 10 on page 1114

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 992.

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 1075.)
- 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 1075.)
- 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 1093 and Example 9 on page 1110

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 1097 and Example 5 on page 1099

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 997 for details.

Restriction: This option affects only the HTML, RTF, and Printer destinations.

Tip: FONT names that contain characters other than letters or underscores must be enclosed by quotation marks.

Featured in: Example 16 on page 1134

target
controls when the compute block executes. If you specify a location (BEFORE or AFTER) for the COMPUTE statement, then you can also specify *target*, which can be one of the following:

break-variable
is a group or order variable.

When you specify a break variable, PROC REPORT executes the statements in the compute block each time the value of the break variable changes.

PAGE </ justification>
in monospace output destinations, causes the compute block to execute once for each page, either immediately after printing any titles or immediately before printing any footnotes. *justification* controls the placement of text and values. It can be one of the following:

CENTER	centers each line that the compute block writes.
LEFT	left-justifies each line that the compute block writes.
RIGHT	right-justifies each line that the compute block writes.

Default: CENTER

Featured in: Example 9 on page 1110

type-specification
specifies the type. (Optional) Also specifies the length of *report-item*. If the report item that is associated with a compute block is a computed variable, then PROC REPORT assumes that it is a numeric variable unless you use a type specification to specify that it is a character variable. A type specification has the form

CHARACTER <LENGTH=*length*>

where

CHARACTER
specifies that the computed variable is a character variable. If you do not specify a length, then the variable's length is 8.

Alias: CHAR

Featured in: Example 10 on page 1114

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 1114

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 1090
Example 3 on page 1093
Example 4 on page 1097
Example 5 on page 1099
Example 6 on page 1103
Example 9 on page 1110
Example 10 on page 1114

DEFINE *report-item* / <*option(s)*>;

Task	Option
Specify how to use a report item. (sSee "Usage of Variables in a Report" on page 987.)	
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
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=

Task	Option
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	
Create a link in the Table of Contents	CONTENTS=
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 headings	
Center the formatted values of the report item within the column width and center the column heading over the values	CENTER
Left-justify the formatted values of the report item within the column width and left-justify the column headings over the values	LEFT
Right-justify the formatted values of the report item within the column width and right-justify the column headings over the values	RIGHT
Specify the color in the REPORT window of the column heading and of the values of the item that you define	COLOR=
Define the column heading for the report item	*column-heading*

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 989.)

Featured in: Example 5 on page 1099

ANALYSIS
defines *report-item*, which must be a data set variable, as an analysis variable. (See "Analysis Variables" on page 988.)

By default, PROC REPORT calculates the Sum statistic for an analysis variable. Specify an alternate statistic with the *statistic* option in the DEFINE statement.

Note: Naming a statistic in the DEFINE statement implies the ANALYSIS option, so you never need to specify ANALYSIS. However, specifying ANALYSIS can make your code easier for novice users to understand. △

Note: Special missing values show up as missing values when they are defined as ANALYSIS variables. △

Featured in: Example 2 on page 1090, Example 3 on page 1093, and Example 4 on page 1097

CENTER
centers the formatted values of the report item within the column width and centers the column heading over the values. This option has no effect on the CENTER option in the PROC REPORT statement, which centers the report on the page.

COLOR=*color*
specifies the color in the REPORT window of the column heading and of the values of the item that you are defining. You can use the following colors:

BLACK	MAGENTA
BLUE	ORANGE
BROWN	PINK
CYAN	RED
GRAY	WHITE
GREEN	YELLOW

Default: The color of **`Foreground`** in the SASCOLOR window. (For more information, see the online Help for the SASCOLOR window.)

Restriction: This option affects output in the interactive report window environment only.

Note: Not all operating environments and devices support all colors, and in some operating environments and devices, one color might map to another color. For example, if the DEFINITION window displays the word BROWN in yellow characters, then selecting BROWN results in a yellow item. △

column-header
defines the column heading for the report item. Enclose each heading in single or double quotation marks. When you specify multiple column headings, PROC REPORT uses a separate line for each one. The split character also splits a column heading over multiple lines.

In traditional (monospace) SAS output, if the first and last characters of a heading are one of the following characters, then PROC REPORT uses that character to expand the heading to fill the space over the column:

```
:– = \_ .* +
```

Similarly, if the first character of a heading is < and the last character is >, or vice versa, then PROC REPORT expands the heading to fill the space over the column by repeating the first character before the text of the heading and the last character after it.

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 headings and the spaces between them. In traditional (monospace) SAS output, you can underline column headings without underlining the spaces between them, by using the special characters `'--'` as the last line of each column heading instead of using HEADLINE. (See Example 4 on page 1097.)

See also: SPLIT= on page 1019

Featured in: Example 3 on page 1093, Example 4 on page 1097, and Example 5 on page 1099

COMPUTED
defines the specified item as a computed variable. Computed variables are variables that you define for the report. They are not in the input data set, and PROC REPORT does not add them to the input data set.

In the interactive report window environment, you add a computed variable to a report from the COMPUTED VAR window.

In the nonwindowing environment, you add a computed variable by

- including the computed variable in the COLUMN statement
- defining the variable's usage as COMPUTED in the DEFINE statement
- computing the value of the variable in a compute block associated with the variable.

Featured in: Example 5 on page 1099 and Example 10 on page 1114

CONTENTS='*link-text*'
specifies the text for the entries in the HTML contents file or PDF table of contents for the output that is produced by PROC REPORT. If the DEFINE statement has the

PAGE= option and the CONTENTS= option specified with a *link-text* value assigned, PROC REPORT adds a directory to the TOC and uses the value of *link-text* as a link for tables created in the Table of Contents.

For information about HTML and PDF output, see "Output Delivery System" on page 33.

Default: If the DEFINE statement has a PAGE option but does not have a CONTENTS= option specified, a directory is created with the directory text as **COLA---COLB**. COLA is the name or alias of the leftmost column and COLB is the name or alias of the rightmost column. If the table has only one column, the directory text is the column name or alias.

Restriction: For HTML output, the CONTENTS= option has no effect in the HTML body file. It affects only the HTML contents file.

Restriction: If CONTENTS= is specified, but no PAGE option is specified, PROC REPORT generates a warning message in the SAS log file.

Interaction: If the DEFINE statement has a page option and there is a BREAK BEFORE statement with a PAGE option and the CONTENTS= option specified has a value other than empty quotation marks, PROC REPORT adds a directory to the TOC and puts links to the tables in that directory.

Interaction: If the DEFINE statement has a PAGE option and there is a BREAK BEFORE statement with no PAGE option, PROC REPORT does not create a directory in the TOC. Instead, PROC REPORT uses the CONTENTS= value from the DEFINE statement to create links to the TOC. If there is no CONTENTS= option in the DEFINE statement, PROC REPORT creates links using the default text **COLA---COLB**. Refer to the Default explanation above.

Interaction: If there is a BREAK BEFORE statement with a CONTENTS=' 'option specified and a PAGE option specified, PROC REPORT does not create a directory in the TOC. Instead, PROC REPORT uses the CONTENTS= value from the DEFINE statement to create links to the TOC. If there is no CONTENTS= option in the DEFINE statement, PROC REPORT creates links using the default text **COLA---COLB**. Refer to the Default explanation above.

Interaction: For RTF output, the CONTENTS= option has no effect on the RTF body file unless you turn on the CONTENTS=YES option in the ODS RTF statement. In that case, a Table of Contents page is inserted at the front of your RTF output file. Your CONTENTS= option text from PROC REPORT will then show up in this separate Table of Contents page.

Tip: If the DEFINE statement has the CONTENTS= option specified where the value is empty quotation marks, the directory to the TOC is not added. An example of this code is as follows:

```
CONTENTS=' '
```

Tip: If there are multiple BREAK BEFORE statements, the link text is the concatenation of all of the CONTENTS= values or of all the default values.

DESCENDING

reverses the order in which PROC REPORT displays rows or values of a group, order, or across variable.

Tip: By default, PROC REPORT orders group, order, and across variables by their formatted values. Use the ORDER= option in the DEFINE statement to specify an alternate sort order.

DISPLAY

defines *report-item*, which must be a data set variable, as a display variable. (See "Display Variables" on page 987.)

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 1114

FORMAT=*format*
assigns a SAS or user-defined format to the item. This format applies to *report-item* as PROC REPORT displays it; the format does not alter the format associated with a variable in the data set. For data set variables, PROC REPORT honors the first of these formats that it finds:

- the format that is assigned with FORMAT= in the DEFINE statement
- the format that is assigned in a FORMAT statement when you invoke PROC REPORT
- the format that is associated with the variable in the data set.

If none of these formats is present, then PROC REPORT uses BEST*w*. for numeric variables and $*w*. for character variables. The value of *w* is the default column width. For character variables in the input data set, the default column width is the variable's length. For numeric variables in the input data set and for computed variables (both numeric and character), the default column width is the value specified by COLWIDTH= in the PROC REPORT statement or in the ROPTIONS window.

In the interactive report window environment, if you are unsure what format to use, then type a question mark (?) in the format field in the DEFINITION window to access the FORMATS window.

Alias: F=

Featured in: Example 2 on page 1090 and Example 6 on page 1103

GROUP
defines *report-item*, which must be a data set variable, as a group variable. (See "Group Variables" on page 988.)

Featured in: Example 4 on page 1097, Example 6 on page 1103, and Example 14 on page 1126

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 1103

ITEMHELP=*entry-name*
references a HELP or CBT entry that contains help information for the report item. Use PROC BUILD in SAS/AF software to create a HELP or CBT entry for a report item. All HELP and CBT entries for a report must be in the same catalog, and you

must specify that catalog with the HELP= option in the PROC REPORT statement or from the **User Help** fields in the ROPTIONS window.

Of course, you can access these entries only from an interactive report window environment. To access a Help entry from the report, select the item and issue the HELP command. PROC REPORT first searches for and displays an entry named *entry-name*.CBT. If no such entry exists, then PROC REPORT searches for *entry-name*.HELP. If neither a CBT nor a HELP entry for the selected item exists, then the opening frame of the Help for PROC REPORT is displayed.

LEFT
left-justifies the formatted values of the report item within the column width and left-justifies the column headings over the values. If the format width is the same as the width of the column, then the LEFT option has no effect on the placement of values.

Restriction This option only affects the LISTING output. It has no effect on other ODS output.

MISSING
considers missing values as valid values for the report item. Special missing values that represent numeric values (the letters A through Z and the underscore (_) character) are each considered as a separate value.

Default: If you omit the MISSING option, then PROC REPORT excludes from the report and the output data sets all observations that have a missing value for any group, order, or across variable.

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 1110.)

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 993.)

Interaction: SHOWALL in the PROC REPORT statement or the ROPTIONS window overrides all occurrences of NOPRINT.

Featured in: Example 3 on page 1093, Example 9 on page 1110, and Example 12 on page 1120

NOZERO
suppresses the display of the report item if its values are all zero or missing.

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

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 993.)

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 987.)

Featured in: Example 2 on page 1090

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 1090

Note: The default value for the ORDER= option in PROC REPORT is not the same as the default value in other SAS procedures. In other SAS procedures, the default is ORDER=INTERNAL. The default for the option in PROC REPORT might change in a future release to be consistent with other procedures. Therefore, in production jobs where it is important to order report items by their formatted values, specify ORDER=FORMATTED even though it is currently the default. Doing so ensures that PROC REPORT will continue to produce the reports you expect even if the default changes. △

PAGE
inserts a page break just before printing the first column containing values of the report item.

Restriction: This option has no affect on the OUTPUT destination.

Interaction: PAGE is ignored if you use WRAP in the PROC REPORT statement or in the ROPTIONS window.

Tip: In listing destinations, a PAGE option in the DEFINE statement causes PROC REPORT to print this column and all columns to its right on a new page. However, for ODS MARKUP, HTML, PRINTER , and RTF destinations, the page break does not occur until all the rows in the report have been printed. Therefore, PROC REPORT prints all the rows for all the columns to the left of the PAGE column and then starts over at the top of the report and prints the PAGE column and the columns to the right.

PRELOADFMT
specifies that the format is preloaded for the variable.

Restriction: PRELOADFMT applies only to group and across variables.

Requirement: PRELOADFMT has no effect unless you specify either EXCLUSIVE or ORDER=DATA and you assign a format to the variable.

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 headings over the values. If the format width is the same as the width of the column, then RIGHT has no effect on the placement of values.

Restriction This option only affects the LISTING output. It has no affect on other ODS output.

SPACING=*horizontal-positions*
defines the number of blank characters to leave between the column being defined and the column immediately to its left. For each column, the sum of its width and the blank characters between it and the column to its left cannot exceed the line size.

Default: 2

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

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 type of variable.

See "Statistics That Are Available in PROC REPORT" on page 991 for a list of available statistics.

Default: SUM

Featured in: Example 2 on page 1090, Example 3 on page 1093, and Example 4 on page 1097

Note: PROC REPORT uses the name of the analysis variable as the default heading for the column. You can customize the column heading with the *column-header* option in the DEFINE statement. △

STYLE<(*location(s)*)>=<*style-element-name*><[*style-attribute-specification(s)*]>
specifies the style element to use for column headings and for text inside cells for this report item. See "Using Style Elements in PROC REPORT" on page 997 for details.

Restriction: This option affects only the HTML, RTF, and Printer destinations.

Tip: FONT names that contain characters other than letters or underscores must be enclosed by quotation marks.

Featured in: Example 16 on page 1134

WEIGHT=*weight-variable*
specifies a numeric variable whose values weight the values of the analysis variable that is specified in the DEFINE statement. The variable value does not have to be an

integer. The following table describes how PROC REPORT treats various values of the WEIGHT variable.

Weight Value	PROC REPORT Response
0	counts the observation in the total number of observations
less than 0	converts the value to zero and counts the observation in the total number of observations
missing	excludes the observation

To exclude observations that contain negative and zero weights from the analysis, use the EXCLNPWGT option in the PROC REPORT statement. Note that most SAS/STAT procedures, such as PROC GLM, exclude negative and zero weights by default.

Alias: WGT=

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: Before 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 1114

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 1090

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 40

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 1090
Example 3 on page 1093
Example 9 on page 1110

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 993.

item-format
is a SAS format or user-defined format. You must specify a format for each item.

Featured in: Example 2 on page 1090

'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 1090

number-of-repetitions'character-string '*
specifies a character string and the number of times to repeat it.

Featured in: Example 3 on page 1093

pointer-control
specifies the column in which PROC REPORT displays the next specification. You can use either of the following forms for pointer controls:

@column-number
specifies the number of the column in which to begin displaying the next item in the specification list.

+column-increment
specifies the number of columns to skip before beginning to display the next item in the specification list.

Both *column-number* and *column-increment* can be either a variable or a literal value.

Restriction: The pointer controls are designed for monospace output. They have no effect in the HTML, RTF, or Printer output.

Featured in: Example 3 on page 1093 and Example 5 on page 1099

Differences between the LINE and PUT Statements

The LINE statement does not support the following features of the PUT statement:

- automatic labeling signaled by an equal sign (=), also known as named output
- the _ALL_, _INFILE_, and _PAGE_ arguments and the OVERPRINT option
- grouping items and formats to apply one format to a list of items
- pointer control using expressions
- line pointer controls (# and /)
- trailing at signs (@ and @@)
- format modifiers
- array elements.

RBREAK Statement

Produces a default summary at the beginning or end of a report or at the beginning or end of each BY group.

Featured in:
Example 1 on page 1087
Example 10 on page 1114

RBREAK *location* </ *option(s)*>;

Task	Option
Specify the color of the break lines in the REPORT window	COLOR=
Specifies the link text used in the Table of Contents	CONTENTS=
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

Task	Option
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 interactive report window environment only.

Note: Not all operating environments and devices support all colors, and in some operating environments and devices, one color might map to another color. For example, if the DEFINITION window displays the word BROWN in yellow characters, then selecting BROWN results in a yellow item. △

CONTENTS='*link-text*'
specifies the text for the entries in the HTML contents file or PDF table of contents for the output that is produced by PROC REPORT. Only the RBREAK BEFORE statement with the PAGE and SUMMARIZE options specified creates a table within the TOC. If the CONTENTS= option plus the PAGE and SUMMARIZE options are specified, PROC REPORT uses the value of *link-text* and places that text in the Table of Contents for the tables created. If the value of CONTENTS= is empty quotation marks, no link is created in the TOC.

For information about HTML and PDF output, see "Output Delivery System" on page 33.

Default: If an RBREAK BEFORE statement is present and the PAGE and SUMMARIZE options are specified but no CONTENTS= option is specified, the default link text in the TOC will show **`Summary`**.

Restriction: For HTML output, the CONTENTS= option has no effect in the HTML body file. It affects only the HTML contents file.

Restriction: If CONTENTS= is specified, but no PAGE option is specified, PROC REPORT generates a warning message in the SAS log file. Only RBREAK BEFORE / with the SUMMARIZE and PAGE options specified can actually create a table in the TOC.

Interaction: For RTF output, the CONTENTS= option has no effect on the RTF body file unless you turn on the CONTENTS=YES option in the ODS RTF statement. In that case, a Table of Contents page is inserted at the front of your RTF output file. Your CONTENTS= option text from PROC REPORT will then show up in this separate Table of Contents page.

Tip: HTML output can now have additional anchor tags.

DOL
(for double overlining) uses the 13th formatting character to overline each value

- that appears in the summary line
- that would appear in the summary line if you specified the SUMMARIZE option.

Default: equal sign (=)

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Interaction: If you specify both the OL and DOL options, then PROC REPORT honors only OL.

See also: the discussion of FORMCHAR= on page 1010.

Featured in: Example 1 on page 1087

DUL
(for double underlining) uses the 13th formatting character to underline each value

- that appears in the summary line
- that would appear in the summary line if you specified the SUMMARIZE option.

Default: equal sign (=)

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Interaction: If you specify both the UL and DUL options, then PROC REPORT honors only UL.

See also: the discussion of FORMCHAR= on page 1010.

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 1010.

Featured in: Example 10 on page 1114

PAGE
starts a new page after the last break line of a break located at the beginning of the report. On RBREAK BEFORE, the PAGE option starts a new table.

Restriction: This option has no affect on the OUTPUT destination.

SKIP
writes a blank line after the last break line of a break located at the beginning of the report.

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

STYLE<(*location(s)*)>=<*style-element-name*><[*style-attribute-specification(s)*]>
specifies the style element to use for default summary lines that are created with the RBREAK statement. See "Using Style Elements in PROC REPORT" on page 997 for details.

Restriction: This option affects only the HTML, RTF, and Printer destinations.

Tip: FONT names that contain characters other than letters or underscores must be enclosed by quotation marks.

SUMMARIZE
includes a summary line as one of the break lines. A summary line at the beginning or end of a report contains values for

- statistics
- analysis variables
- computed variables.

The following table shows how PROC REPORT calculates the value for each type of report item in a summary line created by the RBREAK statement:

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 1032.)

Featured in: Example 1 on page 1087 and Example 10 on page 1114

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 1010.

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 1032 and "LINE Statement" on page 1046. △

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 42. For an example that uses the WEIGHT statement, see "Weighted Statistics Example" on page 43.

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.

Restriction: PROC REPORT will not compute MODE when a weight variable is active. Instead, try using PROC UNIVARIATE when MODE needs to be computed and a weight variable is active.

Tip: When you use the WEIGHT statement, consider which value of the VARDEF= option is appropriate. See VARDEF= on page 1020 and the calculation of weighted statistics in "Keywords and Formulas" on page 1536 for more information.

Note: Before 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 interactive report window environment in PROC REPORT provides essentially the same functionality as the statements, with one major exception: you cannot use the Output Delivery System from the interactive report window environment.

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

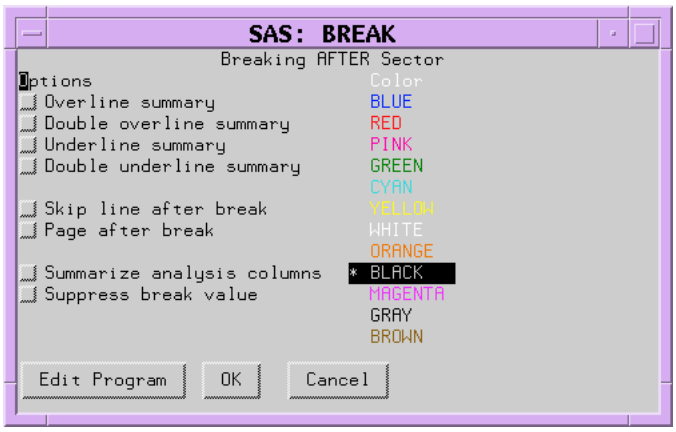
After you select **`Summarize Information`**, PROC REPORT offers you four choices for the location of the break:

- **`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 1010. △

Options

Overline summary
uses the second formatting character to overline each value

- that appears in the summary line
- that would appear in the summary line if you specified the SUMMARIZE option.

Default: hyphen (-)

Interaction: If you specify options to overline and to double overline, then PROC REPORT overlines.

Double overline summary
uses the 13th formatting character to overline each value

- that appears in the summary line
- that would appear in the summary line if you specified the SUMMARIZE option.

Default: equal sign (=)

Interaction: If you specify options to overline and to double overline, then PROC REPORT overlines.

Underline summary
uses the second formatting character to underline each value

- that appears in the summary line
- that would appear in the summary line if you specified the SUMMARIZE option.

Default: hyphen (-)

Interaction: If you specify options to underline and to double underline, then PROC REPORT underlines.

Double underline summary
uses the 13th formatting character to underline each value

- that appears in the summary line
- that would appear in the summary line if you specified the SUMMARIZE option.

Default: equal sign (=)

Interaction: If you specify options to underline and to double underline, then PROC REPORT underlines.

Skip line after break
writes a blank line for the last break line.
This option has no effect if you use it in a break at the end of a report.

Page after break
starts a new page after the last break line. This option has no effect in a break at the end of a report.

Interaction: If you use this option in a break on a variable and you create a break at the end of the report, then the summary for the whole report is on a separate page.

Summarize analysis columns
writes a summary line in each group of break lines. A summary line contains values for

- statistics
- analysis variables
- computed variables.

A summary line between sets of observations also contains

- the break variable (which you can suppress with **`Suppress break value`**)
- other group or order variables to the left of the break variable.

The following table shows how PROC REPORT calculates the value for each type of report item in a summary line created by the BREAK window:

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.

If the report item is...	Then its value is...
a computed variable	the results of the calculations based on the code in the corresponding compute block. (See "COMPUTE Statement" on page 1032.)

*If you reference a variable with a missing value in a customized summary line, then PROC REPORT displays that variable as a blank (for character variables) or a period (for numeric variables).

Suppress break value
suppresses printing of

- the value of the break variable in the summary line
- any underlining and overlining in the break lines in the column containing the break variable.

If you select **`Suppress break value`**, then the value of the break variable is unavailable for use in customized break lines unless you assign it a value in the compute block that is associated with the break.

Color

From the list of colors, select the one to use in the REPORT window for the column heading and the values of the item that you are defining.

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 might map to another color. For example, if the DEFINITION window displays the word BROWN in yellow characters, then selecting BROWN results in a yellow item.

Buttons

Edit Program
opens the COMPUTE window and enables you to associate a compute block with a location in the report.

OK
applies the information in the BREAK window to the report and closes the window.

Cancel
closes the BREAK window without applying information to the report.

COMPUTE

Attaches a compute block to a report item or to a location in the report. Use the SAS Text Editor commands to manipulate text in this window.

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 992.

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

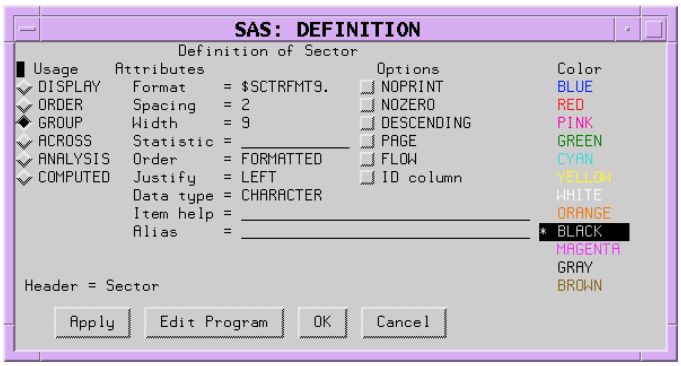
Displays the characteristics associated with an item in the report and lets you change them.

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 986.

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 1059) for an analysis variable. ANALYSIS is the default for numeric variables.

COMPUTED
defines the selected item as a computed variable. Computed variables are variables that you define for the report. They are not in the input data set, and PROC REPORT does not add them to the input data set. However, computed variables are included in an output data set if you create one.

In the interactive report window environment, you add a computed variable to a report from the COMPUTED VAR window.

Attributes

Format=
assigns a SAS or user-defined format to the item. This format applies to the selected item as PROC REPORT displays it; the format does not alter the format that is associated with a variable in the data set. For data set variables, PROC REPORT honors the first of these formats that it finds:

- the format that is assigned with FORMAT= in the DEFINITION window
- the format that is assigned in a FORMAT statement when you start PROC REPORT
- the format that is associated with the variable in the data set.

If none of these formats is present, then PROC REPORT uses BEST*w*. for numeric variables and $*w*. for character variables. The value of *w* is the default column width. For character variables in the input data set, the default column width is the variable's length. For numeric variables in the input data set and for computed variables (both numeric and character), the default column width is the value of the COLWIDTH= attribute in the ROPTIONS window.

If you are unsure what format to use, then type a question mark (?) in the format field in the DEFINITION window to access the FORMATS window.

Spacing=
defines the number of blank characters to leave between the column being defined and the column immediately to its left. For each column, the sum of its width and the blank characters between it and the column to its left cannot exceed the line size.

Default: 2

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 type of variable.

Default: SUM

Note: PROC REPORT uses the name of the analysis variable as the default heading for the column. You can customize the column heading with the **`Header`** field of the DEFINITION window. △

You can use the following values for *statistic*:

Descriptive statistic keywords

CSS	PCTSUM
CV	RANGE
MAX	STD
MEAN	STDERR
MIN	SUM
N	SUMWGT

NMISS	USS
PCTN	VAR
Quantile statistic keywords	
MEDIAN | P50	Q3 | P75
P1	P90
P5	P95
P10	P99
Q1 | P25	QRANGE
Hypothesis testing keyword	
PRT|PROBT	T

Explanations of the keywords, the formulas that are used to calculate them, and the data requirements are discussed in Appendix 1, "SAS Elementary Statistics Procedures," on page 1535.

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 1536.

Order=
orders the values of a GROUP, ORDER, or ACROSS variable according to the specified order, where

DATA
orders values according to their order in the input data set.

FORMATTED
orders values by their formatted (external) values. By default, the order is ascending.

FREQ
orders values by ascending frequency count.

INTERNAL
orders values by their unformatted values, which yields the same order that PROC SORT would yield. This order is operating environment-dependent. This sort sequence is particularly useful for displaying dates chronologically.

Default: FORMATTED

Interaction: DESCENDING in the item's definition reverses the sort sequence for an item.

Note: The default value for the ORDER= option in PROC REPORT is not the same as the default value in other SAS procedures. In other SAS procedures, the default is ORDER=INTERNAL. The default for the option in PROC REPORT might change in a future release to be consistent with other procedures. Therefore, in production jobs where it is important to order report items by their formatted values, specify ORDER=FORMATTED even though it is currently the default. Doing so ensures that PROC REPORT will continue to produce the reports you expect even if the default changes. △

Justify=
You can justify the placement of the column heading and of the values of the item that you are defining within a column in one of three ways:

LEFT
left-justifies the formatted values of the item that you are defining within the column width and left-justifies the column heading over the values. If the format width is the same as the width of the column, then LEFT has no effect on the placement of values.

RIGHT
right-justifies the formatted values of the item that you are defining within the column width and right-justifies the column heading over the values. If the format width is the same as the width of the column, then RIGHT has no effect on the placement of values.

CENTER
centers the formatted values of the item that you are defining within the column width and centers the column heading over the values. This option has no effect on the setting of the SAS system option CENTER.

When justifying values, PROC REPORT justifies the field width defined by the format of the item within the column. Thus, numbers are always aligned.

Data type=
shows you if the report item is numeric or character. You cannot change this field.

Item Help=
references a HELP or CBT entry that contains help information for the selected item. Use PROC BUILD in SAS/AF software to create a HELP or CBT entry for a report item. All HELP and CBT entries for a report must be in the same catalog, and you must specify that catalog with the HELP= option in the PROC REPORT statement or from the **`User Help`** fields in the ROPTIONS window.

To access a help entry from the report, select the item and issue the HELP command. PROC REPORT first searches for and displays an entry named *entry-name*.CBT. If no such entry exists, then PROC REPORT searches for *entry-name*.HELP. If neither a CBT nor a HELP entry for the selected item exists, then the opening frame of the help for PROC REPORT is displayed.

Alias=
By entering a name in the **`Alias`** field, you create an alias for the report item that you are defining. Aliases let you distinguish between different uses of the same report item. When you refer in a compute block to a report item that has an alias, you must use the alias. (See Example 3 on page 1093.)

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 1110.)

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 993.)

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 993.)

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 heading 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 might map to another color. For example, if the DEFINITION window displays the word BROWN in yellow characters, then selecting BROWN results in a yellow item.

Buttons

Apply
applies the information in the open window to the report and keeps the window open.

Edit Program
opens the COMPUTE window and enables you to associate a compute block with the variable that you are defining.

OK
applies the information in the DEFINITION window to the report and closes the window.

Cancel
closes the DEFINITION window without applying changes made with APPLY.

DISPLAY PAGE

Displays a particular page of the report.

Path

View ▶ Display Page

Description

You can access the last page of the report by entering a large number for the page number. When you are on the last page of the report, PROC REPORT sends a note to the message line of the REPORT window.

EXPLORE

Lets you experiment with your data.

Restriction: You cannot open the EXPLORE window unless your report contains at least one group or order variable.

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 1016.)

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 headings. To change other aspects of the report, you select a report item (for example, a column heading) as the target of the next command and issue the command. To select an item, use a mouse or cursor keys to position the cursor over it. Then click the mouse button or press ENTER. To execute a command, make a selection from the menu bar at the top of the REPORT window. PROC REPORT displays the effect of a command immediately unless the DEFER option is on.

Note: Issuing the END command in the REPORT window returns you to the previous report definition with the current data. If there is no previous report definition, then END closes the REPORT window. △

Note: In the REPORT window, there is no Save As option from the File menu to save your report to a file. Instead:

1 From the Report window, select Save Data Set. In the dialog box, enter a SAS library and filename in which to save this data set.
2 From the program editor window, execute a PROC PRINT.
3 In the File menu, select Save As to save the generated output to a file.

△

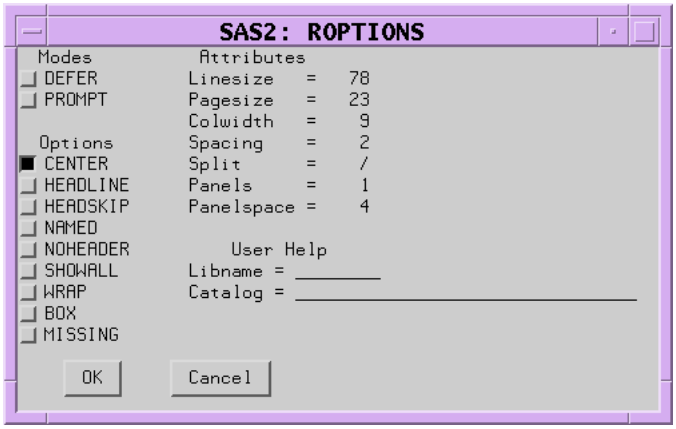
ROPTIONS

Displays choices that control the layout and display of the entire report and identifies the SAS library and catalog containing CBT or HELP entries for items in the report.

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 headings and the spaces between them at the top of each page of the report.

HEADLINE underlines with the second formatting character. (See the discussion of FORMCHAR= on page 1010.)

Default: hyphen (-)

Tip: In traditional (monospace) SAS output, you can underline column headings without underlining the spaces between them, by using `'--'` as the last line of each column heading instead of using HEADLINE.

HEADSKIP
writes a blank line beneath all column headings (or beneath the underlining that the HEADLINE option writes) at the top of each page of the report.

NAMED
writes *name*= in front of each value in the report, where *name* is the column heading for the value.

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 headings, including headings that span multiple columns.

Once you suppress the display of column headings in the interactive report window environment, you cannot select any report items.

SHOWALL
overrides the parts of a definition that suppress the display of a column (NOPRINT and NOZERO). You define a report item with a DEFINE statement or in the DEFINITION window.

WRAP
displays one value from each column of the report, on consecutive lines if necessary, before displaying another value from the first column. By default, PROC REPORT displays values for only as many columns as it can fit on one page. It fills a page with values for these columns before starting to display values for the remaining columns on the next page.

Interaction: When WRAP is in effect, PROC REPORT ignores PAGE in any item definitions.

Tip: Typically, you use WRAP in conjunction with NAMED to avoid wrapping column headings.

BOX
uses formatting characters to add line-drawing characters to the report. These characters

- surround each page of the report
- separate column headings from the body of the report
- separate rows and columns from each other.

Interaction: You cannot use BOX if you use WRAP in the PROC REPORT statement or ROPTIONS window or if you use FLOW in any item's definition.

See also: For information about formatting characters, see the discussion of FORMCHAR= on page 1010.

MISSING
considers missing values as valid values for group, order, or across variables. Special missing values that are used to represent numeric values (the letters A through Z and the underscore (_) character) are each considered as a different value. A group for each missing value appears in the report. If you omit the MISSING option, then PROC REPORT does not include observations with a missing value for one or more group, order, or across variables in the report.

Attributes

LINESIZE=
specifies the line size for a report. PROC REPORT honors the first of these line-size specifications that it finds:

- LS= in the PROC REPORT statement or LINESIZE= in the ROPTIONS window
- the LS= setting stored in the report definition loaded with REPORT= in the PROC REPORT statement
- the SAS system option LINESIZE=.

Range: 64-256 (integer)

Tip: If the line size is greater than the width of the REPORT window, then use SAS interactive report window environment commands RIGHT and LEFT to display portions of the report that are not currently in the display.

PAGESIZE=
specifies the page size for a report. PROC REPORT honors the first of these page size specifications that it finds:

- PS= in the PROC REPORT statement or PAGESIZE= in the ROPTIONS window
- the PS= setting stored in the report definition loaded with REPORT= in the PROC REPORT statement
- the SAS system option PAGESIZE=.

Range: 15-32,767 (integer)

COLWIDTH=
specifies the default number of characters for columns containing computed variables or numeric data set variables.

Range: 1 to the line size

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 1058.) 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 heading when it reaches that character and continues the heading on the next line. The split character itself is not part of the column heading although each occurrence of the split character counts toward the 40-character maximum for a label.

Default: slash (/)

Interaction: The FLOW option in the DEFINE statement honors the split character.

Note: If you are typing over a heading (rather than entering one from the PROMPTER or DEFINITION window), then you do not see the effect of the split character until you refresh the screen by adding or deleting an item, by changing the contents of a 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 type of report is a telephone book, which contains multiple panels of names and telephone numbers on a single page.

When PROC REPORT writes a multipanel report, it fills one panel before beginning the next.

The number of panels that fits on a page depends on the

- width of the panel
- space between panels

□ line size.

Default: 1

Tip: If *number-of-panels* is larger than the number of panels that can fit on the page, then PROC REPORT creates as many panels as it can. Let PROC REPORT put your data in the maximum number of panels that can fit on the page by specifying a large number of panels (for example, 99).

See also: For information about specifying the space between panels see the discussion of PSPACE= on page 1072. For information about setting the line size, see the discussion of LINESIZE= on page 1070).

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 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 variables that are used in the report definition.

Buttons

OK
Creates the report definition and closes the SAVE DEFINITION window.

Cancel
Closes the SAVE DEFINITION window without creating a report definition.

SOURCE

Lists the PROC REPORT statements that build the current report.

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 1076. The sequence of events is the same whether you use programming statements or the interactive report window environment.

To understand the process of building a report, you must understand the difference between report variables and temporary variables. *Report variables* are variables that are specified in the COLUMN statement. A report variable can come from the input data set or can be computed (that is, the DEFINE statement for that variable specifies the COMPUTED option). A report variable might or might not appear in a compute block. Variables that appear only in one or more compute blocks are *temporary variables*. Temporary variables do not appear in the report and are not written to the output data set (if one is requested).

PROC REPORT constructs a report as follows:

1. It consolidates the data by group, order, and across variables. It calculates all statistics for the report, the statistics for detail rows as well as the statistics for summary lines in breaks. Statistics include those statistics that are computed for analysis variables. PROC REPORT calculates statistics for summary lines whether they appear in the report.
2. It initializes all temporary variables to missing.
3. It begins constructing the rows of the report.
   a. At the beginning of each row, it initializes all report variables to missing.
   b. It fills in values for report variables from left to right.

- Values for computed variables come from executing the statements in the corresponding compute blocks.
- Values for all other variables come from the data set or the summary statistics that were computed at the beginning of the report-building process.

c Whenever it comes to a break, PROC REPORT first constructs the break lines that are created with the BREAK or RBREAK statement or with options in the BREAK window. If there is a compute block attached to the break, then PROC REPORT then executes the statements in the compute block. See "Construction of Summary Lines" on page 1076 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 993. △

Note: You cannot use the LINE statement in conditional statements (IF-THEN, IF-THEN/ELSE, and SELECT) because it is not executed until PROC REPORT has executed all other statements in the compute block. △

4 After each report row is completed, PROC REPORT sends the row to all of the ODS destinations that are currently open.

Construction of Summary Lines

PROC REPORT constructs a summary line for a break if either of the following conditions is true:

- You summarize numeric variables in the break.
- You use a compute block at the break. (You can attach a compute block to a break without using a BREAK or RBREAK statement or without selecting any options in the BREAK window.)

  For more information about using compute blocks, see "Using Compute Blocks" on page 992 and "COMPUTE Statement" on page 1032.

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 51.2 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 51.2. The user-defined formats that are used are created by a PROC FORMAT step on page 1089.

```
libname proclib 'SAS-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 before;
    totprof = 0;
    endcomp;

    compute profit;
    if sector ne ' ' or department ne ' ' then do;
       if department='np1' or department='np2'
          then profit=0.4*sales.sum;
       else profit=0.25*sales.sum;
       totprof = totprof + profit;
       end;
    else
       profit = totprof;
    endcomp;

    rbreak after / dol dul summarize;
    compute after;
       sector='TOTALS:';
    endcomp;

    where sector contains 'n';
    title 'Report for Northeast and Northwest Sectors';
  run;
```

Output 51.2 Report with Groups and a Report Summary

```
          Report for Northeast and Northwest Sectors                1

Sector     Department      Sales     Profit          N
-------------------------------------------------------

Northeast  Canned         $840.00    $336.00         2
           Meat/Dairy     $490.00    $122.50         2
           Paper          $290.00     $116.00        2
           Produce        $211.00     $52.75         2
Northwest  Canned       $1,070.00    $428.00         3
           Meat/Dairy   $1,055.00    $263.75         3
           Paper          $150.00     $60.00         3
           Produce        $179.00     $44.75         3
=========  ============ =========  =========  =========
TOTALS:                 $4,285.00  $1,423.75        20
=========  ============ =========  =========  =========
```

A description of how PROC REPORT builds this report follows:

1 PROC REPORT starts building the report by consolidating the data (Sector and Department are group variables) and by calculating the statistics (Sales.sum and N) for each detail row and for the break at the end of the report.

2 Now, PROC REPORT is ready to start building the first row of the report. This report does not contain a break at the beginning of the report or a break before any groups, so the first row of the report is a detail row. The procedure initializes all report variables to missing, as the following figure illustrates. Missing values for a character variable are represented by a blank, and missing values for a numeric variable are represented by a period.

Figure 51.9 First Detail Row with Values Initialized

Sector	Department	Sales	Profit	N
		.	.	.

3 The following figure illustrates the construction of the first three columns of the row. PROC REPORT fills in values for the row from left to right. Values come from the statistics that were computed at the beginning of the report-building process.

Figure 51.10 First Detail Row with Values Filled in from Left to Right

Sector	Department	Sales	Profit	N
Northeast		.	.	.

Sector	Department	Sales	Profit	N
Northeast	Canned	.	.	.

Sector	Department	Sales	Profit	N
Northeast	Canned	$840.00	.	.

4 The next column in the report contains the computed variable Profit. When it gets to this column, PROC REPORT executes the statements in the compute block that is attached to Profit. Nonperishable items (which have a value of **np1** or **np2**) return a profit of 40%; perishable items (which have a value of **p1** or **p2**) return a profit of 25%.

```
if department='np1' or department='np2'
   then profit=0.4*sales.sum;
else profit=0.25*sales.sum;
```

The row now looks like the following figure.

Note: The position of a computed variable is important. PROC REPORT assigns values to the columns in a row of a report from left to right. Consequently, you cannot base the calculation of a computed variable on any variable that appears to its right in the report. △

Figure 51.11 A Computed Variable Added to the First Detail Row

Sector	Department	Sales	Profit	N
Northeast	Canned	$840.00	$336.00	.

5 Next, PROC REPORT fills in the value for the N statistic. The value comes from the statistics created at the beginning of the report-building process. The following figure illustrates the completed row.

Figure 51.12 First Complete Detail Row

Sector	Department	Sales	Profit	N
Northeast	Canned	$840.00	$336.00	2

6 The procedure writes the completed row to the report.

7 PROC REPORT repeats steps 2, 3, 4, 5, and 6 for each detail row in the report.

8 At the break at the end of the report, PROC REPORT constructs the break lines described by the RBREAK statement. These lines include double underlining, double overlining, and a preliminary version of the summary line. The statistics for the summary line were calculated earlier. (See step 1.) The value for the computed variable is calculated when PROC REPORT reaches the appropriate column, just as it is in detail rows. PROC REPORT uses these values to create the preliminary version of the summary line. (See the following figure.)

Figure 51.13 Preliminary Summary Line

Sector	Department	Sales	Profit	N
		$4,285.00	$1,071.25	20

9 If no compute block is attached to the break, then the preliminary version of the summary line is the same as the final version. However, in this example, a compute block is attached to the break. Therefore, PROC REPORT now executes the statements in that compute block. In this case, the compute block contains one statement:

```
sector='TOTALS:';
```

This statement replaces the value of Sector, which in the summary line is missing by default, with the word **TOTALS:**. After PROC REPORT executes the statement, it modifies the summary line to reflect this change to the value of Sector. The final version of the summary line appears in the following figure.

Figure 51.14 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 51.3 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 51.3. The user-defined formats that are used are created by a PROC FORMAT step on page 1089.

Note: Calculations of the percentages do not multiply their results by 100 because PROC REPORT prints them with the PERCENT. format. △

```
libname proclib 'SAS-library';


options nodate pageno=1 linesize=64
        pagesize=60 fmtsearch=(proclib);
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 51.3 Report with Temporary Variables

Report for Northeast and Northwest Sectors 1

Total for all stores is $4,285.00

Sector	Department	Sum of Sales	Percent of Sector	Percent of All Stores
Northeast		$1,831.00	100.00%	42.73%
Northeast	Canned	$840.00	45.88%	19.60%
	Meat/Dairy	$490.00	26.76%	11.44%
	Paper	$290.00	15.84%	6.77%
	Produce	$211.00	11.52%	4.92%
Northwest		$2,454.00	100.00%	57.27%
Northwest	Canned	$1,070.00	43.60%	24.97%
	Meat/Dairy	$1,055.00	42.99%	24.62%
	Paper	$150.00	6.11%	3.50%
	Produce	$179.00	7.29%	4.18%

A description of how PROC REPORT builds this report follows:

1 PROC REPORT starts building the report by consolidating the data (Sector and Department are group variables) and by calculating the statistics (Sales.sum and Sales.pctsum) for each detail row, for the break at the beginning of the report, for the breaks before each group, and for the breaks after each group.

2 PROC REPORT initializes the temporary variable, Sctrtot, to missing. (See the following figure.)

Figure 51.15 Initialized Temporary Variables

Report Variables					Temporary Variable
Sector	Department	Sales.sum	Sctrpct	Sales.pctsum	Sctrtot
		.	.	.	.

3 Because this PROC REPORT step contains a COMPUTE BEFORE statement, the procedure constructs a preliminary summary line for the break at the beginning of the report. This preliminary summary line contains values for the statistics (Sales.sum and Sales.pctsum) and the computed variable (Sctrpct).

At this break, Sales.sum is the sales for all stores, and Sales.pctsum is the percentage those sales represent for all stores (100%). PROC REPORT takes the values for these statistics from the statistics that were computed at the beginning of the report-building process.

The value for Sctrpct comes from executing the statements in the corresponding compute block. Because the value of Sctrtot is missing, PROC REPORT cannot calculate a value for Sctrpct. Therefore, in the preliminary summary line (which is not printed in this case), this variable also has a missing value. (See the following figure.)

The statements in the COMPUTE BEFORE block do not alter any variables. Therefore, the final summary line is the same as the preliminary summary line.

Note: The COMPUTE BEFORE statement creates a break at the beginning of the report. You do not need to use an RBREAK statement. △

Figure 51.16 Preliminary and Final Summary Line for the Break at the Beginning of the Report

	Report Variables				Temporary Variable
Sector	Department	Sales.sum	Sctrpct	Sales.pctsum	Sctrtot
		$4,285.00	.	100.00%	.

4 Because the program does not include an RBREAK statement with the SUMMARIZE option, PROC REPORT does not write the final summary line to the report. Instead, it uses LINE statements to write a customized summary that embeds the value of Sales.sum into a sentence and to write customized column headings. (The NOHEADER option in the PROC REPORT statement suppresses the default column headings, which would have appeared before the customized summary.)

5 Next, PROC REPORT constructs a preliminary summary line for the break before the first group of observations. (This break both uses the SUMMARIZE option in the BREAK statement and has a compute block attached to it. Either of these conditions generates a summary line.) The preliminary summary line contains values for the break variable (Sector), the statistics (Sales.sum and Sales.pctsum), and the computed variable (Sctrpct). At this break, Sales.sum is the sales for one sector (the northeast sector). PROC REPORT takes the values for Sector, Sales.sum, and Sales.pctsum from the statistics that were computed at the beginning of the report-building process.

The value for Sctrpct comes from executing the statements in the corresponding compute blocks. Because the value of Sctrtot is still missing, PROC REPORT cannot calculate a value for Sctrpct. Therefore, in the preliminary summary line, Sctrpct has a missing value. (See the following figure.)

Figure 51.17 Preliminary Summary Line for the Break before the First Group of Observations

	Report Variables				Temporary Variable
Sector	Department	Sales.sum	Sctrpct	Sales.pctsum	Sctrtot
Northeast		$1,831.00	.	42.73%	.

6 PROC REPORT creates the final version of the summary line by executing the statements in the COMPUTE BEFORE SECTOR compute block. These statements execute once each time the value of Sector changes.

- The first statement assigns the value of Sales.sum, which in that part of the report represents total sales for one Sector, to the variable Sctrtot.
- The second statement completes the summary line by recalculating Sctrpct from the new value of Sctrtot. The following figure shows the final summary line.

Note: In this example, you must recalculate the value for Sctrpct in the final summary line. If you do not recalculate the value for Sctrpct, then it will be missing because the value of Sctrtot is missing at the time that the COMPUTE Sctrpct block executes. △

Figure 51.18 Final Summary Line for the Break before the First Group of Observations

Report Variables					Temporary Variable
Sector	**Department**	**Sales.sum**	**Sctrpct**	**Sales.pctsum**	**Sctrtot**
Northeast		$1,831.00	100.00%	42.73%	$1,831.00

7 Because the program contains a BREAK BEFORE statement with the SUMMARIZE option, PROC REPORT writes the final summary line to the report. The UL option in the BREAK statement underlines the summary line.

8 Now, PROC REPORT is ready to start building the first report row. It initializes all report variables to missing. Values for temporary variables do not change. The following figure illustrates the first detail row at this point.

Figure 51.19 First Detail Row with Initialized Values

Report Variables					Temporary Variable
Sector	**Department**	**Sales.sum**	**Sctrpct**	**Sales.pctsum**	**Sctrtot**
		.	.	.	$1,831.00

9 The following figure illustrates the construction of the first three columns of the row. PROC REPORT fills in values for the row from left to right. The values come from the statistics that were computed at the beginning of the report-building process.

Figure 51.20 Filling in Values from Left to Right

Report Variables					Temporary Variable
Sector	**Department**	**Sales.sum**	**Sctrpct**	**Sales.pctsum**	**Sctrtot**
Northeast		.	.	.	$1,831.00

Report Variables					Temporary Variable
Sector	**Department**	**Sales.sum**	**Sctrpct**	**Sales.pctsum**	**Sctrtot**
Northeast	Canned	.	.	.	$1,831.00

Report Variables					Temporary Variable
Sector	**Department**	**Sales.sum**	**Sctrpct**	**Sales.pctsum**	**Sctrtot**
Northeast	Canned	$840.00	.	.	$1,831.00

10 The next column in the report contains the computed variable Sctrpct. When it gets to this column, PROC REPORT executes the statement in the compute block

attached to Sctrpct. This statement calculates the percentage of the sector's total sales that this department accounts for:

```
sctrpct=sales.sum/sctrtot;
```

The row now looks like the following figure.

Figure 51.21 First Detail Row with the First Computed Variable Added

Report Variables					Temporary Variable
Sector	Department	Sales.sum	Sctrpct	Sales.pctsum	Sctrtot
Northeast	Canned	$840.00	45.88%	.	$1,831.00

11 The next column in the report contains the statistic Sales.pctsum. PROC REPORT gets this value from the statistics created at the beginning of the report-building process. The first detail row is now complete. (See the following figure.)

Figure 51.22 First Complete Detail Row

Report Variables					Temporary Variable
Sector	Department	Sales.sum	Sctrpct	Sales.pctsum	Sctrtot
Northeast	Canned	$840.00	45.88%	19.60%	$1,831.00

12 PROC REPORT writes the detail row to the report. It repeats steps 8, 9, 10, 11, and 12 for each detail row in the group.

13 After writing the last detail row in the group to the report, PROC REPORT constructs the default group summary. Because no compute block is attached to this break and because the BREAK AFTER statement does not include the SUMMARIZE option, PROC REPORT does not construct a summary line. The only action at this break is that the SKIP option in the BREAK AFTER statement writes a blank line after the last detail row of the group.

14 Now the value of the break variable changes from **`Northeast`** to **`Northwest`**. PROC REPORT constructs a preliminary summary line for the break before this group of observations. As at the beginning of any row, PROC REPORT initializes all report variables to missing but retains the value of the temporary variable. Next, it completes the preliminary summary line with the appropriate values for the break variable (Sector), the statistics (Sales.sum and Sales.pctsum), and the computed variable (Sctrpct). At this break, Sales.sum is the sales for the Northwest sector. Because the COMPUTE BEFORE Sector block has not yet executed, the value of Sctrtot is still $1,831.00, the value for the Northeast sector. Thus, the value that PROC REPORT calculates for Sctrpct in this preliminary summary line is incorrect. (See the following figure.) The statements in the compute block for this break calculate the correct value. (See the following step.)

Figure 51.23 Preliminary Summary Line for the Break before the Second Group of Observations

Report Variables					Temporary Variable
Sector	Department	Sales.sum	Sctrpct	Sales.pctsum	Sctrtot
Northwest		$2,454.00	134.00%	57.27%	$1,831.00

CAUTION:
Synchronize values for computed variables in break lines to prevent incorrect results. If the PROC REPORT step does not recalculate Sctrpct in the compute block that is attached to the break, then the value in the final summary line will not be synchronized with the other values in the summary line, and the report will be incorrect. △

15 PROC REPORT creates the final version of the summary line by executing the statements in the COMPUTE BEFORE Sector compute block. These statements execute once each time the value of Sector changes.

- The first statement assigns the value of Sales.sum, which in that part of the report represents sales for the Northwest sector, to the variable Sctrtot.
- The second statement completes the summary line by recalculating Sctrpct from the new, appropriate value of Sctrtot. The following figure shows the final summary line.

Figure 51.24 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-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 library PROCLIB to the search path that is used to locate formats.

```
options fmtsearch=(proclib);
```

Specify the report options. The NOWD option runs the REPORT procedure without the REPORT window and sends its output to the open output destination.

```
proc report data=grocery nowd;
```

Specify the report columns. The report contains a column for Manager, Department, and Sales. Because there is no DEFINE statement for any of these variables, PROC REPORT uses the character variables (Manager and Department) as display variables and the numeric variable (Sales) as an analysis variable that is used to calculate the sum statistic.

```
   column manager department sales;
```

Produce a report summary. The RBREAK statement produces a default summary at the end of the report. DOL writes a line of equal signs (=) above the summary information. SUMMARIZE sums the value of Sales for all observations in the report.

```
   rbreak after / dol summarize;
```

Select the observations to process. The WHERE statement selects for the report only the observations for stores in the southeast sector.

```
    where sector='se';
```

Format the report columns. The FORMAT statement assigns formats to use in the report. You can use the FORMAT statement only with data set variables.

```
    format manager $mgrfmt. department $deptfmt.
           sales dollar11.2;
```

Specify the titles. SYSDATE is an automatic macro variable that returns the date when the SAS job or SAS session began. The TITLE2 statement uses double rather than single quotation marks so that the macro variable resolves.

```
    title 'Sales for the Southeast Sector';
    title2 "for &sysdate";
run;
```

Output

```
                Sales for the Southeast Sector                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 1088

Formats: $MGRFMT. and $DEPTFMT. on page 1089

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 headings and writes a blank line beneath the underlining
- creates a default summary of Sales for each manager
- creates a customized summary of Sales for the whole report.

Program

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

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

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page. 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 destinations. COLWIDTH=10 sets the default column width to 10 characters. SPACING= puts five blank characters between columns. HEADLINE underlines all column headings and the spaces between them at the top of each page of the report. HEADSKIP writes a blank line beneath the underlining that HEADLINE writes.

```
proc report data=grocery nowd
            colwidth=10
            spacing=5
            headline headskip;
```

Specify the report columns. The report contains a column for Manager, Department, and Sales.

```
column manager department sales;
```

Define the sort order variables. The values of all variables with the ORDER option in the DEFINE statement determine the order of the rows in the report. In this report, PROC REPORT arranges the rows first by the value of Manager (because it is the first variable in the COLUMN statement) and then by the values of Department.

ORDER= specifies the sort order for a variable. This report arranges the rows according to the formatted values of Manager and the internal values of Department (np1, np2, p1, and p2). FORMAT= specifies the formats to use in the report.

```
define manager / order order=formatted format=$mgrfmt.;
define department / order order=internal format=$deptfmt.;
```

Define the analysis variable. Sum calculates the sum statistic for all observations that are represented by the current row. In this report each row represents only one observation. Therefore, the Sum statistic is the same as the value of Sales for that observation in the input data set. Using Sales as an analysis variable in this report enables you to summarize the values for each group and at the end of the report.

```
define sales / analysis sum format=dollar7.2;
```

Produce a report summary. This BREAK statement produces a default summary after the last row for each manager. OL writes a row of hyphens above the summary line. SUMMARIZE writes the value of Sales (the only analysis or computed variable) in the summary line. PROC REPORT sums the values of Sales for each manager because Sales is an analysis variable that is used to calculate the Sum statistic. SKIP writes a blank line after the summary line.

```
break after manager / ol
                      summarize
                      skip;
```

Produce a customized summary. This COMPUTE statement begins a compute block that produces a customized summary at the end of the report. The LINE statement places the quoted text and the value of Sales.sum (with the DOLLAR9.2 format) in the summary. An ENDCOMP statement must end the compute block.

```
compute after;
   line 'Total sales for these stores were: '
        sales.sum dollar9.2;
endcomp;
```

Select the observations to process. The WHERE statement selects for the report only the observations for stores in the southeast sector.

```
where sector='se';
```

Specify the title.

```
title 'Sales for the Southeast Sector';
run;
```

Output

```
                Sales for the Southeast Sector                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 headings
LINE statement:
pointer controls
quoted text
repeating a character string
variable values and formats
writing a blank line

Other features:
automatic macro variables:
SYSDATE

Data set: GROCERY on page 1088

Formats: $MGRFMT. and $DEPTFMT. on page 1089

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-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 destinations. HEADLINE underlines all column headings and the spaces between them at the top of each page of the report. HEADSKIP writes a blank line beneath the underlining that HEADLINE writes.

```
proc report data=grocery nowd headline headskip;
```

Specify the report columns. The report contains columns for Manager and Department. It also contains three columns for Sales. The column specifications SALES=SALESMIN and SALES=SALESMAX create aliases for Sales. These aliases enable you to use a separate definition of Sales for each of the three columns.

```
column manager department sales
       sales=salesmin
       sales=salesmax;
```

Define the sort order variables. The values of all variables with the ORDER option in the DEFINE statement determine the order of the rows in the report. In this report, PROC REPORT arranges the rows first by the value of Manager (because it is the first variable in the COLUMN statement) and then by the values of Department. The ORDER= option specifies the sort order for a variable. This report arranges the values of Manager by their formatted values and arranges the values of Department by their internal values (np1, np2, p1, and p2). FORMAT= specifies the formats to use in the report. Text in quotation marks specifies column headings.

```
define manager / order
                 order=formatted
                 format=$mgrfmt.
                 'Manager';
define department    / order
                 order=internal
                 format=$deptfmt.
                 'Department';
```

Define the analysis variable. The value of an analysis variable in any row of a report is the value of the statistic that is associated with it (in this case Sum), calculated for all observations that are represented by that row. In a detail report each row represents only one observation. Therefore, the Sum statistic is the same as the value of Sales for that observation in the input data set.

```
define sales / analysis sum format=dollar7.2 'Sales';
```

Define additional analysis variables for use in the summary. These DEFINE statements use aliases from the COLUMN statement to create separate columns for the MIN and MAX statistics for the analysis variable Sales. NOPRINT suppresses the printing of these statistics. Although PROC REPORT does not print these values in columns, it has access to them so that it can print them in the summary.

```
define salesmin / analysis min noprint;
define salesmax / analysis max noprint;
```

Print a horizontal line at the end of the report. This COMPUTE statement begins a compute block that executes at the end of the report. The first LINE statement writes a blank line. The second LINE statement writes 53 hyphens (-), beginning in column 7. Note that the pointer control (@) has no effect on ODS destinations other than traditional SAS monospace output.

```
compute after;
   line ' ';
```

```
        line @7 53*'-';
```

Produce a customized summary. The first line of this LINE statement writes the text in quotation marks, beginning in column 7. The second line writes the value of Salesmin with the DOLLAR7.2 format, beginning in the next column. The cursor then moves one column to the right (+1), where PROC REPORT writes the text in quotation marks. Again, the cursor moves one column to the right, and PROC REPORT writes the value of Salesmax with the DOLLAR7.2 format. (Note that the program must reference the variables by their aliases.) The third line writes the text in quotation marks, beginning in the next column. Note that the pointer control (@) is designed for the Listing destination (traditional SAS output). It has no effect on ODS destinations other than traditional SAS monospace output. The ENDCOMP statement ends the compute block.

```
        line @7 '| Departmental sales ranged from'
             salesmin dollar7.2  +1 'to' +1 salesmax dollar7.2
             '. |';
        line @7 53*'-';
    endcomp;
```

Select the observations to process. The WHERE statement selects for the report only the observations for stores in the southeast sector.

```
    where sector='se';
```

Specify the titles. SYSDATE is an automatic macro variable that returns the date when the SAS job or SAS session began. The TITLE2 statement uses double rather than single quotation marks so that the macro variable resolves.

```
    title 'Sales for the Southeast Sector';
    title2 "for &sysdate";
run;
```

Output

```
                Sales for the Southeast Sector                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 headings

LINE statement:
- quoted text
- variable values

Data set: GROCERY on page 1088

Formats: $MGRFMT. and $DEPTFMT. on page 1089

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 headings.

Program

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

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

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page. 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 destinations. HEADLINE underlines all column headings and the spaces between them at the top of each page of the report. HEADSKIP writes a blank line beneath the underlining that HEADLINE writes.

```
proc report data=grocery nowd headline headskip;
```

Specify the report columns. The report contains columns for Sector, Manager, and Sales.

```
column sector manager sales;
```

Define the group and analysis variables. In this report, Sector and Manager are group variables. Sales is an analysis variable that is used to calculate the Sum statistic. Each detail row represents a set of observations that have a unique combination of formatted values for all group variables. The value of Sales in each detail row is the sum of Sales for all observations in the group. FORMAT= specifies the format to use in the report. Text in quotation marks in a DEFINE statement specifies the column heading.

```
define sector / group
                format=$sctrfmt.
                'Sector';
define manager / group
                 format=$mgrfmt.
                 'Manager';
define sales / analysis sum
               format=comma10.2
               'Sales';
```

Produce a report summary. This BREAK statement produces a default summary after the last row for each sector. OL writes a row of hyphens above the summary line. SUMMARIZE writes the value of Sales in the summary line. PROC REPORT sums the values of Sales for each manager because Sales is an analysis variable used to calculate the Sum statistic. SUPPRESS prevents PROC REPORT from displaying the value of Sector in the summary line. SKIP writes a blank line after the summary line.

```
break after sector / ol
                     summarize
                     suppress
                     skip;
```

Produce a customized summary. This compute block creates a customized summary at the end of the report. The LINE statement writes the quoted text and the value of Sales.sum (with a format of DOLLAR9.2) in the summary. An ENDCOMP statement must end the compute block.

```
compute after;
   line 'Combined sales for the northern sectors were '
         sales.sum dollar9.2 '.';
endcomp;
```

Specify a format for the summary rows. In detail rows, PROC REPORT displays the value of Sales with the format that is specified in its definition (COMMA10.2). The compute block specifies an alternate format to use in the current column on summary rows. Summary rows are identified as a value other than a blank for _BREAK_.

```
compute sales;
   if _break_ ne ' ' then
   call define(_col_,"format","dollar11.2");
endcomp;
```

Select the observations to process. The WHERE statement selects for the report only the observations for stores in the northeast and northwest sectors. The TITLE statement specifies the title.

```
where sector contains 'n';
```

Specify the title.

```
   title 'Sales Figures for Northern Sectors';
run;
```

Output

```
              Sales Figures for Northern Sectors              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 1088
Formats: $SCTRFMT., $MGRFMT., and $DEPTFMT. on page 1089

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 headings, some of which contain blank lines
- double-spaces between detail rows
- uses pointer controls to control the placement of text and variable values in a customized summary.

Program

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

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

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page. 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 destinations. HEADLINE underlines the column headings. HEADSKIP writes a blank line beneath the underlining that HEADLINE writes. SPLIT= defines the split character as an asterisk (*) because the default split character (/) is part of the name of a department.

```
proc report data=grocery nowd
            headline
```

```
        headskip
        split='*';
```

Specify the report columns. Department and Sales are separated by a comma in the COLUMN statement, so they collectively determine the contents of the column that they define. Each item generates a heading, but the heading for Sales is set to blank in its definition. Because Sales is an analysis variable, its values fill the cells that are created by these two variables.

```
column sector manager department,sales perish;
```

Define 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 heading. **`'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 1088

Formats: $MGRFMT. on page 1089

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-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 destinations. HEADLINE underlines all column headings and the spaces between them at the top of each page of the report. HEADSKIP writes a blank line beneath the underlining that HEADLINE writes. LS= sets the line size for the report to 66, and PS= sets the page size to 18.

```
proc report data=grocery nowd headline headskip
            ls=66 ps=18;
```

Specify the report columns. This COLUMN statement creates a column for Sector, Manager, and each of the six statistics that are associated with Sales.

```
column sector manager (Sum Min Max Range Mean Std),sales;
```

Define the group variables and the analysis variable. ID specifies that Manager is an ID variable. An ID variable and all columns to its left appear at the left of every page of a report. In this report, Sector and Manager are group variables. Each detail row of the report consolidates the information for all observations with the same values of the group variables. FORMAT= specifies the formats to use in the report.

```
   define manager / group format=$mgrfmt. id;
   define sector / group format=$sctrfmt.;
   define sales / format=dollar11.2 ;
```

Specify the title.

```
   title 'Sales Statistics for All Sectors';
run;
```

Output

Sales Statistics for All Sectors 1

Sector	Manager	Sum Sales	Min Sales	Max 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

Sector	Manager	Range Sales	Mean Sales	Std 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 1088

Formats: $SCTRFMT., $MGRFMT. and $DEPTFMT. on page 1089

The first PROC REPORT step in this example creates a report that displays one value from each column of the report, using two rows to do so, before displaying another value from the first column. (By default, PROC REPORT displays values for only as

many columns as it can fit on one page. It fills a page with values for these columns before starting to display values for the remaining columns on the next page.)

Each item in the report is identified in the body of the report rather than in a column heading.

The report definition created by the first PROC REPORT step is stored in a catalog entry. The second PROC REPORT step uses it to create a similar report for a different sector of the city.

Program to Store a Report Definition

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

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

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page. 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 destinations. NAMED writes *name*= in front of each value in the report, where *name*= is the column heading for the value. When you use NAMED, PROC REPORT suppresses the display of column headings at the top of each page.

```
proc report data=grocery nowd
            named
            wrap
            ls=64 ps=36
            outrept=proclib.reports.namewrap;
```

Specify the report columns. The report contains a column for Sector, Manager, Department, and Sales.

```
    column sector manager department sales;
```

Define the display and analysis variables. Because no usage is specified in the DEFINE statements, PROC REPORT uses the defaults. The character variables (Sector, Manager, and Department) are display variables. Sales is an analysis variable that is used to calculate the sum statistic. FORMAT= specifies the formats to use in the report.

```
    define sector / format=$sctrfmt.;
    define manager / format=$mgrfmt.;
    define department / format=$deptfmt.;
    define sales / format=dollar11.2;
```

Select the observations to process. A report definition might differ from the SAS program that creates the report. In particular, PROC REPORT stores neither WHERE statements nor TITLE statements.

```
   where manager='1';
```

Specify the title. SYSDATE is an automatic macro variable that returns the date when the SAS job or SAS session began. The TITLE statement uses double rather than single quotation marks so that the macro variable resolves.

```
   title "Sales Figures for Smith on &sysdate";
run;
```

Output

The following output is the output from the first PROC REPORT step, which creates the report definition.

```
              Sales Figures for Smith on 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 1088

Formats: $MGRFMT. and $DEPTFMT. on page 1089

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-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 destinations. HEADLINE underlines all column headings and the spaces between them at the top of each panel of the report. FORMCHAR= sets the value of the second formatting character (the one that HEADLINE uses) to the tilde (~). Therefore, the tilde underlines the column headings in the output. HEADSKIP writes a blank line beneath the underlining that HEADLINE writes. LS= sets the line size for the report to 64, and PS= sets the page size to 18. PANELS= creates a multipanel report. Specifying PANELS=99 ensures that PROC REPORT fits as many panels as possible on one page. PSPACE=6 places six spaces between panels.

```
proc report data=grocery nowd headline
            formchar(2)='~'
            panels=99 pspace=6
            ls=64 ps=18;
```

Specify the report columns. The report contains a column for Manager, Department, and Sales.

```
    column manager department sales;
```

Define the sort order and analysis columns. The values of all variables with the ORDER option in the DEFINE statement determine the order of the rows in the report. In this report, PROC REPORT arranges the rows first by the value of Manager (because it is the first variable in the COLUMN statement) and then, within each value of Manager, by the values of Department. The ORDER= option specifies the sort order for a variable. This report arranges the values of Manager by their formatted values and arranges the values of Department by their internal values (np1, np2, p1, and p2). FORMAT= specifies the formats to use in the report.

```
    define manager / order
                     order=formatted
                     format=$mgrfmt.;
    define department / order
                  order=internal
                  format=$deptfmt.;
```

```
    define sales / format=dollar7.2;
```

Produce a report summary. This BREAK statement produces a default summary after the last row for each manager. Because SKIP is the only option in the BREAK statement, each break consists of only a blank line.

```
    break after manager / skip;
```

Select the observations to process. The WHERE statement selects for the report only the observations for stores in the northwest or southwest sector.

```
    where sector='nw' or sector='sw';
```

Specify the title.

```
    title 'Sales for the Western Sectors';
run;
```

Output

```
                Sales for the Western Sectors                  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 1088

Formats: $SCTRFMT., $MGRFMT., and $DEPTFMT. on page 1089

The report in this example displays a record of one day's sales for each store. The rows are arranged so that all the information about one store is together, and the information for each store begins on a new page. Some variables appear in columns. Others appear only in the page heading that identifies the sector and the store's manager.

The heading that appears at the top of each page is created with the _PAGE_ argument in the COMPUTE statement.

Profit is a computed variable based on the value of Sales and Department.

The text that appears at the bottom of the page depends on the total of Sales for the store. Only the first two pages of the report appear here.

Program

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

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

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page. 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 destinations. NOHEADER in the PROC REPORT statement suppresses the default column headings.

```
proc report data=grocery nowd
            headline headskip;
```

Specify the title.

```
title 'Sales for Individual Stores';
```

Specify the report columns. The report contains a column for Sector, Manager, Department, Sales, and Profit, but the NOPRINT option suppresses the printing of the columns for Sector and Manager. The page heading (created later in the program) includes their values. To get these variable values into the page heading, Sector and Manager must be in the COLUMN statement.

```
column sector manager department sales Profit;
```

Define the group, computed, and analysis variables. In this report, Sector, Manager, and Department are group variables. Each detail row of the report consolidates the information for all observations with the same values of the group variables. Profit is a computed variable whose values are calculated in the next section of the program. FORMAT= specifies the formats to use in the report.

```
define sector / group noprint;
define manager / group noprint;
define profit / computed format=dollar11.2;
define sales / analysis sum format=dollar11.2;
define department / group format=$deptfmt.;
```

Calculate the computed variable. Profit is computed as a percentage of Sales. For nonperishable items, the profit is 40% of the sale price. For perishable items the profit is 25%. Notice that in the compute block you must reference the variable Sales with a compound name (Sales.sum) that identifies both the variable and the statistic that you calculate with it.

```
compute profit;
   if department='np1' or department='np2'
      then profit=0.4*sales.sum;
   else profit=0.25*sales.sum;
endcomp;
```

Create a customized page heading. This compute block executes at the top of each page, after PROC REPORT writes the title. It writes the page heading for the current manager's store. The LEFT option left-justifies the text in the LINE statements. Each LINE statement writes the text in quotation marks just as it appears in the statement. The first two LINE statements write a variable value with the format specified immediately after the variable's name.

```
 compute before _page_ / left;
   line sector $sctrfmt. ' Sector';
   line 'Store managed by ' manager $mgrfmt.;
   line ' ';
   line ' ';
   line ' ';
endcomp;
```

Produce a report summary. This BREAK statement creates a default summary after the last row for each manager. OL writes a row of hyphens above the summary line. SUMMARIZE writes the value of Sales (the only analysis or computed variable) in the summary line. The PAGE option starts a new page after each default summary so that the page heading that is created in the preceding compute block always pertains to the correct manager.

```
break after manager / ol summarize page;
```

Produce a customized summary. This compute block places conditional text in a customized summary that appears after the last detail row for each manager.

```
compute after manager;
```

Specify the length of the customized summary text. The LENGTH statement assigns a length of 35 to the temporary variable TEXT. In this particular case, the LENGTH statement is unnecessary because the longest version appears in the first IF/THEN statement. However, using the LENGTH statement ensures that even if the order of the conditional statements changes, TEXT will be long enough to hold the longest version.

```
   length text $ 35;
```

Specify the conditional logic for the customized summary text. You cannot use the LINE statement in conditional statements (IF-THEN, IF-THEN/ELSE, and SELECT) because it does not take effect until PROC REPORT has executed all other statements in the compute block. These IF-THEN/ELSE statements assign a value to TEXT based on the value of Sales.sum in the summary row. A LINE statement writes that variable, whatever its value happens to be.

```
   if sales.sum lt 500 then
      text='Sales are below the target region.';
   else if sales.sum ge 500 and sales.sum lt 1000 then
      text='Sales are in the target region.';
   else if sales.sum ge 1000 then
      text='Sales exceeded goal!';
   line ' ';
   line text $35.;
endcomp;
run;
```

Output

```
                  Sales for Individual Stores                  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 headings

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 1088

Formats: $MGRFMT. and $DEPTFMT. on page 1089

The summary report in this example shows the total sales for each store and the percentage that these sales represent of sales for all stores. Each of these columns has its own heading. A single heading also spans all the columns. This heading looks like a title, but it differs from a title because it would be stored in a report definition. You must submit a null TITLE statement whenever you use the report definition, or the report will contain both a title and the spanning heading.

The report includes a computed character variable, COMMENT, that flags stores with an unusually high percentage of sales. The text of COMMENT wraps across multiple rows. It makes sense to compute COMMENT only for individual stores. Therefore, the compute block that does the calculation includes conditional code that prevents PROC REPORT from calculating COMMENT on the summary line.

Program

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

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

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page. 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 destinations. 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 1089

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-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 destinations. HEADLINE underlines all column headings and the spaces between them.

```
proc report data=grocmiss nowd headline;
```

Specify the report columns. The report contains columns for Sector, Manager, the N statistic, and Sales.

```
   column sector manager N sales;
```

Define the group and analysis variables. In this report, Sector and Manager are group variables. Sales is, by default, an analysis variable that is used to calculate the Sum statistic. Each detail row represents a set of observations that have a unique combination of formatted values for all group variables. The value of Sales in each detail row is the sum of Sales for all observations in the group. In this PROC REPORT step, the procedure does not include observations with a missing value for the group variable. FORMAT= specifies formats to use in the report.

```
    define sector / group format=$sctrfmt.;
    define manager / group format=$mgrfmt.;
    define sales / format=dollar9.2;
```

Produce a report summary. This RBREAK statement creates a default summary at the end of the report. DOL writes a row of equal signs above the summary line. SUMMARIZE writes the values of N and Sales.sum in the summary line.

```
    rbreak after / dol summarize;
```

Specify the title.

```
    title 'Summary Report for All Sectors and Managers';
run;
```

Output with No Missing Values

```
          Summary Report for All Sectors and Managers          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 1088

Formats: $MGRFMT. on page 1089

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-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 destinations. OUT= creates the output data set TEMP. The output data set contains a variable for each column in the report (Manager and Sales) as well as for the variable _BREAK_, which is not used in this example. Each observation in the data set represents a row of the report. Because Manager is a group variable and Sales is an analysis variable that is used to calculate the Sum statistic, each row in the report (and therefore each observation in the output data set) represents multiple observations from the input data set. In particular, each value of Sales in the output data set is the total of all values of Sales for that manager. The WHERE= data set option in the OUT= option filters those rows as PROC REPORT creates the output data set. Only those observations with sales that exceed $1,000 become observations in the output data set.

```
proc report data=grocery nowd
            out=temp( where=(sales gt 1000) );
   column manager sales;
```

Define the group and analysis variables. Because the definitions of all report items in this report include the NOPRINT option, PROC REPORT does not print a report. However, the PROC REPORT step does execute and create an output data set.

```
   define manager / group noprint;
   define sales / analysis sum noprint;
run;
```

Output Showing the Output Data Set

The following output is the output data set that PROC REPORT creates. It is used as the input set in the second PROC REPORT step.

```
                        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 1088
Formats: $SCTRFMT. on page 1089

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-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 destinations. OUT= creates the output data set PROFIT.

```
proc report data=grocery nowd out=profit;
```

Specify the report columns. The report contains columns for Manager, Department, Sales, and Profit, which is not in the input data set. Because the purpose of this report is to generate an output data set to use in another procedure, the report layout simply uses the default usage for all the data set variables to list all the observations. DEFINE statements for the data set variables are unnecessary.

```
    column sector manager department sales Profit;
```

Define the computed column. The COMPUTED option tells PROC REPORT that Profit is defined in a compute block somewhere in the PROC REPORT step.

```
    define profit / computed;
```

Calculate the computed column. Profit is computed as a percentage of Sales. For nonperishable items, the profit is 40% of the sale price. For perishable items the profit is 25%. Notice that in the compute block, you must reference the variable Sales with a compound name (Sales.sum) that identifies both the variable and the statistic that you calculate with it.

```
       /* Compute values for Profit. */
    compute profit;
       if department='np1' or department='np2' then profit=0.4*sales.sum;
       else profit=0.25*sales.sum;
    endcomp;
run;
```

The Output Data Set

The following output is the output data set that is created by PROC REPORT. It is used as input for PROC CHART.

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

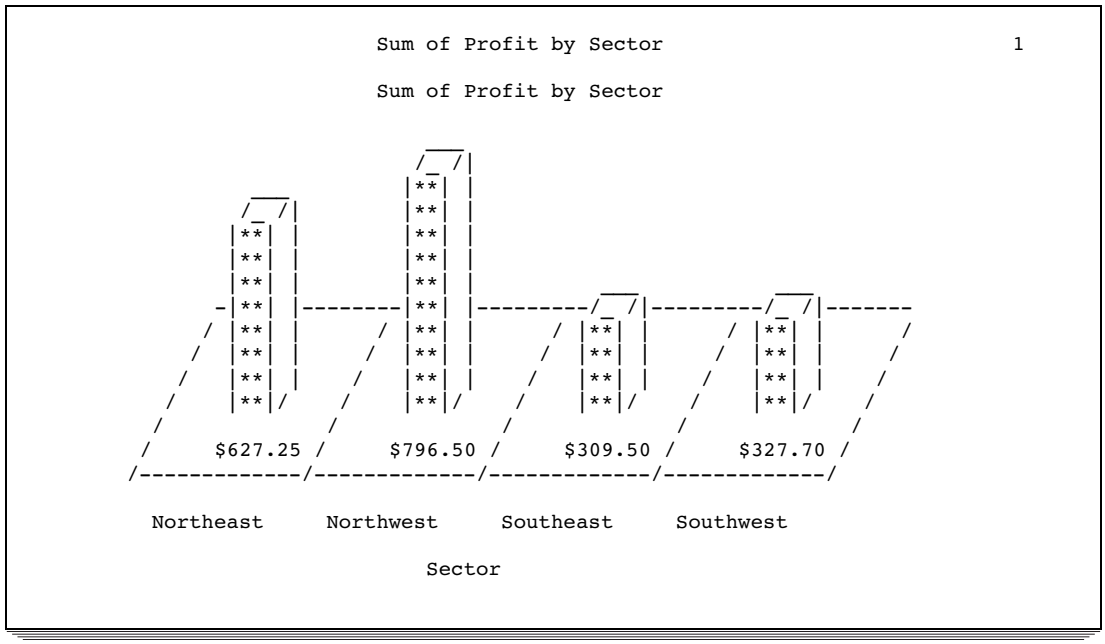
Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page. FMTSEARCH= specifies the library to include when 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



Example 14: Using a Format to Create Groups

Procedure features:
DEFINE statement options:
GROUP
Other features: FORMAT procedure
Data set: GROCERY on page 1088
Formats: $MGRFMT. on page 1089

This example shows how to use formats to control the number of groups that PROC REPORT creates. The program creates a format for Department that classifies the four departments as one of two types: perishable or nonperishable. Consequently, when Department is an across variable, PROC REPORT creates only two columns instead of four. The column heading is the formatted value of the variable.

Program

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

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

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page. 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 destinations. HEADLINE underlines all column headings and the spaces between them at the top of each page of the report. HEADSKIP writes a blank line beneath the underlining that HEADLINE writes.

```
proc report data=grocery nowd
     headline
     headskip;
```

Specify the report columns. Department and Sales are separated by a comma in the COLUMN statement, so they collectively determine the contents of the column that they define. Because Sales is an analysis variable, its values fill the cells that are created by these two variables. The report also contains a column for Manager and a column for Sales by itself (which is the sales for all departments).

```
    column manager department,sales sales;
```

Define the group and across variables. Manager is a group variable. Each detail row of the report consolidates the information for all observations with the same value of Manager. Department is an across variable. PROC REPORT creates a column and a column heading for each formatted value of Department. ORDER=FORMATTED arranges the values of Manager and Department alphabetically according to their formatted values. FORMAT= specifies the formats to use. The empty quotation marks in the definition of Department specify a blank column heading, so no heading spans all the departments. However, PROC REPORT uses the formatted values of Department to create a column heading for each individual department.

```
define manager / group order=formatted
                 format=$mgrfmt.;
define department / across order=formatted
             format=$perish. '';
```

Define the analysis variable. Sales is an analysis variable that is used to calculate the Sum statistic. Sales appears twice in the COLUMN statement, and the same definition applies to both occurrences. FORMAT= specifies the format to use in the report. WIDTH= specifies the width of the column. Notice that the column headings for the columns that both Department and Sales create are a combination of the heading for Department and the (default) heading for Sales.

```
define sales / analysis sum
               format=dollar9.2 width=13;
```

Produce a customized summary. This COMPUTE statement begins a compute block that produces a customized summary at the end of the report. The LINE statement places the quoted text and the value of Sales.sum (with the DOLLAR9.2 format) in the summary. An ENDCOMP statement must end the compute block.

```
compute after;
   line ' ';
   line 'Total sales for these stores were: '
         sales.sum dollar9.2;
endcomp;
```

Specify the title.

```
title 'Sales Summary for All Stores';
run;
```

Output

```
                Sales Summary for All Stores                  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 1088

Formats: $MGRFMT. and $DEPTFMT. on page 1089

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-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 the ones that are specified here.

```
style(header)=[color=yellow
               fontstyle=italic fontsize=6]
```

Specify the style attributes for the report columns. This STYLE= option sets the style element for all the cells in all the columns. Because no style element is specified, PROC REPORT uses all the style attributes of the default style element for this location except for the ones that are specified here.

```
style(column)=[color=moderate brown
               fontfamily=helvetica fontsize=4]
```

Specify the style attributes for the compute block lines. This STYLE= option sets the style element for all the LINE statements in all compute blocks. Because no style element is specified, PROC REPORT uses all the style attributes of the default style element for this location except for the ones that are specified here.

```
style(lines)=[color=white backgroundcolor=black
              fontstyle=italic fontweight=bold fontsize=5]
```

Specify the style attributes for report summaries. This STYLE= option sets the style element for all the default summary lines. Because no style element is specified, PROC REPORT uses all the style attributes of the default style element for this location except for the ones that are specified here.

```
style(summary)=[color=cx3e3d73 backgroundcolor=cxaeadd9
                fontfamily=helvetica fontsize=3 textalign=r];
```

Specify the report columns. The report contains columns for Manager, Department, and Sales.

```
    column manager department sales;
```

Define the sort order variables. In this report Manager and Department are order variables. PROC REPORT arranges the rows first by the value of Manager (because it is the first variable in the COLUMN statement), then by the value of Department. For Manager, ORDER= specifies that values of Manager are arranged according to their formatted values; similarly, for Department, ORDER= specifies that values of Department are arranged according to their internal values. FORMAT= specifies the format to use for each variable. Text in quotation marks specifies the column headings.

```
define manager / order
                 order=formatted
                 format=$mgrfmt.
                 'Manager';
define department / order
                    order=internal
                    format=$deptfmt.
                    'Department';
```

Produce a report summary. The BREAK statement produces a default summary after the last row for each manager. SUMMARIZE writes the values of Sales (the only analysis or computed variable in the report) in the summary line. PROC REPORT sums the values of Sales for each manager because Sales is an analysis variable that is used to calculate the Sum statistic.

```
break after manager / summarize;
```

Produce a customized summary. The COMPUTE statement begins a compute block that produces a customized summary after each value of Manager. The LINE statement places the quoted text and the values of Manager and Sales.sum (with the formats $MGRFMT. and DOLLAR7.2) in the summary. An ENDCOMP statement must end the compute block.

```
compute after manager;
   line 'Subtotal for ' manager $mgrfmt. 'is '
        sales.sum dollar7.2 '.';
endcomp;
```

Produce a customized end-of-report summary. This COMPUTE statement begins a compute block that executes at the end of the report. The LINE statement writes the quoted text and the value of Sales.sum (with the DOLLAR7.2 format). An ENDCOMP statement must end the compute block.

```
compute after;
    line 'Total for all departments is: '
         sales.sum dollar7.2 '.';
endcomp;
```

Select the observations to process. The WHERE statement selects for the report only the observations for stores in the southeast sector.

```
where sector='se';
```

Specify the title.

```
   title 'Sales for the Southeast Sector';
run;
```

Close the ODS destinations.

```
ods 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 1088
Formats: $MGRFMT. on page 1089 and $DEPTFMT. on page 1089

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-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 the ones that are specified here.

```
style(report)=[cellspacing=5 borderwidth=10 bordercolor=blue]
```

Specify the style attributes for the column headings. This STYLE= option sets the style element for all column headings. Because no style element is specified, PROC REPORT uses all the style attributes of the default style element for this location except for the ones that are specified here.

```
style(header)=[color=yellow
               fontstyle=italic fontsize=6]
```

Specify the style attributes for the report columns. This STYLE= option sets the style element for all the cells in all the columns. Because no style element is specified, PROC REPORT uses all the style attributes of the default style element for this location except for the ones that are specified here.

```
style(column)=[color=moderate brown
               fontfamily=helvetica fontsize=4]
```

Specify the style attributes for the compute block lines. This STYLE= option sets the style element for all the LINE statements in all compute blocks. Because no style element is specified, PROC REPORT uses all the style attributes of the default style element for this location except for the ones that are specified here.

```
style(lines)=[color=white backgroundcolor=black
              fontstyle=italic fontweight=bold fontsize=5]
```

Specify the style attributes for the report summaries. This STYLE= option sets the style element for all the default summary lines. Because no style element is specified, PROC REPORT uses all the style attributes of the default style element for this location except for the ones that are specified here.

```
style(summary)=[color=cx3e3d73 backgroundcolor=cxaeadd9
                fontfamily=helvetica fontsize=3 textalign=r];
```

Specify the report columns. The report contains columns for Manager, Department, and Sales.

```
column manager department sales;
```

Define the first sort order variable. In this report Manager is an order variable. PROC REPORT arranges the rows first by the value of Manager (because it is the first variable in the COLUMN statement). ORDER= specifies that values of Manager are arranged according to their formatted values. FORMAT= specifies the format to use for this variable. Text in quotation marks specifies the column headings.

```
define manager / order
                 order=formatted
                 format=$mgrfmt.
                 'Manager'
```

Specify the style attributes for the first sort order variable column heading. The STYLE= option sets the foreground and background colors of the column heading for Manager. The other style attributes for the column heading will match the ones that were established for the HEADER location in the PROC REPORT statement.

```
                 style(header)=[color=white
                                backgroundcolor=black];
```

Define the second sort order variable. In this report Department is an order variable. PROC REPORT arranges the rows first by the value of Manager (because it is the first variable in the COLUMN statement), then by the value of Department. ORDER= specifies that values of Department are arranged according to their internal values. FORMAT= specifies the format to use for this variable. Text in quotation marks specifies the column heading.

```
define department / order
                    order=internal
```

```
                        format=$deptfmt.
                        'Department'
```

Specify the style attributes for the second sort order variable column. The STYLE= option sets the font of the cells in the column Department to italic. The other style attributes for the cells will match the ones that were established for the COLUMN location in the PROC REPORT statement.

```
                        style(column)=[fontstyle=italic];
```

Produce a report summary. The BREAK statement produces a default summary after the last row for each manager. SUMMARIZE writes the values of Sales (the only analysis or computed variable in the report) in the summary line. PROC REPORT sums the values of Sales for each manager because Sales is an analysis variable that is used to calculate the Sum statistic.

```
break after manager / summarize;
```

Produce a customized summary. The COMPUTE statement begins a compute block that produces a customized summary at the end of the report. This STYLE= option specifies the style element to use for the text that is created by the LINE statement in this compute block. This style element switches the foreground and background colors that were specified for the LINES location in the PROC REPORT statement. It also changes the font style, the font weight, and the font size.

```
compute after manager
        / style=[fontstyle=roman fontsize=3 fontweight=bold
          backgroundcolor=white color=black];
```

Specify the text for the customized summary. The LINE statement places the quoted text and the values of Manager and Sales.sum (with the formats $MGRFMT. and DOLLAR7.2) in the summary. An ENDCOMP statement must end the compute block.

```
   line 'Subtotal for ' manager $mgrfmt. 'is '
        sales.sum dollar7.2 '.';
endcomp;
```

Produce a customized background for the analysis column. This compute block specifies a background color and a bold font for all cells in the Sales column that contain values of 100 or greater and that are not summary lines.

```
compute sales;
   if sales.sum>100 and _break_=' ' then
   call define(_col_, "style",
               "style=[backgroundcolor=yellow
                       fontfamily=helvetica
                       fontweight=bold]");
endcomp;
```

Produce a customized end-of-report summary. This COMPUTE statement begins a compute block that executes at the end of the report. The LINE statement writes the quoted text and the value of Sales.sum (with the DOLLAR7.2 format). An ENDCOMP statement must end the compute block.

```
compute after;
    line 'Total for all departments is: '
         sales.sum dollar7.2 '.';
```

```
    endcomp;
```

Select the observations to process. The WHERE statement selects for the report only the observations for stores in the southeast sector.

```
    where sector='se';
```

Specify the title.

```
    title 'Sales for the Southeast Sector';
run;
```

Close the ODS destinations.

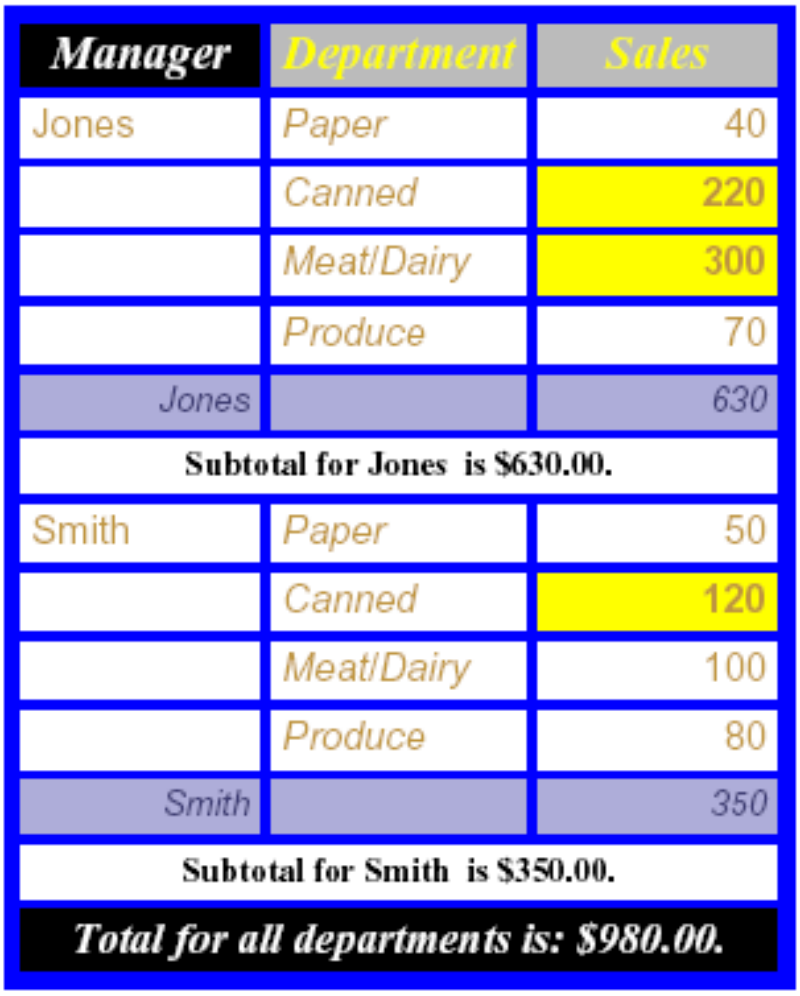
```
ods html close;
ods pdf close;
ods rtf close;
```

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

52

The SCAPROC Procedure

Overview: SCAPROC Procedure **1143**
Syntax: SCAPROC Procedure **1144**
PROC SCAPROC Statement **1144**
RECORD Statement **1144**
WRITE Statement **1145**
Results: SCAPROC Procedure **1145**
Examples: SCAPROC Procedure **1148**
Example 1: Specifying a Record File **1148**
Example 2: Specifying the Grid Job Generator **1149**

Overview: SCAPROC Procedure

The SCAPROC procedure implements the SAS Code Analyzer, which captures information about input, output, and the use of macro symbols from a SAS job while it is running. The SAS Code Analyzer can write this information and the information that is in the original SAS file to a file that you specify. The SCAPROC procedure can also generate a grid-enabled job that can concurrently run independent pieces of the job. You can issue the SCAPROC procedure on your operating system's command line or in SAS code in the SAS Editor window.

The following command runs your SAS job with the SAS Code Analyzer from your operating system's command line:

```
sas yourjob.sas -initstmt "proc scaproc;  record 'yourjob.txt' ; run;"
```

sas
is the command used at your site to start SAS.

yourjob.sas
is the name of the SAS job that you want to analyze.

yourjob.txt
is the name of the file that will contain a copy of your SAS code. The file will also contain the comments that are inserted to show input and output information, macro symbol usage, and other aspects of your job.

For information about issuing PROC SCAPROC in SAS code, see "Examples: SCAPROC Procedure" on page 1148.

Note: For the GRID statement to work, your site has to license SAS Grid Manager or SAS/CONNECT. SAS Grid Manager enables your generated grid job to run on a grid of distributed machines. SAS/CONNECT enables your generated grid job to run on parallel SAS sessions on one symmetric multiprocessing (SMP) machine. △

Syntax: SCAPROC Procedure

PROC SCAPROC;

<statements>;

Table 52.1

Task	Statement
Implement the SAS Code Analyzer	“PROC SCAPROC Statement” on page 1144
Specify a filename or a fileref to contain the output of the SAS Code Analyzer	“RECORD Statement” on page 1144
Output information to the record file	“WRITE Statement” on page 1145

PROC SCAPROC Statement

PROC SCAPROC;

specifies that SAS will run the SAS Code Analyzer with your SAS job.

RECORD Statement

Specify a filename or a fileref to contain the output of the SAS Code Analyzer.

RECORD *filespec* <ATTR> <OPENTIMES> <GRID *filespec* <RESOURCE "*resource name*">>;

Table 52.2

Task	Option
Specify a filename or a fileref to contain the output of the SAS Code Analyzer	*filespec*
Output additional information about the variables in data sets	ATTR

Task	Option
Output the open time, size, and physical name of input data sets and views	OPENTIMES
Specify a filename or a fileref to contain the output of the Grid Job Generator	GRID

filespec
specifies a physical filename in quotation marks, or a fileref, that indicates a file to contain the output of the SAS Code Analyzer. The output is the original SAS source and comments that contain information about the job. For more information about the output comments, see "Results: SCAPROC Procedure" on page 1145.

ATTR
outputs additional information about the variables in the input data sets and views.

OPENTIMES
outputs the open time, size, and physical filename of the input data sets.

GRID

filespec
specifies a physical filename in quotation marks, or a fileref, that points to a file that will contain the output of the Grid Job Generator.

RESOURCE "*resource name*"
specifies the resource to use in the **`grdsvs_enable`** function call. The default is SASMain.

WRITE Statement

Specify output information to the record file.

WRITE;

The WRITE statement specifies that the SAS Code Analyzer outputs information to the record file, if a file has been specified with the RECORD statement. The Grid Job Generator will also run at this time if it has been specified. Termination of SAS also causes the SAS Code Analyzer to output information to the specified record file.

Results: SCAPROC Procedure

The following list contains explanations of the comments that the SAS Code Analyzer writes to the record file that you specify with PROC SCAPROC. The output comments are bounded by `/*` and `*/` comment tags in the record file. That format is represented here to enhance clarity when the user reads a record file.

`/* JOBSPLIT: DATASET INPUT|OUTPUT|UPDATE SEQ|MULTI` ***`name`*** **`*/`**
specifies that a data set was opened for reading, writing, or updating.

INPUT
specifies that SAS read the data set.

OUTPUT
specifies that SAS wrote the data set.

UPDATE
specifies that SAS updated the data set.

SEQ
specifies that SAS opened the data set for sequential access.

MULTI
specifies that SAS opened the data set for multipass access.

name
specifies the name of the data set.

/* JOBSPLIT: CATALOG INPUT|OUTPUT|UPDATE *name* */
specifies that a catalog was opened for reading, writing, or updating.

INPUT
specifies that SAS read the catalog.

OUTPUT
specifies that SAS wrote the catalog.

UPDATE
specifies that SAS updated the catalog.

name
specifies the name of the catalog.

/* JOBSPLIT: FILE INPUT|OUTPUT|UPDATE *name* */
specifies that an external file was opened for reading, writing, or updating.

INPUT
specifies that SAS read the file.

OUTPUT
specifies that SAS wrote the file.

UPDATE
specifies that SAS updated the file.

name
specifies the name of the file.

/* JOBSPLIT: ITEMSTOR INPUT|OUTPUT|UPDATE *name* */
specifies that an ITEMSTOR was opened for reading, writing, or updating.

INPUT
specifies that SAS read the ITEMSTOR.

OUTPUT
specifies that SAS wrote the ITEMSTOR.

UPDATE
specifies that SAS updated the ITEMSTOR.

name
specifies the name of the ITEMSTOR.

/* JOBSPLIT: OPENTIME *name* DATE:*date* PHYS:*phys* SIZE:*size* */
specifies that a data set was opened for input. SAS outputs the OPENTIME and the SIZE of the file.

name
specifies the name of the data set.

DATE
specifies the date and time that the data set was opened. The value that is returned for DATE is not the creation time of the file.

PHYS
specifies the complete physical name of the data set that was opened.

SIZE
specifies the size of the data set in bytes.

/* JOBSPLIT: ATTR *name* INPUT|OUTPUT VARIABLE:*variable name* TYPE:CHARACTER|NUMERIC LENGTH:*length* LABEL:*label* FORMAT:*format* INFORMAT:*informat* */

specifies that when a data set is closed, SAS reopens it and outputs the attributes of each variable. One ATTR line is produced for each variable.

name
specifies the name of the data set.

INPUT
specifies that SAS read the data set.

OUTPUT
specifies that SAS wrote the data set.

VARIABLE
specifies the name of the current variable.

TYPE
specifies whether the variable is character or numeric.

LENGTH
specifies the length of the variable in bytes.

LABEL
specifies the variable label if it has one.

FORMAT
specifies the variable format if it has one.

INFORMAT
specifies the variable informat if it has one.

/* JOBSPLIT: SYMBOL SET|GET *name* */

specifies that a macro symbol was accessed.

SET
specifies that SAS set the symbol. For example, SAS set the symbol **sym1** in the following code:

```
%let sym1=sym2
```

GET
specifies that SAS retrieved the symbol. For example, SAS retrieved the symbol **sym** in the following code:

```
a="&sym"
```

name
specifies the name of the symbol.

/* JOBSPLIT: ELAPSED *number* */

specifies a number for you to use to determine the relative run times of tasks.

number
specifies a number for you to use to determine the relative run times of tasks.

/* JOBSPLIT: USER *useroption* */
specifies that SAS uses the USER option with the grid job code to enable single-level data set names to reside in the WORK library.

useroption
specifies the value that is to be used while the code is running.

/* JOBSPLIT: _DATA_ */
specifies that SAS is to use the reserved data set name _DATA_.

/* JOBSPLIT: _LAST_ */
specifies that SAS is to use the reserved data set name _LAST_ .

/* JOBSPLIT: PROCNAME procname|DATASTEP */
specifies the name of the SAS procedure or DATA step for this step.

/* JOBSPLIT: LIBNAME <libname options> */
specifies the LIBNAME options that were provided on a LIBNAME statement or were set internally.

/* JOBSPLIT: CONCATMEM <libref> <libname> */
specifies the name of a concatenated library that contains a specified libref.

Examples: SCAPROC Procedure

Example 1: Specifying a Record File

This example specifies the record file **'record.txt'**, and writes information from the SAS Code Analyzer to the file.

```
proc scaproc;
   record 'record.txt';
run;

data a;
   do i = 1 to 100000;
      j = cos(i);
      output;
   end;
run;

proc print data=a(obs=25);
run;

proc means data=a;
run;

proc scaproc;
   write;
```

```
run;
```

Contents of the **record.txt** file:

```
/* JOBSPLIT: DATASET OUTPUT SEQ WORK.A.DATA */
/* JOBSPLIT: LIBNAME WORK ENGINE V9 PHYS C:\DOCUME~1\userid\LOCALS~1\Temp\SAS Temporary Files\_TD1252 */
/* JOBSPLIT: ELAPSED 3984  */
/* JOBSPLIT: PROCNAME DATASTEP */
/* JOBSPLIT: STEP SOURCE FOLLOWS */

data a;
    do i = 1 to 1000000;
        j = cos(i);
        output;
    end;
run;


/* JOBSPLIT: ITEMSTOR INPUT SASUSER.TEMPLAT */
/* JOBSPLIT: ITEMSTOR INPUT SASHELP.TMPLMST */
/* JOBSPLIT: DATASET INPUT SEQ WORK.A.DATA */
/* JOBSPLIT: LIBNAME WORK ENGINE V9 PHYS C:\DOCUME~1\userid\LOCALS~1\Temp\SAS Temporary Files\_TD1252 */
/* JOBSPLIT: ELAPSED 5187  */
/* JOBSPLIT: PROCNAME PRINT */
/* JOBSPLIT: STEP SOURCE FOLLOWS */
proc print data=a(obs=25);
run;


/* JOBSPLIT: DATASET INPUT SEQ WORK.A.DATA */
/* JOBSPLIT: LIBNAME WORK ENGINE V9 PHYS C:\DOCUME~1\userid\LOCALS~1\Temp\SAS Temporary Files\_TD1252 */
/* JOBSPLIT: FILE OUTPUT C:\winnt\profiles\userid\record.txt */
/* JOBSPLIT: SYMBOL GET SYSSUMTRACE */
/* JOBSPLIT: ELAPSED 2750  */
/* JOBSPLIT: PROCNAME MEANS */
/* JOBSPLIT: STEP SOURCE FOLLOWS */
proc means data=a;
run;


/* JOBSPLIT: END */
```

Example 2: Specifying the Grid Job Generator

This example writes information from the SAS Code Analyzer to the file named **1.txt**. The example code also runs the Grid Job Generator, and writes that information to the file named **1.grid**. Notice that this example does not have an ending statement that contains this code:

```
proc scaproc;
   write;
run;
```

When SAS terminates, PROC SCAPROC automatically runs any pending RECORD or GRID statements.

Note: For the GRID statement to work, your site has to license SAS Grid Manager or SAS/CONNECT. SAS Grid Manager enables your generated grid job to run on a grid of distributed machines. SAS/CONNECT enables your generated grid job to run on parallel SAS sessions on one symmetric multiprocessing (SMP) machine. △

```
proc scaproc;
   record '1.txt' grid '1.grid';
run;

data a;
   do i = 1 to 100000;
      j = cos(i);
      output;
   end;
run;

proc print data=a(obs=25);
run;

proc means data=a;
run;
```

Contents of the **`1.txt`** file:

```
/* JOBSPLIT: DATASET OUTPUT SEQ WORK.A.DATA */
/* JOBSPLIT: LIBNAME WORK ENGINE V9 PHYS C:\DOCUME~1\userid\LOCALS~1\Temp\SAS Temporary Files\_TD1252 */
/* JOBSPLIT: ELAPSED 375  */
/* JOBSPLIT: PROCNAME DATASTEP */
/* JOBSPLIT: STEP SOURCE FOLLOWS */

data a;
    do i = 1 to 1000000;
        j = cos(i);
        output;
    end;
run;



/* JOBSPLIT: DATASET INPUT SEQ WORK.A.DATA */
/* JOBSPLIT: LIBNAME WORK ENGINE V9 PHYS C:\DOCUME~1\userid\LOCALS~1\Temp\SAS Temporary Files\_TD1252 */
/* JOBSPLIT: ELAPSED 46  */
/* JOBSPLIT: PROCNAME PRINT */
/* JOBSPLIT: STEP SOURCE FOLLOWS */
proc print data=a(obs=25);
run;



/* JOBSPLIT: DATASET INPUT SEQ WORK.A.DATA */
/* JOBSPLIT: LIBNAME WORK ENGINE V9 PHYS C:\DOCUME~1\userid\LOCALS~1\Temp\SAS Temporary Files\_TD1252 */
/* JOBSPLIT: FILE OUTPUT C:\WINNT\Profiles\userid\1.txt */
/* JOBSPLIT: SYMBOL GET SYSSUMTRACE */
/* JOBSPLIT: ELAPSED 81453  */
/* JOBSPLIT: PROCNAME MEANS */
/* JOBSPLIT: STEP SOURCE FOLLOWS */
```

```
proc means data=a;
run;

/* JOBSPLIT: END */
```

CHAPTER

53

The SOAP Procedure

Overview: SOAP Procedure **1153**
What Does the Simple Object Access Protocol (SOAP) Procedure Do? **1153**
Syntax: SOAP Procedure **1154**
PROC SOAP Statement **1154**
Concepts: SOAP Procedure **1157**
WS-Security: Client Configuration **1157**
Using PROC SOAP with Secure Socket Layer (SSL) **1158**
SSL and Data Encryption **1158**
Making PROC SOAP Calls by Using the HTTPS Protocol **1158**
Tracing the Request and Response **1158**
Methods of Calling SAS Web Services **1159**
Examples: SOAP Procedure **1160**
Example 1: Using PROC SOAP with a SOAPEnvelope Element **1160**
Example 2: Using PROC SOAP without a SOAPEnvelope Element **1160**
Example 3: Calling a Web Service by Using a Proxy **1161**
Example 4: Calling a SAS Web Service Using the Service Registry Service **1162**
Example 5: Calling a SAS Web Service Using the SAS Environments File **1162**

Overview: SOAP Procedure

What Does the Simple Object Access Protocol (SOAP) Procedure Do?

PROC SOAP reads XML input from a file that has a fileref and writes XML output to another file that has a fileref. The envelope and headings are part of the content of the fileref. They are defined in the IN option of PROC SOAP. The input XML is either a SOAPEnvelope element, or an element inside the SOAPEnvelope that is required to invoke the Web service.

Operating Environment Information: PROC SOAP can run on any platform; however, WS-Security features are not available in the z/OS operating environment. The message component is an XML document that corresponds to a service request. △

Syntax: SOAP Procedure

PROC SOAP *option(s)* <*properties*>;

Task	Statement
Invoke a Web service.	Chapter 53, "The SOAP Procedure," on page 1153

PROC SOAP Statement

Invokes a Web service through Java Native Interface (JNI).

PROC SOAP *option(s)* <*properties*>;

Task	Option
Specify a complete path to the Axis2 repository, which is a folder or directory to which the deployment search mechanism is limited.	AXIS2CONFIGDIR on page 1155
Specify a complete path to your Axis2 configuration XML file.	AXIS2CONFIGFILE on page 1155
Specify the location of the SAS environments file.	ENVFILE on page 1155
Specify to use the environment that is defined in the SAS environments file.	ENVIRONMENT on page 1156
Specify the fileref to input XML data that contains the SOAP request.	IN on page 1156
Specify the fileref where the SOAP response output XML will be written.	OUT on page 1156
Specify an HTTP proxy server host name.	PROXYHOST on page 1156
Specify an HTTP proxy server port.	PROXYPORT on page 1156
Specify an HTTP proxy server domain.	PROXYDOMAIN on page 1156
Specify an HTTP proxy server password. Encodings that are produced by PROC PWENCODE are supported.	PROXYPASSWORD on page 1156
Specify an HTTP proxy server user name.	PROXYUSERNAME on page 1156
Specify a SOAPAction element to invoke on the Web service.	SOAPACTION on page 1156

Task	Option
Specify the URL of the System Registry Service.	SRSURL on page 1156
Specify the SAS Web service to call.	SERVICE on page 1156
Specify a URL of the Web service endpoint.	URL on page 1156
Specify that a user name and password be retrieved from metadata for the specified authentication domain.	WEBAUTHDOMAIN on page 1156
Specify the domain or realm for the user name and password for NTLM authentication.	WEBDOMAIN on page 1156
Specify a password for either Basic or NTLM Web server authentication. Encodings that are produced by PROC PWENCODE are supported.	WEBPASSWORD on page 1156
Specify a user name for either Basic or NTLM Web authentication.	WEBUSERNAME on page 1156
Specify that the active connection to the SAS Metadata Server will be used to retrieve credentials in the specified authentication domain.	WSSAUTHDOMAIN on page 1156
Specify a WS-Security password, which is the password for WSSUSERNAME. Encodings that are produced by PROC PWENCODE are supported.	WSSPASSWORD on page 1157
Specify a WS-Security user name.	WSSUSERNAME on page 1157

Operating Environment Information: In the VMS operating environment, you must use the RECFM=*streamlf* option on the FILE statement to create valid XML. The following example shows this option in the FILE statement:

```
FILE REQUEST recfm=streamlf;
```

△

Options

AXIS2CONFIGDIR
specifies a complete path to the Axis2 repository, which is a folder or directory to which the deployment search mechanism is limited.

The deployment model can load and read files only inside the repository. The repository includes two subdirectories: services and modules. Service archives are located in the services directory, and modules are located in the modules directory. If WS-Security is required, then rampart.mar is required. The version of rampart.mar that is required for PROC SOAP will be listed in the classpath. However, if you choose to deploy rampart.mar or other modules in a repository, you could use this option to specify the repository location.

AXIS2CONFIGFILE
specifies the complete path to your Axis2 configuration XML file. If you do not specify this option, then axis2_default.xml in the Axis2 jar is used.

ENVFILE
specifies the location of the SAS environments file.

ENVIRONMENT
specifies to use the environment that is defined in the SAS environments file.

IN=*fileref*|*'your-input-file'*
specifies the fileref that is used to input XML data that contains a PROC SOAP request. The fileref might have SOAPEnvelope and SOAPHeader elements as part of its content, but they are not required unless you have specific header information to provide.

MUSTUNDERSTAND
specifies the setting for the mustUnderstand attribute in the PROC SOAP header.

OUT=*fileref*|*'your-output-file'*
specifies the fileref where the PROC SOAP XML response output is written.

PROXYDOMAIN
specifies an HTTP proxy server domain. This option is required only if your proxy server requires domain- or realm-qualified credentials.

PROXYHOST
specifies an HTTP proxy server host name.

PROXYPASSWORD
specifies an HTTP proxy server password. This option is required only if your proxy server requires credentials.

PROXYPORT
specifies an HTTP proxy server port.

PROXYUSERNAME
specifies an HTTP proxy server user name. This option is required only if your proxy server requires credentials.

SERVICE
specifies the SAS Web service to use.

SOAPACTION
specifies a SOAPAction element to invoke on the Web service.

SRSURL
specifies the URL of the System Registry Services.

URL
specifies the URL of the Web service endpoint.

WEBAUTHDOMAIN
specifies that a user name and password be retrieved from metadata for the specified authentication domain.

WEBDOMAIN
specifies the domain or realm for the user name and password for NTLM authentication.

WEBPASSWORD
specifies a password for either Basic or NTLM Web server authentication.

WEBUSERNAME
specifies a user name for either Basic or NTLM Web server authentication.

WSSAUTHDOMAIN
specifies that the active connection to the SAS Metadata Server will be used to retrieve credentials in the specified authentication domain.

If credentials are found, they will be used as the credentials for a WS-Security UsernameToken.

WSSPASSWORD
specifies a WS-Security password that is the password for WSSUSERNAME.

WSSUSERNAME
specifies a WS-Security user name. If a value is set, then WS-Security is used and a UsernameToken is sent with the Web service request for user authentication, security, and encryption.

Properties

ENVELOPE
specifies that a SOAP envelope is to be included in the response.

Concepts: SOAP Procedure

With PROC SOAP, you can include an optional SOAPEnvelope element in your XML file. Do this if you want to include custom information in the SOAPHeader element. A SOAP envelope wraps the message, which has an application-specific message vocabulary. The SOAPHeader content will be added to the actual Web service request that is made by Axis2. This addition occurs because there could be additional SOAPHeader elements included by Axis2. These elements support WS-Addressing or WS-Security that were not included in the XML file that was passed to PROC SOAP. The XML code that is transmitted might not exactly match the XML code provided in this case.

A request does not need to be contained in a SOAP envelope. Axis2 will add the appropriate envelope if you do not specify an envelope, or will incorporate the specified envelope into the envelope that is sent. A response is returned within an envelope only if the envelope property is set. The default behavior is to return only the contents of the envelope.

WS-Security: Client Configuration

The client provides the user name and password to be added to the UsernameToken element. You can provide this information in the outflow configuration of the Apache Rampart module as shown in the following example:

```
<parameter name="OutflowSecurity">
    <action>
        <items>UsernameToken</items>
        <passwordType>PasswordText</passwordType>
     </action>
</parameter>
```

Include the configuration parameter in the client's Axis2 configuration XML file, which is typically axis2.xml.

This configuration results in a password digest being sent to the service. To pass a plain-text password, include the following code as an <action> element:

```
<passwordType>PasswordText</passwordType>
```

Note that passing plain-text passwords is not recommended.

Using PROC SOAP with Secure Socket Layer (SSL)

SSL and Data Encryption

SSL enables Web browsers and Web servers to communicate over a secured connection by encrypting data. Both browsers and servers encrypt data before the data is transmitted. The receiving browser or server then decryptes the data before it is processed.

Making PROC SOAP Calls by Using the HTTPS Protocol

In order to make PROC SOAP calls by using the HTTPS protocol, you must configure a trust source that contains the certificate of the service to be trusted. This trust store and its password must be provided to the SAS session by setting Java system options using jreoptions. You can provide this information on the SAS command line or in a SAS configuration file. Use the following syntax. Be sure to enter the following entry on one line:

```
-jreoptions (-Djavax.net.ssl.trustStore=full-path-to-the-trust-store
 -Djavax.net.ssl.trustStorePassword=trustStorePassword)
```

The following example shows how to use the entry on the SAS command line. The example uses the Windows operating environment. Be sure to enter the following entry on one line:

```
"C:\Program Files\SAS\SASFoundation\9.2\sas.exe" -CONFIG "C:\Program Files\SAS
\SASFoundation\9.2\nls\en\SASV9.CFG" -jreoptions (- Djavax.net.ssl.trustStore=C:
\Documents and Settings\mydir\.keystore
-Djavax.net.ssl.trustStorePassword=trustpassword)
```

Tracing the Request and Response

PROC SOAP uses log4j for logging requests and responses so that you can trace them. To create a log file that contains the request issued and the response received, create a file that has the following contents:

```
log4j.appender.FILE=org.apache.log4j.FileAppender
log4j.appender.FILE.File=wire.log
log4j.appender.FILE.layout=org.apache.log4j.PatternLayout
log4j.appender.FILE.layout.ConversionPattern =%d %5p [%c] %m%n


log4j.logger.httpclient.wire=DEBUG, FILE
#log4j.logger.org.apache.commons.httpclient=DEBUG, FILE
```

This configuration logs only the request and response as it is sent and received. To log context information, which includes settings and system information, remove the comment character in the last line of the configuration shown above.

Enable logging by setting a Java system option using jreoptions on the SAS command line or in a SAS configuration file. The following syntax shows how to set the system option:

```
-jreoptions (-Dlog4j.configuration=path-to-log4j-config-file)
```

The following example shows how to use the entry on the SAS command line. The example uses the Windows operating environment. Be sure to enter the entry on one line:

```
"C:\Program Files\SAS\SASFoundation\9.2\sas.exe" -CONFIG "C:\Program Files\SAS
\SASFoundation\9.2\nls\en\SASV9.CFG" -jreoptions
(- Dlog4j.configuration=file:/c:/public/log4j.properties)
```

Methods of Calling SAS Web Services

You can use two methods to call SAS Web services. The first method requires that you know the URL of the Service Registry Service and the URL of the endpoint of the service you are calling. You must set the URL of the Service Registry Service on the SRSURL option. The URL option indicates the endpoint of the service that you are calling. See Example 4 on page 1162.

The second method used to call SAS Web services uses the SAS environments file to specify the endpoint of the service you are calling. Using this method, you can indicate the location of the SAS environments file in one of two ways:

- use the ENVFILE option in PROC SOAP
- define the Java property **env.definition.location** in JREOPTIONS on the SAS command line or in the SAS configuration file

Use the following -JREOPTIONS syntax:

```
-jreoptions (-Denv.definition.location=http://your-SAS-environment.xml)
```

You must also specify the desired environment within that file using the ENVIRONMENT option, and specify the name of the service you are calling using the SERVICE option. See Example 5 on page 1162.

In both cases, the WSUSERNAME and WSPASSWORD options will be set to the user name and password that are required to contact the Service Registry Service. Use the AXIS2CONFIGFILE option to indicate the configuration file that contains the WS-Security configuration for Axis2.

Your Axis2 configuration file must have the OutflowSecurity section defined as follows:

```
<parameter-name = "OutflowSecurity">
   <action>
      <items>UsernameToken Timestamp </items>
   </action>
</parameter>
```

Examples: SOAP Procedure

Example 1: Using PROC SOAP with a SOAPEnvelope Element

This example calls a service that requires WS-Security. It provides an envelope:

```
FILENAME REQUEST 'C:\temp\simpleTest_REQUEST.xml';
FILENAME RESPONSE 'C:\temp\simpleTest_RESPONSE.xml';

data _null_;
   file request;
   input;
   put _infile_;
   datalines4;
<soapenv:Envelope xmlns:add="http://tempuri.org/addintegersWS"
                  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Header/>
   <soapenv:Body>
      <add:addintegers>
         <add:parameters>
            <add:int1>20</add:int1>
            <add:int2>30</add:int2>
         </add:parameters>
      </add:addintegers>
   </soapenv:Body>
   </soapenv:Envelope>
;;;;

run;

%let RESPONSE=RESPONSE;
proc soap in=REQUEST
          out=&RESPONSE
          url="http://localhost:8080/SASBIWS/services/addintegersWS"
          wssusername="your-user-name"
          wsspassword="your-password"
          axis2configfile="C:\temp\axis2.xml";

run;
```

Example 2: Using PROC SOAP without a SOAPEnvelope Element

This example calls the same service as is called in the process described in Example 1 on page 1160. But here the service is called without an envelope:

```
FILENAME REQUEST 'C:\temp\simpleTest_REQUEST.xml';
FILENAME RESPONSE 'C:\temp\simpleTest_RESPONSE.xml';
```

```
data _null_;
   file request;
   input;
   put _infile_;
   datalines4;
<add:addintegers xmlns:add="http://tempuri.org/addintegersWS">
   <add:parameters>
      <add:int1>20</add:int1>
      <add:int2>30</add:int2>
   </add:parameters>
</add:addintegers>
;;;;

run;

%let RESPONSE=RESPONSE;
proc soap in=REQUEST
          out=&RESPONSE
          url="http://localhost:8080/SASBIWS/services/addintegersWS"
          wssusername="your-user-name"
          wsspassword="your-password"
          axis2configfile="C:\temp\axis2.xml";
run;
```

Example 3: Calling a Web Service by Using a Proxy

This example calls an external Web service and therefore uses a proxy:

```
FILENAME REQUEST TEMP;
FILENAME RESPONSE "C:\temp\Output.xml";

data _null_;
   file request;
   input;
   put _infile_;
   datalines4;
<soapenv:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
                  xmlns:ndf="http://www.weather.gov/forecasts/xml/
                            DWMLgen/wsdl/ndfdXML.wsdl">
   <soapenv:Header/>
   <soapenv:Body>
      <ndf:NDFDgenByDay soapenv:encodingStyle=
                        "http://schemas.xmlsoap.org/soap/encoding/">
         <latitude xsi:type="xsd:decimal">35.79</latitude>
         <longitude xsi:type="xsd:decimal">-78.82</longitude>
         <startDate xsi:type="xsd:date">2006-10-03</startDate>
         <numDays xsi:type="xsd:integer">3</numDays>
         <format xsi:type="dwml:formatType"
                 xmlns:dwml="http://www.weather.gov/forecasts/xml/
```

```
                                DWMLgen/schema/DWML.xsd">24 hourly</format>
      </ndf:NDFDgenByDay>
   </soapenv:Body>
  </soapenv:Envelope>
;;;;

proc soap in=request
          out=response
          url="http://www.weather.gov/forecasts/xml/SOAP_server/
               ndfdXMLserver.php"
          soapaction="http://www.weather.gov/forecasts/xml/DWMLgen/
                      wsdl/ndfdXML.wsdl#NDFDgenByDay"
          proxyhost="proxygw.abc.sas.com"
          proxyport=80;
run;
```

Example 4: Calling a SAS Web Service Using the Service Registry Service

This example calls a SAS Web service using the service URL and the Service Registry Service:

```
filename request temp;
filename response "c:\temp\output.xml";

proc soap in=request
                out=response
url="http://machine.abc.xyz.com:1234/SASWIPSoapServices/services/
   ReportRepositoryService"
soapaction="http://www.xyz.com/xml/schema/sas-svcs/reportrepository-9.2/
   DirectoryServiceInterface/isDirectory"
srsurl="http://machine.abc.xyz.com:1234/SASWIPSoapServices/services/
   ServiceRegistry"
       wssusername="your-user-name"
       wsspassword="your-password"
       axis2configfile="c:\temp\axis2wip.xml";
run;
```

Example 5: Calling a SAS Web Service Using the SAS Environments File

This example uses the SAS environments file and the test environment to call the SAS Web service:

```
filename request temp;
filename response "c:\temp\output.xml";

proc soap in=request
```

```
                out=response
                service="ReportRepositoryService"
soapaction="http://machine.abc.xyz.com:1234/SASWIPSoapServices/services/
   ReportRepositoryService"
                envfile="http://file-server-name.abc.xyz.com/sas-environment.xml"
                environment="test";
                    wssusername="your-user-name""
                    wsspassword="your-password"
                    axis2configfile="c:\temp\axis2wip.xml";
run;
```

CHAPTER

54

The SORT Procedure

Overview: SORT Procedure **1165**
What Does the SORT Procedure Do? **1165**
Sorting SAS Data Sets **1166**
Syntax: SORT Procedure **1167**
PROC SORT Statement **1167**
BY Statement **1179**
KEY Statement **1180**
Concepts: SORT Procedure **1181**
Multi-threaded Sorting **1181**
Sorting Orders for Numeric Variables **1182**
Sorting Orders for Character Variables **1182**
Default Collating Sequence **1182**
EBCDIC Order **1182**
ASCII Order **1183**
Specifying Sorting Orders for Character Variables **1183**
Stored Sort Information **1183**
Presorted Input Data Sets **1184**
In-Database Processing: PROC SORT **1184**
Integrity Constraints: SORT Procedure **1185**
Results: SORT Procedure **1186**
Procedure Output **1186**
Output Data Set **1186**
Examples: SORT Procedure **1187**
Example 1: Sorting by the Values of Multiple Variables **1187**
Example 2: Sorting in Descending Order **1189**
Example 3: Maintaining the Relative Order of Observations in Each BY Group **1191**
Example 4: Retaining the First Observation of Each BY Group **1193**

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 1186.

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 54.1 shows the log that results from running this PROC SORT step. Output 54.2 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 54.1 SAS Log Generated by PROC SORT

```
NOTE: There were six observations read from the data set WORK.EMPLOYEE.
NOTE: The data set WORK.EMPLOYEE has six observations and three variables.
NOTE: PROCEDURE SORT used:
      real time           0.01 seconds
      cpu time            0.01 seconds
```

Output 54.2 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 1189.

Output 54.3 Observations Sorted by the Values of Three Variables

Customers with Past-Due Accounts 1
Listed by Town, Amount, Account Number

Obs	Company	Town	Debt	Account Number
1	Paul's Pizza	Apex	83.00	1019
2	Peter's Auto Parts	Apex	65.79	7288
3	Watson Tabor Travel	Apex	37.95	3131
4	Tina's Pet Shop	Apex	37.95	5108
5	Apex Catering	Apex	37.95	9923
6	Deluxe Hardware	Garner	467.12	8941
7	Boyd & Sons Accounting	Garner	312.49	4762
8	World Wide Electronics	Garner	119.95	1122
9	Elway Piano and Organ	Garner	65.79	5217
10	Ice Cream Delight	Holly Springs	299.98	2310
11	Tim's Burger Stand	Holly Springs	119.95	6335
12	Strickland Industries	Morrisville	657.22	1675
13	Pauline's Antiques	Morrisville	302.05	9112
14	Bob's Beds	Morrisville	119.95	4998

Syntax: SORT Procedure

Requirements: BY statement

Tip: You can use the ATTRIB, FORMAT, LABEL, and WHERE statements. See Chapter 3, "Statements with the Same Function in Multiple Procedures," on page 35 for details. You can also use any global statements. See "Global Statements" on page 20 for a list.

See: SORT Procedure under 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)*>;

Task	Option
Specify the collating sequence	
Specify ASCII	ASCII
Specify EBCDIC	EBCDIC

Task	Option
Specify Danish	DANISH
Specify Finnish	FINNISH
Specify Norwegian	NORWEGIAN
Specify Polish	POLISH
Specify Swedish	SWEDISH
Specify a customized sequence	NATIONAL
Specify any of the collating sequences listed above (ASCII, EBCDIC, DANISH, FINNISH, ITALIAN, NORWEGIAN, POLISH, SPANISH, SWEDISH, or NATIONAL), the name of any other system provided translation table (POLISH, SPANISH), and the name of a user-created translation table. You can specify an encoding. You can also specify either the keyword LINGUISTIC or UCA to achieve a locale appropriate collating sequence.	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	
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 whether or not the data set is likely already sorted.	PRESORTED
Reverse the collation order for character variables	REVERSE
Specify the available memory	SORTSIZE=
Force redundant sorting	FORCE
Reduce temporary disk usage	TAGSORT
Override SAS system option THREADS	

Task	Option
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 want to achieve an ASCII ordering on a system where EBCDIC is the native collating sequence.

See also: "Sorting Orders for Character Variables" on page 1182

DANISH
NORWEGIAN
sorts characters according to the Danish and Norwegian convention.

The Danish and Norwegian collating sequence is shown in Figure 54.1 on page 1170.

EBCDIC
sorts character variables using the EBCDIC collating sequence. You need this option only when you want to achieve an EBCDIC ordering on a system where ASCII is the native collating sequence.

See also: "Sorting Orders for Character Variables" on page 1182

POLISH
sorts characters according to the Polish convention.

FINNISH
SWEDISH
sorts characters according to the Finnish and Swedish convention. The Finnish and Swedish collating sequence is shown in Figure 54.1 on page 1170.

NATIONAL
sorts character variables using an alternate collating sequence, as defined by your installation, to reflect a country's National Use Differences. To use this option, your site must have a customized national sort sequence defined. Check with the SAS Installation Representative at your site to determine whether a customized national sort sequence is available.

NORWEGIAN

See DANISH.

SWEDISH
See FINNISH.

SORTSEQ= *collating-sequence*
specifies the collating sequence. The *collating-sequence* can be a collating-sequence-option, a translation table, an encoding, or the keyword LINGUISTIC. Only one collating sequence can be specified. For detailed information, refer to the Collating Sequence section in the *SAS National Language Support (NLS): Reference Guide*.

Here are descriptions of the collating sequences:

collating—sequence—option | translation_table
specifies either a translation table, which can be one that SAS provides or any user-defined translation table, or one of the PROC SORT statement Collating-Sequence-Options. For an example of using PROC TRANTAB and PROC SORT with SORTSEQ=, see Using Different Translation Tables for Sorting in *SAS National Language Support (NLS): Reference Guide*.

The available translation tables are

ASCII
DANISH
EBCDIC
FINNISH
ITALIAN
NORWEGIAN
POLISH
REVERSE
SPANISH
SWEDISH

The following figure shows how the alphanumeric characters in each language will sort.

Figure 54.1 National Collating Sequences of Alphanumeric Characters

```
Danish:     0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZÆØÅabcdefghijklmnopqrstuvwxyzæøå
Finnish:    0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZÅÄÖabcdefghijklmnopqrstuvwxyzåäö
Italian:    0123456789AÀBCÇDEÉÈFGHIÌJKLMNOÒPQRSTUÙVWXYZaàbcçdeéèfghiìjklmnoòpqrstuùvwxyz
Norwegian:  0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZÆØÅabcdefghijklmnopqrstuvwxyzæøå
Spanish:    0123456789AÁaáBbCcDdEÉeéFfGgHhIÍiíJjKkLlMmNnÑñOÓoóPpQqRrSsTtUÚuúÜüVvWwXxYyZz
Swedish:    0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZÅÄÖabcdefghijklmnopqrstuvwxyzåäö
```

Restriction: You can specify only one *collating-sequence-option* in a PROC SORT step.

Interaction: In-database processing will not occur when the SORTSEQ= option is specified.

Tip: The SORTSEQ= collating-sequence options are specified without parenthesis and have no arguments associated with them. An example of how to specify a collating sequence follows:

```
proc sort data=mydata SORTSEQ=ASCII;
```

encoding-value
specifies an encoding value. The result is the same as a binary collation of the character data represented in the specified encoding. See the supported encoding values in the *SAS National Language Support (NLS): Reference Guide*.

Restriction: PROC SORT is the only procedure or part of the SAS system that recognizes an encoding specified for the SORTSEQ= option.

Tip: When the encoding value contains a character other than an alphanumeric character or underscore, the value needs to be enclosed in quotation marks.

See: The list of the encodings that can be specified in the *SAS National Language Support (NLS): Reference Guide*.

LINGUISTIC<*(collating-rules)*>
specifies linguistic collation, which sorts characters according to rules of the specified language. The rules and default collating-sequence options are based on the language specified in the current locale setting. The implementation is provided by the International Components for Unicode (ICU) library. It produces results that are largely compatible with the Unicode Collation Algorithms (UCA).

Alias: UCA

Restriction: The SORTSEQ=LINGUISTIC option is available only on the PROC SORT SORTSEQ= option and is not available for the SAS System SORTSEQ= option.

Restriction Note that linguistic collation is not supported on platforms VMS on Itanium (VMI) or 64-bit Windows on Itanium (W64).

Tip: The *collating-rules* must be enclosed in parentheses. More than one collating rule can be specified.

Tip: When BY processing is performed on data sets that are sorted with linguistic collation, the NOBYSORTED system option might need to be specified in order for the data set to be treated properly. BY processing is performed differently than collating sequence processing.

See: The Appendix 4, "ICU License," on page 1643 agreement.

See: The section on Linguistic Collation in the *SAS National Language Support (NLS): Reference Guide* for detailed information.

See Also: Refer to *http://www.unicode.org* Web site for the Unicode Collation Algorithm (UCA) specification.

The following are the *collation-rules* that can be specified for the LINGUISTIC option. These rules modify the linguistic collating sequence:

ALTERNATE_HANDLING=SHIFTED
controls the handling of variable characters like spaces, punctuation, and symbols. When this option is not specified (using the default value Non-Ignorable), differences among these variable characters are of the same importance as differences among letters. If the ALTERNATE_HANDLING option is specified, these variable characters are of minor importance.

Default: NON_IGNORABLE

Tip: The SHIFTED value is often used in combination with STRENGTH= set to Quaternary. In such a case, spaces, punctuation, and symbols are considered when comparing strings, but only if all other aspects of the strings (base letters, accents, and case) are identical.

CASE_FIRST=

specifies the order of uppercase and lowercase letters. This argument is valid for only TERTIARY, QUATERNARY, or IDENTICAL levels. The following table provides the values and information for the CASE_FIRST argument:

Table 54.1

Value	Description
UPPER	Sorts uppercase letters first, then the lowercase letters.
LOWER	Sorts lowercase letters first, then the uppercase letters.

COLLATION=

The following table lists the available COLLATION= values: If you do not select a collation value, then the user's locale-default collation is selected.

Table 54.2

Value	Description
UPPER	Sorts uppercase letters first, then the lowercase letters.
LOWER	Sorts lowercase letters first, then the uppercase letters.

LOCALE= *locale_name*

specifies the locale name in the form of a POSIX name (for example, ja_JP). See the Values for the LOCALE= System Option in *SAS National Language Support (NLS): Reference Guide* for a list of locale and POSIX values supported by PROC SORT.

Restriction: The following locales are not supported by PROC SORT:

Afrikaans_SouthAfrica, af_ZA

Cornish_UnitedKingdom, kw_GB

ManxGaelic_UnitedKingdom, gv_GB

NUMERIC_COLLATION=

orders integer values within the text by the numeric value instead of characters used to represent the numbers.

Table 54.3

Value	Description
ON	Order numbers by the numeric value. For example, "8 Main St." would sort before "45 Main St.".
OFF	Order numbers by the character value. For example, "45 Main St." would sort before "8 Main St.".

Default: OFF

STRENGTH=

The value of strength is related to the collation level. There are five collation-level values. The following table provides information about the five levels. The default value for strength is related to the locale.

Table 54.4

Value	Type of Collation	Description
PRIMARY or 1	PRIMARY specifies differences between base characters (for example, "a" < "b").	It is the strongest difference. For example, dictionaries are divided into different sections by base character.
SECONDARY or 2	Accents in the characters are considered secondary differences (for example, "as" < "às" < "at").	A secondary difference is ignored when there is a primary difference anywhere in the strings. Other differences between letters can also be considered secondary differences, depending on the language.
TERTIARY or 3	Upper and lowercase differences in characters are distinguished at the tertiary level (for example, "ao" < "Ao" < "aò").	A tertiary difference is ignored when there is a primary or secondary difference anywhere in the strings. Another example is the difference between large and small Kana.
QUATERNARY or 4	When punctuation is ignored at level 1-3, an additional level can be used to distinguish words with and without punctuation (for example, "ab" < "a-b" < "aB").	The quaternary level should be used if ignoring punctuation is required or when processing Japanese text. This difference is ignored when there is a primary, secondary, or tertiary difference.
IDENTICAL or 5	When all other levels are equal, the identical level is used as a tiebreaker. The Unicode code point values of the Normalization Form D (NFD) form of each string are compared at this level, just in case there is no difference at levels 1-4.	This level should be used sparingly, because code-point value differences between two strings rarely occur. For example, only Hebrew cantillation marks are distinguished at this level.

Alias: LEVEL=

CAUTION:
If you use a host sort utility to sort your data, then specifying a translation-table-based collating sequence with the SORTSEQ= option might corrupt the character BY variables. For more information, see the PROC SORT documentation for your operating environment. △

Other Options

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

Restriction: For in-database processing to occur, it is necessary that the data set refer to a table residing on the DBMS.

Main discussion: "Input Data Sets" on page 20

DATECOPY
copies the SAS internal date and time when the SAS data set was created and the date and time when it was last modified before the sort to the resulting sorted data set. Note that the operating environment date and time are not preserved.

Restriction: DATECOPY can be used only when the resulting data set uses the V8 or V9 engine.

Tip: You can alter the file creation date and time with the DTC= option in the MODIFY statement in PROC DATASETS. For more information, see "MODIFY Statement" on page 342.

DUPOUT= *SAS-data-set*
specifies the output data set to which duplicate observations are written.

Interaction: In-database processing does not occur when the DUPOUT= option is specified.

Tip: If the DUPOUT= data set name that is specified is the same as the INPUT data set name, SAS will not sort or overwrite the INPUT data set. Instead, SAS will generate an error message. The FORCE option must be specified in order to overwrite the INPUT data set with the DUPOUT= data set of the same name.

EQUALS | NOEQUALS
specifies the order of the observations in the output data set. For observations with identical BY-variable values, EQUALS maintains the relative order of the observations within the input data set in the output data set. NOEQUALS does not necessarily preserve this order in the output data set.

Default: EQUALS

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 might be reduced when using the EQUALS option with the multi-threaded sort because partitioned data sets will be processed as if they consist of a single partition.

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.

Tip: Using NOEQUALS can save CPU time and memory.

FORCE
sorts and replaces an indexed data set when the OUT= option is not specified. Without the FORCE option, PROC SORT does not sort and replace an indexed data set because sorting destroys user-created indexes for the data set. When you specify FORCE, PROC SORT sorts and replaces the data set and destroys all user-created indexes for the data set. Indexes that were created or required by integrity constraints are preserved.

Restriction: If you use PROC SORT with the FORCE option on data sets that were created with the Version 5 compatibility engine or with a sequential engine such as a tape format engine, you must also specify the OUT= option.

Tip: PROC SORT checks for the sort indicator before it sorts a data set so that data is not sorted again unnecessarily. By default, PROC SORT does not sort a data set

if the sort information matches the requested sort. You can use FORCE to override this behavior. You might need to use FORCE if SAS cannot verify the sort specification in the data set option SORTEDBY=. For more information about SORTEDBY=, see the chapter on SAS data set options in *SAS Language Reference: Dictionary*.

NODUPKEY
checks for and eliminates observations with duplicate BY values. If you specify this option, then PROC SORT compares all BY values for each observation to the ones for the previous observation that is written to the output data set. If an exact match is found, then the observation is not written to the output data set.

Operating Environment Information: If you use the VMS operating environment and are using the VMS host sort, the observation that is written to the output data set is not always the first observation of the BY group. △

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.

Interaction: In-database sorting occurs when the NODUPKEY option is specified and the system option SQLGENERATION= is assigned a DBMS and the system option SORTPGM=BEST.

Tip: Use the EQUALS option with the NODUPKEY option for consistent results in your output data sets.

Featured in: Example 4 on page 1193

NODUPRECS
checks for and eliminates duplicate observations. If you specify this option, then PROC SORT compares all variable values for each observation to the ones 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.

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*.

Interaction: In-database processing does not occur when the NODUPRECS option is specified. However, if the NODUPRECS and NODUPKEY options are specified, system option SQLGENERATION= set for in-database processing, and system

option SORTPGM=BEST, the NODUPRECS option is ignored and in-database processing does occur.

Tip: Use the EQUALS option with the NODUPRECS option for consistent results in your output data sets.

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 the OUT= option, PROC SORT replaces the original data set with the sorted observations when the procedure executes without errors. △

Default: Without OUT=, PROC SORT overwrites the original data set.

Tip: With in-database sorts, the output data set cannot refer to the input table on the DBMS.

Tip: You can use data set options with OUT=.

Featured in: Example 1 on page 1187

OVERWRITE
enables the input data set to be deleted before the replacement output data set is populated with observations.

Restriction: The OVERWIRTE option has no effect when an OUT= data set is specified.

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. △

PRESORTED
before sorting, checks within the input data set to determine whether the sequence of observations are in order. Use the PRESORTED option when you know or strongly suspect that a data set is already in order according to the key variables that are specified in the BY statement. By specifying this option, you avoid the cost of sorting the data set.

Note: See the NODUPRECS option for information about eliminating duplicate observations. △

Interaction: Sequence checking is not performed when the FORCE option is specified.

Interaction: When the NODUPRECS option has been specified and one or more variables have been dropped from the input data set, then the SORTDUP system option setting will affect the detection of adjacent duplicate observations.

Tip: You can use the DATA step to import data, from external text files, in a sequence compatible with SAS processing and according to the sort order specified by the combination of SORT options and key variables listed in the BY statement. You can then specify the PRESORTED option if you know or highly suspect that the data is sorted accordingly.

Tip: Using the PRESORTED option with ACCESS engines and DBMS data is not recommended. These external databases are not guaranteed to return observations in sorted order unless an ORDER BY clause is specified in a query. Generally, physical ordering is not a concept that external databases use. Therefore, these databases are not guaranteed to return observations in the same order when executing a query multiple times. Physical order can be important for producing consistent, repeatable results when processing data. Without a repeatable data retrieval order, PROC SORT does not guarantee the return of observations in the same order from one PROC SORT execution to another, even when the EQUALS option is used to request sort stability. Without a repeatable retrieval order, the detection and elimination of adjacent duplicate records (requested by specifying the NODUPRECS option) by PROC SORT can also vary from one PROC SORT execution to another.

See also: System option SORTVALIDATE in *SAS Language Reference: Dictionary*

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 1182, "ASCII Order" on page 1183, and the SAS documentation for your operating environment. △

Restriction: The REVERSE option cannot be used with a *collating-sequence-option*. You can specify either a *collating-sequence-option* or the REVERSE option in a PROC SORT, but you cannot specify both.

Interaction: Using REVERSE with the DESCENDING option in the BY statement restores the sequence to the normal order.

See 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 might treat this option differently. Refer to the SAS documentation for your operating environment. △

Alias: SIZE=

Default: the value of the SAS system option SORTSIZE=

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 REALMEMSIZE and MEMSIZE. .

Operating Environment Information: For information about the SAS system options REALMEMSIZE and MEMSIZE, see the SAS documentation for your operating environment. △

TAGSORT

stores only the BY variables and the observation numbers in temporary files. The BY variables and the observation numbers are called *tags*. At the completion of the sorting process, PROC SORT uses the tags to retrieve records from the input data set in sorted order.

Note: The utility file created is much smaller than it would be if the TAGSORT option were not specified. △

Restriction: The TAGSORT option is not compatible with the OVERWRITE option.

Interaction: The TAGSORT option is not supported by the multi-threaded sort.

Tip: When the total length of BY variables is small compared with the record length, TAGSORT reduces temporary disk usage considerably. However, processing time might be much higher.

THREADS | NOTHREADS

enables or prevents the activation of multi-threaded sorting.

Default: the value of the THREADS | NOTHREADS SAS system option.

Note: The default can be overridden using the procedure THREADS | NOTHREADS option. △

Restriction: Your site administrator can create a restricted options table. A restricted options table specifies SAS system option values that are established at startup and cannot be overridden. If the THREADS | NOTHREADS system option is listed in the restricted options table, any attempt to set these system options is ignored and a warning message is written to the SAS log.

Restriction: If a failure occurs when adding the THREADS | NOTHREADS procedure option using the SPD engine, PROC SORT stops processing and writes a message to the SAS log.

Interaction: The PROC SORT THREADS | NOTHREADS options override the SAS system THREADS | NOTHREADS options unless the system option is restricted. (See Restriction.) For more information about THREADS, see the chapter on SAS system options in *SAS Language Reference: Dictionary*.

Interaction: The THREADS system option is honored if PROC SORT determines that multi-threaded processing is deemed to be beneficial. If the value of the SAS system option CPUCOUNT=1, then multi-threaded processing is not beneficial. However, you can specify the PROC SORT THREADS option to force multi-threaded processing when the system option is set to NOTHREADS or when the system option is THREADS and the procedure option is NOTHREADS. This

option combination prevents multi-threaded processing and overrides the actions taken that are based on the system options.

Note: When multi-threaded sorting is in effect and NOEQUALS is specified, observations within BY groups might be returned in an unpredictable order. △

Interaction: If multi-threaded SAS sort is being used, the UTILLOC= system option will affect the placement of utility files. Thread-enabled SAS applications are able to create temporary files that can be accessed in parallel by separate threads. For more information about the UTILLOC= system option, see the chapter on SAS system options in *SAS Language Reference: Dictionary*.

Interaction: The TAGSORT option is not supported by the multi-threaded sort. Specifying the TAGSORT option will prevent multi-threaded processing.

See also: "Multi-threaded Sorting" on page 1181 and Support for Parallel Processing in *SAS Language Reference: Concepts*.

BY Statement

Specifies the sorting variables.

Featured in:

Example 1 on page 1187
Example 2 on page 1189
Example 4 on page 1193

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. The DESCENDING keyword modifies the variable that follows it.

Tip: In a PROC SORT BY statement, the DESCENDING keyword modifies the variable that follows it.

Tip: The THREADS SAS system option is the default as long as the PROC SORT THREADS | NOTHREADS option is unspecified.

Featured in: Example 2 on page 1189

KEY Statement

Specifies sorting keys and variables. The KEY statement is an alternative to the BY statement. The KEY statement syntax allows for the future possibility of specifying different collation options for each KEY variable. Currently, the only options allowed are ASCENDING and DESCENDING.

Restriction: The BY statement cannot be used with the KEY statement.

Tip: Multiple KEY statements can be specified.

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

Required Arguments

variable(s)
specifies the variable by which PROC SORT orders the observations. Multiple variables can be specified. Each of these variables must be separated by a space. A range of variables can also be specified. For example, the following code shows how to specify multiple variables and a range of variables:

```
data sortKeys;
   input x1 x2 x3 x4 ;
cards;
   7 8 9 8
   0 0 0 0
   1 2 3 4
;
run;

proc sort data=sortKeys out=sortedOutput;
   key x1 x2-x4;
run;
```

Multiple KEY statements can also be specified. The first sort key encountered from among all sort keys is considered the primary sort key. Sorting continues for every specified KEY statement and its variables. For example, the following code shows how to specify multiple KEY statements:

```
proc sort data=sortKeys out=sortedOutput;
   key x2;
   key x3;
  run;
```

The following code example uses the BY statement to accomplish the same type of sort as the previous example:

```
proc sort data=sortKeys out=sortedOutput;
   by x2 x3;
run;
```

Options

ASCENDING
sorts in ascending order the variable or variables that it follows. Observations are sorted from the smallest value to the largest value. The ASCENDING keyword modifies all the variables that precede it in the KEY statement.

Alias: ASC

Default: ASCENDING is the default sort order.

Tip: In a PROC SORT KEY statement, the ASCENDING option modifies all the variables that it follows. The option must follow the `/`. In the following example, the x1 variable in the input data set will be sorted in ascending order.

```
proc sort data=sortVar out=sortedOutput;
   key x1 / ascending;
run;
```

DESCENDING
reverses the sort order for the variable that it follows in the statement so that observations are sorted from the largest value to the smallest value. The DESCENDING keyword modifies all the variables that it precede in the KEY statement.

Alias: DESC

Default: ASCENDING (ASC) is the default sort order.

Tip: In a PROC SORT KEY statement, the DESCENDING option modifies the variables that follows it. The option must follow the `/`. In the following example, the x1 and x2 variables in the input data set will be sorted in descending order:

```
proc sort data=sortVar out=sortedOutput;
   key x1 x2 / descending;
run;
```

The following example uses the BY statement to accomplish the same type of sort as the previous example:

```
proc sort data=sortVar out=sortedOutput;
   by descending x1 descending x2 ;
run;
```

Concepts: SORT Procedure

Multi-threaded Sorting

The SAS system option THREADS permits multi-threaded sorting, which is new with SAS System 9. Multi-threaded sorting achieves a degree of parallelism in the sorting operations. This parallelism is intended to reduce the real time to completion for a given operation and therefore limit the cost of additional CPU resources. For more information, see the section on "Support for Parallel Processing" in *SAS Language Reference: Concepts*.

Note: The TAGSORT option does not support multi-threaded sorting. △

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 procedures.

For more information about THREADS and CPUCOUNT=, see the chapter on SAS system options in *SAS Language Reference: Dictionary*.

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

The order in which alphanumeric characters are sorted is known as the collating sequence. This sort order is determined by the session encoding.

By default, PROC SORT uses either the EBCDIC or the ASCII collating sequence when it compares character values, depending on the environment under which the procedure is running.

Refer to the Collating Sequence chapter of the *SAS National Language Support (NLS): Reference Guide* for detailed information about the various collating sequences and when they are used.

Note: ASCII and EBCDIC represent the family names of the session encodings. The sort order can be determined by referring to the encoding. △

EBCDIC Order

The z/OS operating environment uses the EBCDIC collating sequence.

The sorting order of the English-language EBCDIC sequence is consistent with the following sort order example.

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
- Windows
- OpenVMS.

From the smallest to the largest character that you can display, the English-language ASCII sequence is consistent with the order shown in the following:

```
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): Reference 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.

Linguistic Collation, which sorts data according to rules of language, is supported in SAS System 9.2. Refer to the Collating Sequence chapter in *SAS National Language Support (NLS): Reference Guide* for detailed information about Linguistic Collation.

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 342.

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 314.

Presorted Input Data Sets

A new option, PRESORTED, has been added to the PROC SORT statement in the 9.2 version of SAS. Specifying the PRESORTED options prevents SAS from sorting an already sorted data set. Before sorting, SAS checks the sequence of observations within the input data set to determine whether the observations are in order. Use the PRESORTED option when you know or strongly suspect that a data set is already in order according to the key variables specified in the BY statement. The sequence of observations within the data set is checked by reading the data set and comparing the BY variables of each observation read to the BY variables of the preceding observation. This process continues until either the entire data set has been read or an out-of-sequence observation is detected.

If the entire data set has been read and no out-of-sequence observations have been found, then one of two actions is taken. If no output data set has been specified, the sort order metadata of the input data set is updated to indicate that the sequence has been verified. This verification notes that the data set is validly sorted according to the specified BY variables. Otherwise, if the observation sequence has been verified and an output data set is specified, the observations from the input data set are copied to the output data set, and the metadata for the output data set indicates that the data is validly sorted according to the BY variables.

If observations within the data set are not in sequence, then the data set will be sorted.

If the NODUPKEY option has been specified, then the sequence checking determines whether observations with duplicate keys are present in the data set. Otherwise, if the NODUPRECS option has been specified, then the sequence checking determines whether there are adjacent duplicate observations. The input data set is deemed not to be sorted if the NODUPKEY option is specified and observations with duplicate keys are detected. Likewise, the input data set is deemed not to be sorted if the NODUPRECS option is specified and adjacent duplicate observations are detected.

If the metadata of the input data set indicates that the data is already sorted according to the key variables listed in the BY statement and the input data set has been validated, then neither sequence checking nor sorting will be performed.

See Sorted Data Sets in the *SAS Language Reference: Concepts* and interactions with the SORTVALIDATE system option in *SAS Language Reference: Dictionary*.

In-Database Processing: PROC SORT

When the DATA= input data set is stored as a table or view in a database management system (DBMS), the PROC SORT procedure can use in-database processing to sort the data. In-database processing can provide the advantages of faster processing and reduced data transfer between the database and SAS software.

PROC SORT performs in-database processing using SQL explicit pass-through. The Pass-Through Facility uses SAS/ACCESS to connect to a DBMS and to send statements directly to the DBMS for execution. This facility lets you use the SQL syntax of your DBMS. For details, see "Pass-Through Facility for Relational Databases" in *SAS/ACCESS for Relational Databases: Reference*.

In the third maintenance release for SAS 9.2, in-database processing is used by PROC SORT when a combination of procedure and system options are properly set. When system option SORTPGM=BEST, system option SQLGENERATION= is set to cause in-database processing, and when the PROC SORT NODUPKEY option is specified, PROC SORT generates a DBMS SQL query that sorts the data. The sorted results can either remain as a new table within the DBMS or can be returned to SAS. To view the SQL queries generated, set the SASTRACE= option.

The SAS System Option SORTPGM= can also be used without setting the SQLGENERATION option to instruct PROC SORT to use either the DBMS, SAS, or the HOST to perform the sort. If SORTPGM=BEST is specified, then either the DBMS, SAS, or HOST will perform the sort. The observation ordering that is produced by PROC SORT will depend on whether the DBMS or SAS performs the sorting.

If the DBMS performs the sort, then the configuration and characteristics of the DBMS sorting program will affect the resulting data order. The DBMS configuration settings and characteristics that can affect data order include character collation, ordering of NULL values, and sort stability. Most database management systems do not guarantee sort stability, and the sort might be performed by the DBMS regardless of the state of the SORTEQUALS/NOSORTEQUALS system option and EQUALS/ NOEQUALS procedure option.

If you set the SAS system option SORTPGM= to SAS, then unordered data is delivered from the DBMS to SAS and SAS performs the sorting. However, consistency in the delivery order of observations from a DBMS is not guaranteed. Therefore, even though SAS can perform a stable sort on the DBMS data, SAS cannot guarantee that the ordering of observations within output BY groups will be the same from one PROC SORT execution to the next. To achieve consistency in the ordering of observations within BY groups, first populate a SAS data set with the DBMS data, then use the EQUALS or SORTEQUALS option to perform a stable sort.

In-database processing is affected by the following circumstances:

- When any of the PROC SORT options, NODUPRECS, SORTSEQ=, or DUPOUT=, are specified, no in-database processing occurs.
- For in-database processing, the OUT= procedure option must be specified and the output data set cannot refer to the input table on the DBMS.
- If the SQLGENERATION system option is set to cause in-database processing and both the NODUPKEY and NODUPRECS procedure options are specified, the NODUPRECS option is ignored and in-database processing occurs.
- LIBNAME options and data set options can also affect whether or not in-database processing occurs and what type of query will be generated. See "Overview of In-Database Procedures" in *SAS/ACCESS for Relational Databases: Reference* for a complete list of these options. The user can also set OPTIONS MSGLEVEL=I in SAS to see which options prevent or affect in-database processing.

Integrity Constraints: SORT Procedure

Sorting the input data set and replacing it with the sorted data set preserves both referential and general integrity constraints, as well as any indexes that they might require. A sort that creates a new data set will not preserve any integrity constraints or indexes. For more information about implicit replacement, explicit replacement, and no replacement with and without the OUT= option, see “Output Data Set” on page 1186. 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.

Task	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 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*.

Also note that PROC SORT manipulates the uncompressed observation in memory and, if there is insufficient memory to complete the sort, stores the uncompressed data in a utility file. For these reasons, sorting compressed data sets might be intensive and require more storage than anticipated. Consider using the TAGSORT option when sorting compressed data sets.

Note: If the SAS system option NOREPLACE is in effect, then you cannot replace an original permanent data set with a sorted version. You must either use the OUT= option or specify the SAS system option REPLACE in an OPTIONS statement. The SAS system option NOREPLACE does not affect temporary SAS data sets. △

Examples: SORT Procedure

Example 1: Sorting by the Values of Multiple Variables

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 1187

- □ 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

Obs	Company	Town	Debt	Account Number
1	Paul's Pizza	Apex	83.00	1019
2	Peter's Auto Parts	Apex	65.79	7288
3	Watson Tabor Travel	Apex	37.95	3131
4	Tina's Pet Shop	Apex	37.95	5108
5	Apex Catering	Apex	37.95	9923
6	Deluxe Hardware	Garner	467.12	8941
7	Boyd & Sons Accounting	Garner	312.49	4762
8	World Wide Electronics	Garner	119.95	1122
9	Elway Piano and Organ	Garner	65.79	5217
10	Ice Cream Delight	Holly Springs	299.98	2310
11	Tim's Burger Stand	Holly Springs	119.95	6335
12	Strickland Industries	Morrisville	657.22	1675
13	Pauline's Antiques	Morrisville	302.05	9112
14	Bob's Beds	Morrisville	119.95	4998

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

Obs	Years Worked	Insurance 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

Obs	Years Worked	Insurance 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 1187

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.

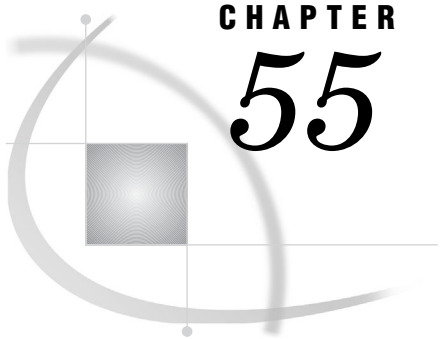
```
   title 'Towns of Customers with Past-Due Accounts';
run;
```

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

Obs	Town	Company	Debt	Account 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

55

The SQL Procedure

Overview: SQL Procedure **1199**
What Is the SQL Procedure? **1199**
What Are PROC SQL Tables? **1199**
What Are Views? **1199**
SQL Procedure Coding Conventions **1200**
Syntax: SQL Procedure **1201**
PROC SQL Statement **1204**
ALTER TABLE Statement **1213**
CONNECT Statement **1217**
CREATE INDEX Statement **1218**
CREATE TABLE Statement **1219**
CREATE VIEW Statement **1223**
DELETE Statement **1226**
DESCRIBE Statement **1227**
DISCONNECT Statement **1228**
DROP Statement **1228**
EXECUTE Statement **1229**
INSERT Statement **1230**
RESET Statement **1232**
SELECT Statement **1233**
UPDATE Statement **1245**
VALIDATE Statement **1246**
SQL Procedure Component Dictionary **1247**
BETWEEN condition **1247**
BTRIM function **1247**
CALCULATED **1248**
CASE expression **1249**
COALESCE Function **1250**
column-definition **1251**
column-modifier **1252**
column-name **1254**
CONNECTION TO **1255**
CONTAINS condition **1255**
EXISTS condition **1256**
IN condition **1256**
IS condition **1257**
joined-table **1258**
LIKE condition **1268**
LOWER function **1270**
query-expression **1270**
sql-expression **1277**

SUBSTRING function **1284**
summary-function **1285**
table-expression **1292**
UPPER function **1293**
PROC SQL and the ANSI Standard **1293**
Compliance **1293**
SQL Procedure Enhancements **1293**
Reserved Words **1293**
Column Modifiers **1294**
Alternate Collating Sequences **1294**
ORDER BY Clause in a View Definition **1294**
CONTAINS Condition **1294**
In-Line Views **1294**
Outer Joins **1294**
Arithmetic Operators **1295**
Orthogonal Expressions **1295**
Set Operators **1295**
Statistical Functions **1295**
SAS DATA Step Functions **1295**
PROC FCMP Functions **1295**
SQL Procedure Omissions **1296**
COMMIT Statement **1296**
ROLLBACK Statement **1296**
Identifiers and Naming Conventions **1296**
Granting User Privileges **1296**
Three-Valued Logic **1296**
Embedded SQL **1296**
Examples: SQL Procedure **1296**
Example 1: Creating a Table and Inserting Data into It **1296**
Example 2: Creating a Table from a Query's Result **1299**
Example 3: Updating Data in a PROC SQL Table **1300**
Example 4: Joining Two Tables **1303**
Example 5: Combining Two Tables **1305**
Example 6: Reporting from DICTIONARY Tables **1307**
Example 7: Performing an Outer Join **1309**
Example 8: Creating a View from a Query's Result **1313**
Example 9: Joining Three Tables **1316**
Example 10: Querying an In-Line View **1319**
Example 11: Retrieving Values with the SOUNDS-LIKE Operator **1320**
Example 12: Joining Two Tables and Calculating a New Value **1322**
Example 13: Producing All the Possible Combinations of the Values in a Column **1324**
Example 14: Matching Case Rows and Control Rows **1328**
Example 15: Counting Missing Values with a SAS Macro **1331**

Overview: SQL Procedure

What Is the SQL Procedure?

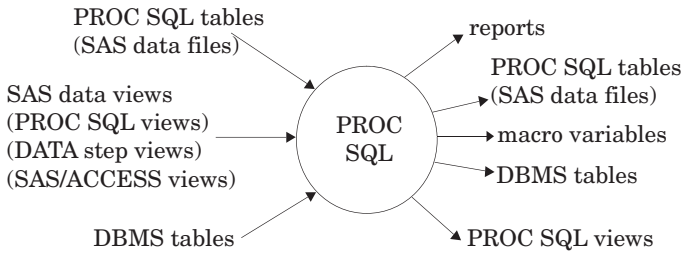
The SQL procedure implements Structured Query Language (SQL) for SAS. SQL is a standardized, widely used language that retrieves data from and updates data in tables and the views that are based on those tables.

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 55.1 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 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" in the *SAS 9.2 SQL Procedure User's Guide*. The SAS/ACCESS LIBNAME statement is described in "Connecting to a DBMS Using the LIBNAME Statement" in the *SAS 9.2 SQL Procedure User's Guide*.

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 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 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 tables or views.

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" in the *SAS 9.2 SQL Procedure User's Guide*.

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 1223 for details. △

Note: When you process PROC SQL views between a client and a server, getting the correct results depends on the compatibility between the client and server architecture. For more information, see "Accessing a SAS View" in the *SAS/CONNECT User's Guide*. △

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: Basic Concepts in *SAS Output Delivery System: User's Guide* for details.

Tip: You can use any global statements. See Chapter 2, "Fundamental Concepts for Using Base SAS Procedures," on page 17 for a list.

Tip: 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" in *SAS 9.2 SQL Procedure User's Guide* for details.

Tip:

Regular type indicates the name of a component that is described in "SQL Procedure Component Dictionary" on page 1247.

view-name indicates a SAS 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**;>

Task	Statement
Create, maintain, retrieve, and update data in tables and views that are based on these tables	“PROC SQL Statement” on page 1204
Modify, add, or drop columns	“ALTER TABLE Statement” on page 1213
Establish a connection with a DBMS	“CONNECT Statement” on page 1217
Create an index on a column	“CREATE INDEX Statement” on page 1218
Create a PROC SQL table	“CREATE TABLE Statement” on page 1219
Create a PROC SQL view	“CREATE VIEW Statement” on page 1223
Delete rows	“DELETE Statement” on page 1226
Display a definition of a table or view	“DESCRIBE Statement” on page 1227
Terminate the connection with a DBMS	“DISCONNECT Statement” on page 1228
Delete tables, views, or indexes	“DROP Statement” on page 1228
Send a DBMS-specific nonquery SQL statement to a DBMS	“EXECUTE Statement” on page 1229
Add rows	“INSERT Statement” on page 1230
Reset options that affect the procedure environment without restarting the procedure	“RESET Statement” on page 1232
Select and execute rows	“SELECT Statement” on page 1233
Query a DBMS	“CONNECTION TO” on page 1255

Task	Statement
Modify values	“UPDATE Statement” on page 1245
Verify the accuracy of your query	“VALIDATE Statement” on page 1246

PROC SQL Statement

PROC SQL <*option(s)*>;

Task	Option
Control output	
Specify the buffer page size for the output	BUFFERSIZE= on page 1205
Double-space the report	DOUBLE|NODOUBLE on page 1206
Write a statement to the SAS log that expands the query	FEEDBACK|NOFEEDBACK on page 1207
Flow characters within a column	FLOW|NOFLOW on page 1207
Include a column of row numbers	NUMBER|NONUMBER on page 1208
Specify whether PROC SQL prints the query’s result	PRINT|NOPRINT on page 1209
Specify whether PROC SQL should display sorting information	SORTMSG|NOSORTMSG on page 1211
Specify a collating sequence	SORTSEQ= on page 1211
Control execution	
Specify whether PROC SQL replaces references to the DATE, TIME, DATETIME, and TODAY functions in a query with their equivalent constant values before the query executes	CONSTDATETIME|NOCONSTDATETIME on page 1205
Allow PROC SQL to use names other than SAS names	DQUOTE= on page 1206
Specify whether PROC SQL should stop executing after an error	ERRORSTOP|NOERRORSTOP on page 1206
Specify whether PROC SQL should execute statements	EXEC|NOEXEC on page 1206
Specify whether PROC SQL clears an error code for the SQLEXITCODE macro variable for each statement	EXITCODE on page 1207
Restrict the number of input rows	INOBS= on page 1207
Specify whether implicit pass–through is enabled or disabled	IPASSTHRU|NOIPASSTHRU on page 1207

Task	Option
Restrict the number of loops	LOOPS= on page 1207
Restrict the number of output rows	OUTOBS= on page 1208
Specify whether PROC SQL prompts you when a limit is reached with the INOBS=, OUTOBS=, or LOOPS= options	PROMPT|NOPROMPT on page 1209
Specify the engine type that a query uses for which optimization is performed by replacing a PUT function in a query with a logically equivalent expression	REDUCEPUT= on page 1209
When the REDUCEPUT= option is set to NONE, specifies the minimum number of observations that must be in a table in order for PROC SQL to consider optimizing the PUT function in a query	REDUCEPUTOBS= on page 1209
When the REDUCEPUT= option is set to NONE, specifies the maximum number of SAS format values that can exist in a PUT function expression in order for PROC SQL to consider optimizing the PUT function in a query	REDUCEPUTVALUES= on page 1210
Specify whether PROC SQL processes queries that use remerging of data	REMERGE|NOREMERGE on page 1211
Specify whether PROC SQL writes timing information for each statement to the SAS log	STIMER|NOSTIMER on page 1211
Override the SAS system option THREADS|NOTHREADS	THREADS|NOTHREADS on page 1212
Specify how PROC SQL handles updates when there is an interruption	UNDO_POLICY= on page 1212

Options

BUFFERSIZE=n|nK|nM|nG
specifies the permanent buffer page size for the output in multiples of 1 (bytes), 1024 (kilobytes), 1,048,576 (megabytes), or 1,073,741,824 (gigabytes). For example, a value of 65536 specifies a page size of `65536` bytes and a value of `64k` specifies a page size of 65536 bytes.

BUFFERSIZE can also be specified in a RESET statement for use in particular queries.

Default: 0, which causes SAS to use the minimum optimal page size for the operating environment.

CONSTDATETIME|NOCONSTDATETIME
specifies whether the SQL procedure replaces references to the DATE, TIME, DATETIME, and TODAY functions in a query with their equivalent constant values before the query executes. Computing these values once ensures consistency of

results when the functions are used multiple times in a query or when the query executes the functions close to a date or time boundary.

When the NOCONSTDATETIME option is set, PROC SQL evaluates these functions in a query each time it processes an observation.

Default: CONSTDATETIME

Interaction: If both the CONSTDATETIME option and the REDUCEPUT= option are specified, PROC SQL replaces the DATE, TIME, DATETIME, and TODAY functions with their respective values in order to determine the PUT function value before the query executes.

Tip: Alternatively, you can set the SQLCONSTDATETIME system option. The value that is specified in the SQLCONSTDATETIME system option is in effect for all SQL procedure statements, unless the PROC SQL CONSTDATETIME option is set. The value of the CONSTDATETIME option takes precedence over the SQLCONSTDATETIME system option. The RESET statement can also be used to set or reset the CONSTDATETIME option. However, changing the value of the CONSTDATETIME option does not change the value of the SQLCONSTDATETIME system option. For more information, see the SQLCONSTDATETIME system option in the *SAS Language Reference: Dictionary*.

DOUBLE|NODOUBLE

double-spaces the report.

Default: NODOUBLE

Featured in: Example 5 on page 1305

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 1206

EXITCODE
specifies whether PROC SQL clears an error code for any SQL statement. Error codes are assigned to the SQLEXITCODE macro variable.

Default: 0

Tip: The exit code can be reset to the default value between PROC SQL statements with the "RESET Statement" on page 1232.

See Also: "Using the PROC SQL Automatic Macro Variables" in *SAS 9.2 SQL Procedure User's Guide*.

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.

IPASSTHRU|NOIPASSTHRU
specifies whether implicit pass through is enabled or disabled.

Implicit pass through is enabled when PROC SQL is invoked. You can disable it for a query or series of queries. The primary reasons you might want to disable implicit pass through are as follows:

- DBMSs use SQL2 semantics for NULL values, which behave somewhat differently than SAS missing values.
- PROC SQL might do a better job of query optimization.

Default: IPASSTHRU

See Also: The documentation on the Pass-Through Facility for your DBMS in *SAS/ACCESS for Relational Databases: Reference*.

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 the PROC SQL Automatic Macro Variables" in the *SAS 9.2 SQL Procedure User's Guide*

NOCONSTDATETIME
See CONSTDATETIME|NOCONSTDATETIME on page 1205.

NODOUBLE
See DOUBLE|NODOUBLE on page 1206.

NOERRORSTOP
See ERRORSTOP|NOERRORSTOP on page 1206.

NOEXEC
See EXEC|NOEXEC on page 1206.

NOFEEDBACK
See FEEDBACK|NOFEEDBACK on page 1207.

NOFLOW
See FLOW|NOFLOW on page 1207.

NOIPASSTHRU
See IPASSTHRU|NOIPASSTHRU on page 1207.

NONUMBER
See NUMBER|NONUMBER on page 1208.

NOPRINT
See PRINT|NOPRINT on page 1209.

NOPROMPT
See PROMPT|NOPROMPT on page 1209.

NOREMERGE
See REMERGE|NOREMERGE on page 1211.

NOSORTMSG
See SORTMSG|NOSORTMSG on page 1211.

NOSTIMER
See STIMER|NOSTIMER on page 1211.

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 1303

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

Interaction: NOPRINT affects the value of the SQLOBS automatic macro variable. See "Using the PROC SQL Automatic Macro Variables" in the *SAS 9.2 SQL Procedure User's Guide* for details.

Tip: NOPRINT is useful when you are selecting values from a table into macro variables and do not want anything to be displayed.

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

REDUCEPUT=ALL|NONE|DBMS|BASE
specifies the engine type that a query uses for which optimization is performed by replacing a PUT function in a query with a logically equivalent expression. The engine type can be one of the following values.

ALL
specifies that optimization is performed on all PUT functions regardless of the engine that is used by the query to access the data.

NONE
specifies that no optimization is to be performed.

DBMS
specifies that optimization is performed on all PUT functions whose query is performed by a SAS/ACCESS engine.

Requirement: The first argument to the PUT function must be a variable that is obtained by a table that is accessed using a SAS/ACCESS engine.

BASE
specifies that optimization is performed on all PUT functions whose query is performed by a SAS/ACCESS engine or a Base SAS engine.

Default: DBMS

Interaction: If both the REDUCEPUT= option and the CONSTDATETIME option are specified, PROC SQL replaces the DATE, TIME, DATETIME, and TODAY functions with their respective values in order to determine the PUT function value before the query executes.

Interaction: If the query also contains a WHERE or HAVING clause, the evaluation of the WHERE or HAVING clause is simplified.

Tip: Alternatively, you can set the SQLREDUCEPUT system option. The value that is specified in the SQLREDUCEPUT system option is in effect for all SQL procedure statements, unless the PROC SQL REDUCEPUT= option is set. The value of the REDUCEPUT= option takes precedence over the SQLREDUCEPUT system option. The RESET statement can also be used to set or reset the REDUCEPUT= option. However, changing the value of the REDUCEPUT= option does not change the value of the SQLREDUCEPUT system option. For more information, see the SQLREDUCEPUT system option in the *SAS Language Reference: Dictionary*.

REDUCEPUTOBS=n | nK |nM |nG | nT | hexX | MIN | MAX
when the REDUCEPUT= option is set to NONE, specifies the minimum number of observations that must be in a table in order for PROC SQL to consider optimizing

the PUT function in a query. The number of observations can be one of the following values:

n | *n*K | *n*M | *n*G | *n*T
specifies the number of observations that must be in a table before the SQL procedure considers to optimize the PUT function. **n** is an integer that can be allocated in multiples of 1 (bytes); 1,024 (kilobytes); 1,048,576 (megabytes); 1,073,741,824 (gigabytes); or 1,099,511,627,776 (terabytes). For example, a value of **8** specifies eight buffers, and a value of 3k specifies 3,072 buffers.

Default: 0, which indicates that there is no minimum number of observations in a table required for PROC SQL to optimize the PUT function.

Range: $0 - 2^{63}$-1

*hex*X
specifies the number of observations that must be in a table before the SQL procedure considers to optimize the PUT function as a hexadecimal value. You must specify the value beginning with a number (0-9), followed by an X. For example, the value **2dx** specifies 45 buffers.

MIN
sets the number of observations that must be in a table before the SQL procedure considers to optimize the PUT function to 0. A value of 0 indicates that there is no minimum number of observations required. This value is the default.

MAX
sets the maximum number of observations that must be in a table before the SQL procedure considers to optimize the PUT function to 2^{63}-1, or approximately 9.2 quintillion.

Default: 0

Tip: Alternatively, you can set the SQLREDUCEPUTOBS system option. The value that is specified in the SQLREDUCEPUTOBS system option is in effect for all SQL procedure statements, unless the PROC SQL REDUCEPUTOBS= option is set. The value of the REDUCEPUTOBS= option takes precedence over the SQLREDUCEPUTOBS system option. The RESET statement can also be used to set or reset the REDUCEPUTOBS= option. However, changing the value of the REDUCEPUTOBS= option does not change the value of the SQLREDUCEPUTOBS system option. For more information, see the SQLREDUCEPUTOBS system option in the *SAS Language Reference: Dictionary*.

REDUCEPUTVALUES=n | nK |nM |nG | nT | hexX | MIN | MAX
when the REDUCEPUT= option is set to NONE, specifies the maximum number of SAS format values that can exist in a PUT function expression in order for PROC SQL to consider optimizing the PUT function in a query.

n | *n*K | *n*M | *n*G | *n*T
specifies the number of SAS format values that can exist in a PUT function expression, where **n** is an integer that can be allocated in multiples of 1 (bytes); 1,024 (kilobytes); 1,048,576 (megabytes); 1,073,741,824 (gigabytes); or 1,099,511,627,776 (terabytes). For example, a value of **8** specifies eight buffers, and a value of 3k specifies 3,072 buffers.

Default: 0, which indicates that there is no minimum number of SAS format values that can exist in a PUT function expression.

Range: 0 – 5,000

Interaction: If the number of format values in a PUT function expression is greater than this value, the SQL procedure does not optimize the PUT function.

*hex*X

specifies the number of SAS format values that can exist in a PUT function expression as a hexadecimal value. You must specify the value beginning with a number (0-9), followed by an X. For example, the value **`2dx`** specifies 45 buffers.

MIN
sets the number of SAS format values that can exist in a PUT function expression to 0. A value of 0 indicates that there is no minimum number of SAS format values required. This value is the default.

MAX
sets the maximum number of SAS format values that can exist in a PUT function expression to 5,000.

Default: 0

Tip: Alternatively, you can set the SQLREDUCEPUTVALUES system option. The value that is specified in the SQLREDUCEPUTVALUES system option is in effect for all SQL procedure statements, unless the PROC SQL REDUCEPUTVALUES= option is set. The value of the REDUCEPUTVALUES= option takes precedence over the SQLREDUCEPUTVALUES system option. The RESET statement can also be used to set or reset the REDUCEPUTVALUES= option. However, changing the value of the REDUCEPUTVALUES= option does not change the value of the SQLREDUCEPUTVALUES system option. For more information, see the SQLREDUCEPUTVALUES system option in the *SAS Language Reference: Dictionary*.

REMERGE|NOREMERGE
Specifies whether PROC SQL can process queries that use remerging of data. The remerge feature of PROC SQL makes two passes through a table, using data in the second pass that was created in the first pass, in order to complete a query. When the NOREMERGE system option is set, PROC SQL cannot process remerging of data. If remerging is attempted when the NOREMERGE option is set, an error is written to the SAS log.

Default: REMERGE

Tip: Alternatively, you can set the SQLREMERGE system option. The value that is specified in the SQLREMERGE system option is in effect for all SQL procedure statements, unless the PROC SQL REMERGE option is set. The value of the REMERGE option takes precedence over the SQLREMERGE system option. The RESET statement can also be used to set or reset the REMERGE option. However, changing the value of the REMERGE option does not change the value of the SQLREMERGE system option. For more information, see the SQLREMERGE system option in the *SAS Language Reference: Dictionary*.

See also: “Remerging Data” on page 1288

SORTMSG|NOSORTMSG
Certain operations, such as ORDER BY, can 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): Reference 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 unless the system option is restricted (see Restriction). 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.

Restriction: Your site administrator can create a restricted options table. A restricted options table specifies SAS system option values that are established at startup and cannot be overridden. If the THREADS | NOTHREADS system option is listed in the restricted options table, any attempt to set it is ignored and a warning message is written to the SAS log.

Interaction: 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 might 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 can 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

Tip: If you are updating a data set using the SPD Engine, you can significantly improve processing performance by setting UNDO_POLICY=NONE. However, ensure that NONE is an appropriate setting for your application.

Tip: Alternatively, you can set the SQLUNDOPOLICY system option. The value that is specified in the SQLUNDOPOLICY= system option is in effect for all SQL procedure statements, unless the PROC SQL UNDO_POLICY= option is set. The value of the UNDO_POLICY= option takes precedence over the SQLUNDOPOLICY= system option. The RESET statement can also be used to set or reset the UNDO_POLICY= option. However, changing the value of the UNDO_POLICY= option does not change the value of the SQLUNDOPOLICY= system option. After the procedure completes, the undo policy reverts to the value of the SQLUNDOPOLICY= system option. For more information, see the SQLUNDOPOLICY system option in the *SAS Language Reference: Dictionary*.

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 1300

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 or columns that are specified in each column-definition.

column
names a column in *table-name*.

column-definition
See "column-definition" on page 1251.

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, the variables in a data file are part of both a primary key and a foreign key definition. The restrictions are as follows:

- 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, the 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 1227).

<**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 or columns.

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

You cannot use the RENAME= data set option with the ALTER TABLE statement to change a column's name. However, you can use the RENAME= data set option with the CREATE TABLE or SELECT statement. For more information on the RENAME= data set option, see 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 1218 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 SAS/ACCESS software. supports

Requirement: SAS/ACCESS software is required. For more information about this statement, see your SAS/ACCESS documentation.

See also: "Connecting to a DBMS Using the SQL Procedure Pass-Through Facility" in the *SAS 9.2 SQL Procedure User's Guide*

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, remains unique for every row in a table. A unique index can be defined for a column that includes NULL or missing values if each row has a unique index value.

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 "Accessing SAS System Information Using DICTIONARY Tables" in the *SAS 9.2 SQL Procedure User's Guide*.

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 1296
Example 2 on page 1299

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, the 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, the variables in a data file are part of both a primary key definition and a foreign key definition. The restrictions are as follows:

- 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 1251.

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, the variables in a data file are part of both a primary key definition and a foreign key definition. The restrictions are as follows:

- 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 key constraints, the 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 1243.

query-expression
creates *table-name* from the results of a query. See "query-expression" on page 1270.

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" in *SAS 9.2 SQL Procedure User's Guide* 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 tables 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 1223.

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.

The following interactions apply to integrity constraints when they are part of a column specification.

- You cannot specify compound primary keys.
- The check constraint that you specify in a column specification does not need to reference that same column in its WHERE clause.
- You can specify more than one integrity constraint.
- You can specify the MSGTYPE= and MESSAGE= options on a constraint.

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 1199

Featured in: Example 8 on page 1313

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 1270.

libname-clause
is one of the following:

LIBNAME *libref* <*engine*> *'SAS-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 1243.

proc-sql-view
specifies the name for the PROC SQL view that you are creating. See "What Are Views?" on page 1199 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 example, 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 the 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—even if an index is present. 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 results to appear in a particular order, then use the ORDER BY clause. △

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 1208. △

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 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 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" in the *SAS 9.2 SQL Procedure User's Guide*.

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, which 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-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 1305

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 1277.

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" in the *SAS 9.2 SQL Procedure User's Guide*.

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 1307

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 SAS/ACCESS software supports, 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 or tables. However, names of the foreign key data set variables that reference the primary key constraint will not be displayed as part of the primary key constraint's DESCRIBE TABLE output.

DISCONNECT Statement

Ends the connection with a DBMS that a SAS/ACCESS interface supports.

Requirement: SAS/ACCESS software is required. For more information on this statement, see your SAS/ACCESS documentation.

See also: "Connecting to a DBMS Using the SQL Procedure Pass-Through Facility" in the *SAS 9.2 SQL Procedure User's Guide*

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 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 tables and views 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 a SAS/ACCESS view describes.

EXECUTE Statement

Sends a DBMS-specific SQL statement to a DBMS that a SAS/ACCESS interface supports.

Requirement: SAS/ACCESS software is required. For more information on this statement, see your SAS/ACCESS documentation.

See also:
"Connecting to a DBMS Using the SQL Procedure Pass-Through Facility" in the *SAS 9.2 SQL Procedure User's Guide*
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, which can be executed by the DBMS-specific dynamic SQL. The SQL statement can contain a semicolon. The SQL statement can be case-sensitive, depending on your data source, and it is passed to the data source exactly as you type it.

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.

Example

The following example, after the connection, uses the EXECUTE statement to drop a table, create a table, and insert a row of data.

```
proc sql;
    execute(drop table ' My Invoice ') by db;
    execute(create table ' My Invoice '(
       ' Invoice Number ' LONG not null,
       ' Billed To '  VARCHAR(20),
       ' Amount '  CURRENCY,
       ' BILLED ON '  DATETIME)) by db;
    execute(insert into  ' My Invoice '
       values( 12345, 'John Doe', 123.45, #11/22/2003#)) by db;
quit;
```

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 1296

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 1270.

sas/access-view
specifies a SAS/ACCESS view into which you are inserting rows.

sql-expression
See "sql-expression" on page 1277.

Restriction: You cannot use a logical operator (AND, OR, or NOT) in an expression in a SET clause.

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" in the *SAS 9.2 SQL Procedure User's Guide*.

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 the "CONTENTS Statement" on page 314.
- the DICTIONARY.INDEXES table. See "Accessing SAS System Information Using DICTIONARY Tables" in the *SAS 9.2 SQL Procedure User's Guide SAS 9.2 SQL Procedure User's Guide* for more information.

For more information on creating and using indexes, see the "CREATE INDEX Statement" on page 1218.

RESET Statement

Resets PROC SQL options without restarting the procedure.

Featured in: Example 5 on page 1305

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 1204 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 1292
"query-expression" on page 1270

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 1251

Featured in: Example 1 on page 1296 and Example 2 on page 1299

SELECT <DISTINCT> *object-item* <, … *object-item*>

Arguments

alias
assigns a temporary, alternate name to the column.

DISTINCT
eliminates duplicate rows.

Tip: A row is considered a duplicate when all of its values are the same as the values of another row. The DISTINCT argument applies to all columns in the SELECT list. One row is displayed for each existing combination of values.

Note: DISTINCT works on the internal or stored value, not necessarily on the value as it is displayed. Numeric precision can cause multiple rows to be returned with values that appear to be the same. △

Tip: If available, PROC SQL uses index files when processing SELECT DISTINCT statements.

Featured in: Example 13 on page 1324

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 1249.

column-name <**AS** *alias*>
<column-modifier <… column-modifier>>
names a single column. See "column-name" on page 1254 and "column-modifier" on page 1252.

sql-expression <**AS** *alias*>
<column-modifier <… column-modifier>>
derives a column from an sql-expression. See "sql-expression" on page 1277 and "column-modifier" on page 1252.

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 view that is specified in *view-name*.

view-alias.*
specifies all columns in the SAS view that has the alias that is specified in *view-alias*.

Asterisk (*) Notation

The asterisk (*) represents all columns of the table or tables 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.

Note: A warning will occur if you create an output table using the SELECT * syntax when columns with the same name exist in the multiple tables that are listed on the FROM clause. You can avoid the warning by using one of the following actions:

- Individually list the desired columns in the SELECT statement at the same time as you omit the duplicate column names.
- Use the RENAME= and DROP= data set options. In this example, the ID column is renamed **tmpid**.

```
proc sql;
   create table all(drop=tmpid) as
      select * from
         one, two(rename=(id=tmpid))
            where one.id=two.tmpid;
quit;
```

If table aliases are used, place the RENAME= data set option after the table name and before the table alias. You can omit the DROP= data set option if you want to keep the renamed column in the final output table.

△

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 1252. The keyword AS is required with a column alias to distinguish the alias from other column names in the SELECT clause.

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 the PROC SQL Automatic Macro Variables" in the *SAS 9.2 SQL Procedure User's Guide*

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.

Tip: When you specify a range of macro variables, the SAS Macro Facility creates only the number of macro variables that are needed. For example, if you specify **`:var1-:var9999`** and only 55 variables are needed, only **`:var1-:var55`** is created. The SQLOBS automatic variable is useful if a subsequent part of your program needs to know how many variables were actually created. In this example, SQLOBS would have the value of 55.

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 the PROC SQL Automatic Macro Variables" in the *SAS 9.2 SQL Procedure User's Guide* for more information on SQLOBS.

  Note: The SQLOBS automatic macro variable is assigned a value *after* the SQL SELECT statement executes. △
- Values assigned by the INTO clause use the BEST8. format.

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. The SQLOBS macro variable is useful to reveal how many distinct variables there were in the data processed by the query.

```
proc sql noprint;
   select distinct style
      into :s1 separated by ','
      from proclib.houses;
%put &s1;
%put There were &sqlobs distinct values.;
```

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
There were 4 distinct values.
```

- 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 1296, Example 4 on page 1303, Example 9 on page 1316, and Example 10 on page 1319

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 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 1258.

(query-expression) <<**AS**> *alias*>
<(*column* <, … *column*>)>
specifies an in-line view. See "query-expression" on page 1270.

CONNECTION TO
specifies a DBMS table. See "CONNECTION TO" on page 1255.

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" in the *SAS 9.2 SQL Procedure User's Guide*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 256 tables in any one query. The 256-table limit includes underlying tables that can 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 list of names enclosed in parentheses following the alias. This syntax can be useful for renaming columns. See Example 10 on page 1319 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 1303 and Example 9 on page 1316

WHERE sql-expression

Argument

sql-expression
See “sql-expression” on page 1277.

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.

  In this example, MAX is a summary function; therefore, its context is that of a GROUP BY clause. It cannot be used to group, or summarize, data.

```
where max(measure1) > 50;
```

  However, this WHERE clause will work.

```
where max(measure1,measure2) > 50;
```

  In this case, MAX is a SAS function. It works with the WHERE clause because you are comparing the values of two columns within the same row. Consequently, it can be used to subset the data.

GROUP BY Clause

Specifies how to group the data for summarizing.

Featured in: Example 8 on page 1313 and Example 12 on page 1322

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 1254.

sql-expression
See “sql-expression” on page 1277.

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 1313 and Example 12 on page 1322

HAVING sql-expression

Argument

sql-expression
See “sql-expression” on page 1277.

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 or groups 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 1299) and groups the rows by Gender to determine the oldest employee of each gender. In SAS, dates are stored as integers. The lower the birthdate as an integer, the greater the age. The expression **`birth=min(birth)`** is evaluated for each row in the table. When the minimum birthdate 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 1288 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 1270

Featured in: Example 11 on page 1320

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 1254.

sql-expression
See “sql-expression” on page 1277.

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 1204.
- 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—even if an index is present. 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 1300

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 1277.

Restriction: You cannot use a logical operator (AND, OR, or NOT) in an expression in a SET clause.

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" in the *SAS 9.2 SQL Procedure User's Guide*.

Details

- Any column that is not modified retains its original values, except in certain queries using the CASE expression. See "CASE expression" on page 1249 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 1213.
- 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 1270.

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 the PROC SQL Automatic Macro Variables" in the *SAS 9.2 SQL Procedure User's Guide* 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 might 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 1277.

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 1277.

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 1300
Example 13 on page 1324

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 1277.

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 1309

COALESCE (column-name <, … column-name>)

Arguments

column-name
is described in "column-name" on page 1254.

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 1293 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 1265. △

column-definition

Defines PROC SQL's data types and dates.

See also: "column-modifier" on page 1252
Featured in: Example 1 on page 1296

column data-type <column-modifier <… column-modifier>>

Arguments

column
is a column name.

column-modifier
is described in "column-modifier" on page 1252.

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 1251 and SELECT Clause on page 1233

Featured in:

Example 1 on page 1296
Example 2 on page 1299

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): Reference Guide*.

Default: YES

Restriction: The TRANSCODE=NO argument is not supported by some SAS Workspace Server clients. In SAS 9.2, if the argument is not supported, column values with TRANSCODE=NO are replaced (masked) with asterisks (*). Before SAS 9.2, column values with TRANSCODE=NO were transcoded.

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 1252

SELECT Clause on page 1233

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" in the *SAS 9.2 SQL Procedure User's Guide*

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 syntax is not valid for PROC SQL. For example, your DBMS query can contain a semicolon.

The DBMS determines the number of tables that you can join with *dbms-query*. Each CONNECTION TO component counts as one table toward the 256-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 1309

sql-expression <NOT> **CONTAINS** sql-expression

Argument

sql-expression

is described in “sql-expression” on page 1277.

EXISTS condition

Tests if a subquery returns one or more rows.

See also: “Query Expressions (Subqueries)” on page 1280

<NOT> **EXISTS** (query-expression)

Argument

query-expression
is described in “query-expression” on page 1270.

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 1299) 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 1303), 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 1303

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 1270.

sql-expression
is described in "sql-expression" on page 1277.

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 1305

sql-expression **IS** <NOT> **NULL** | **MISSING**

Argument

sql-expression
is described in "sql-expression" on page 1277.

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 method for evaluating missing values differs from the method of the ANSI Standard for SQL. According to the Standard, these expressions are NULL. See "sql-expression" on page 1277 for more information on predicates and operators. See "PROC SQL and the ANSI Standard" on page 1293 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 256 tables.

See also: FROM Clause on page 1239 and "query-expression" on page 1270

Featured in:

Example 4 on page 1303
Example 7 on page 1309
Example 9 on page 1316
Example 13 on page 1324
Example 14 on page 1328

❶ *table-name* <<**AS**> *alias*>, *table-name* <<**AS**> *alias*>
<, … *table-name* <<**AS**> *alias*>>

❷ *table-name* <<**AS**> *alias*> <**INNER**> **JOIN** *table-name* <<**AS**> *alias*>
ON sql-expression

❸ *table-name* <<**AS**> *alias*> **LEFT JOIN** | **RIGHT JOIN** | **FULL JOIN**
table-name <<**AS**> *alias*> **ON** sql-expression

❹ *table-name* <<**AS**> *alias*> **CROSS JOIN** *table-name* <<**AS**> *alias*>

❺ *table-name* <<**AS**> *alias*> **UNION JOIN** *table-name* <<**AS**> *alias*>

❻ *table-name* <<**AS**> *alias*> **NATURAL**
<**INNER** | **FULL** <**OUTER**> | **LEFT** <**OUTER** > | **RIGHT** <**OUTER** >>
JOIN *table-name* <<**AS**> *alias*>

Arguments

alias
specifies an alias for *table-name*. The AS keyword is optional.

sql-expression
is described in "sql-expression" on page 1277.

table-name
can be one of the following:

- the name of a PROC SQL table.
- the name of a SAS 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 1239 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 1255 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 1260.
❸ Outer join. See "Outer Joins" on page 1262.
❹ Cross join. See "Cross Joins" on page 1263.
❺ Union join. See "Union Joins" on page 1264.
❻ Natural join. See "Natural Joins" on page 1265.

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 can 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 256 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 256-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 1259. 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 1309 for an example. △

Table Aliases

Table aliases are used in joins to distinguish the columns of one table from the columns in the other table or tables. 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 1239 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 1324 and Example 14 on page 1328 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 tables, as specified by the sql-expression. Inner joins can be performed on up to 256 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;
```

Output 55.1 Cartesian Product of LEFTTAB and RIGHTTAB Tables

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;
```

Output 55.2 Inner Join

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 1303, Example 13 on page 1324, and Example 14 on page 1328 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;
```

Output 55.3 Left Outer Join

```
                      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;
```

Output 55.4 Right Outer Join

```
                        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;
```

Output 55.5 Full Outer Join

```
                        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 1309 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;
```

Output 55.6 Cross Join

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;
```

Output 55.7 Union Join

```
                          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 1270 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;
```

Output 55.8 Natural Inner Join

```
                    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;
```

Output 55.9 Natural Left Outer Join

```
                  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 256 tables in PROC SQL. You can combine several joins of the same or different types as shown in the following code lines:

```
a natural join b natural join c

a natural join b cross join c
```

You can also use parentheses to group joins together and control what joins happen in what order as shown in the following examples:

```
(a, b) left join c on a.X=c.Y

a left join (b full join c on b.Z=c.Z) on a.Y=b.Y
```

Note: Commutative behavior varies depending on the type of join that is performed. △

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 condition relates the first table to the second table; and the other condition relates the second

table to the third table. It is possible to break this example into stages. You could perform a two-way join to create a temporary table and then you could join the temporary table with the third one. However, PROC SQL can do it all in one step as shown in the next example. The final table would be the same in both cases.

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 as c JOIN price as p
       on (c.export=p.export)
        JOIN amount as a
      on (c.country=a.country);
quit;
```

Output 55.10 Three-Way Join

Total Export Revenue

Country	Export	Price	Quantity	Total
Canada	wheat	2.98	16000	47680
France	corn	3.45	2400	8280
Italy	rice	3.56	500	1780
Egypt	oil	18	10000	180000

See Example 9 on page 1316 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 1299. △

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 1277.

character-expression
is an sql-expression that evaluates to a single character. The operands of *character-expression* must be character or string literals.

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 strings 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 consist 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 examples 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 might 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 1293

LOWER (sql-expression)

Argument

sql-expression
must resolve to a character string and is described in "sql-expression" on page 1277.

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 1292
"Query Expressions (Subqueries)" on page 1280
"In-Line Views" on page 1240

table-expression <*set-operator* table-expression> <…*set-operator* table-expression>

Arguments

table-expression
is described in "table-expression" on page 1292.

set-operator
is one of the following:

INTERSECT <CORRESPONDING> <ALL>

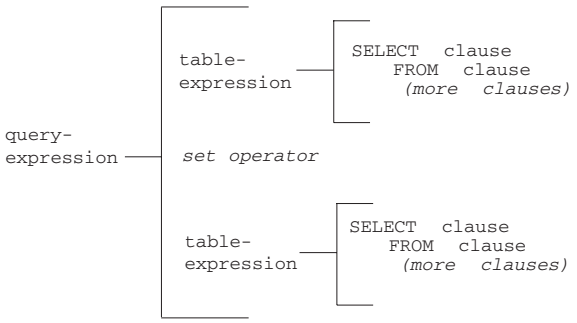
OUTER UNION <CORRESPONDING>

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 expressions: 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 or ones 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 1272.

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 55.11 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 55.11 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 1264 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;
```

Output 55.12 Outer Union Corresponding

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;
```

Output 55.13 Union of ME1 and ME2 Tables

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;
```

Output 55.14 Union All

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 1305 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.

IN_USA

Flight	Dest
145	ORD
156	WAS
188	LAX
193	FRA
207	LON

OUT_USA

Flight	Dest
193	FRA
207	LON
311	SJA

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;
```

Output 55.15 Flights from IN_USA Only

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;
```

Output 55.16 Flights from Both IN_USA and 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 1254.
- a CASE expression, which is described in "CASE expression" on page 1249.
- any supported SAS function. PROC SQL supports many of the functions available to the SAS DATA step. Some of the functions that aren't supported are the variable information functions, functions that work with arrays of data, and functions that operate on rows other than the current row. Other SQL databases support their own sets of functions. Functions are described in the *SAS Language Reference: Dictionary*.
- any functions, except those with array elements, that are created with PROC FCMP.
- the ANSI SQL functions COALESCE, BTRIM, LOWER, UPPER, and SUBSTRING.

- a summary-function, which is described in "summary-function" on page 1285.
- a query-expression, which is described in "query-expression" on page 1270.
- 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 1278.

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 many of the functions available to the SAS DATA step. Some of the functions that aren't supported are the variable information functions and functions that work with arrays of data. Other SQL databases support their own sets of functions. For example, the SCAN function is used in the following query:

```
select style, scan(street,1) format=$15.
   from houses;
```

PROC SQL also supports any user-written functions, except those functions with array elements, that are created using PROC FCMPChapter 23, "The FCMP Procedure," on page 417.

See the *SAS Language Reference: Dictionary* for complete documentation of SAS functions. Summary functions are also SAS functions. See "summary-function" on page 1285 for more information.

USER Literal

USER can be specified in a view definition, for example, to create a view that restricts access to the views 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 55.1 Operators and Order of Evaluation

Group	Operator	Description
0	( )	forces the expression enclosed to be evaluated first
1	case-expression	selects result values that satisfy specified conditions
2	**	raises to a power
	unary +, unary -	indicates a positive or negative number
3	*	multiplies
	/	divides
4	+	adds
	–	subtracts
5	||	concatenates
6	<NOT> BETWEEN condition	See “BETWEEN condition” on page 1247.
	<NOT> CONTAINS condition	see “CONTAINS condition” on page 1255.
	<NOT> EXISTS condition	See “EXISTS condition” on page 1256.
	<NOT> IN condition	See “IN condition” on page 1256.
	IS <NOT> condition	See “IS condition” on page 1257.
	<NOT> LIKE condition	See “LIKE condition” on page 1268.
7	=, eq	equals
	¬=, ^=, < >, ne	does not equal
	>, gt	is greater than
	<, lt	is less than
	>=, ge	is greater than or equal to
	<=, le	is less than or equal to
	=*	sounds like (use with character operands only). See Example 11 on page 1320.
	eqt	equal to truncated strings (use with character operands only). See “Truncated String Comparison Operators” on page 1280.
	gtt	greater than truncated strings
	ltt	less than truncated strings

Group	Operator	Description
	get	greater than or equal to truncated strings
	let	less than or equal to truncated strings
	net	not equal to truncated strings
8	¬, ^, NOT	indicates logical NOT
9	&, AND	indicates logical AND
10	|, OR	indicates logical OR

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 55.1 on page 1279.) 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 1299, and PROCLIB.STAFF is shown in Example 4 on page 1303.) 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');
```

Output 55.17 Query Output – One Value

```
                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');
```

Output 55.18 Query Output – Multiple Values

```
                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 of the employees 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');
```

Output 55.19 Query Output Using ALL Comparison

```
                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 1283 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 1613) creates PROCLIB.DELAY. PROCLIB.MARCH is shown in Example 13 on page 1324. 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 determine whether **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 55.20 Correlated Subquery Output

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 1277.
- *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 1241
HAVING Clause on page 1242
SELECT Clause on page 1233
"table-expression" on page 1292

Featured in:

Example 8 on page 1313
Example 12 on page 1322
Example 15 on page 1331

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
is the two-tailed *p*-value for Student's *t* statistic, T with $n - 1$ degrees of freedom.

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 1535.

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 1277.

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

* 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.

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*).

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 1313 and Example 15 on page 1331 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.

Note: When more than one argument is used within an SQL aggregate function, the function is no longer considered to be an SQL aggregate or summary function. If there is a like-named Base SAS function, then PROC SQL executes the Base SAS function, and the results that are returned are based on the values for the current row. If no like-named Base SAS function exists, then an error will occur. For example, if you use multiple arguments for the AVG function, an error will occur because there is no AVG function for Base SAS. △

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;
```

Output 55.21 Summary Functions

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.

Note: To specify that PROC SQL not process queries that use remerging of data, use either the PROC SQL NOREMERGE option or the NOSQLREMERGE system option. If remerging is attempted when the NOMERGE option or the NOSQLREMERGE system option is set, an error is written to the SAS log. For more information, see the REMERGE option on page 1211 and the SQLREMERGE system option in the *SAS Language Reference: Dictionary*. △

The following examples use the PROCLIB.PAYROLL table (shown in Example 2 on page 1299) 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;
```

Output 55.22 Salary Information That Required Remerging

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;
```

Output 55.23 Salary Information That Did Not Require Remerging

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 might 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;
```

Output 55.24 Jobcode Information That Did Not Require Remerging

```
                  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;
```

Output 55.25 Jobcode Information That Did Require Remerging

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 percentage 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:

```
data summary;
   input x;
   datalines;
32
86
49
49
;


proc sql;
   title 'Percentage of the Total';
   select X, (100*x/sum(X)) as Pct_Total
      from summary;
```

Output 55.26 Values of X as a Percentage of Total

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 1270

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 1233 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 1270

UPPER (sql-expression)

- sql-expression must be a character string and is described in "sql-expression" on page 1277.

Details

The UPPER function operates on character strings. UPPER converts the case of its argument to all uppercase.

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.

* 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.

- 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.

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 1204 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 1223 for more information.

CONTAINS Condition

PROC SQL enables you to test whether a string is part of a column's value when you specify the CONTAINS condition. See "CONTAINS condition" on page 1255 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 1243 and GROUP BY Clause on page 1241 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 tables that have a different number of columns, by creating virtual columns so that a query can evaluate correctly. See "query-expression" on page 1270 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 1285 for more information on the available summary functions and remerging data.

SAS DATA Step Functions

PROC SQL supports many of the functions available to the SAS DATA step. Some of the functions that aren't supported are the variable information functions and functions that work with arrays of data. Other SQL databases support their own sets of functions.

PROC FCMP Functions

PROC SQL supports any user-written functions, except those functions with array elements that are created using PROC FCMPChapter 23, "The FCMP Procedure," on page 417.

SQL Procedure Omissions

COMMIT Statement

The COMMIT statement is not supported.

ROLLBACK Statement

The ROLLBACK statement is not supported. The PROC SQL UNDO_POLICY= option or the SQLUNDOPOLICY system option addresses rollback. See the description of the UNDO_POLICY= option in "PROC SQL Statement" on page 1204 or the SQLUNDOPOLICY system option in the *SAS Language Reference: Dictionary* 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-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: Listing

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

Output: HTML

PROCLIB.PAYLIST Table

IdNum	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
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-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: Listing

```
                          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 to Create the Employee Table

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;
```

Output: Listing

```
                          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
```

Program to Update the Employee Table

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: Listing

```
                          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 Num	LName	FName	Job 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
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
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-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: Listing

Information for Certain Employees Only

Row	Lname	Fname	City	State	Id 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 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

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-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: Listing

```
                          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 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-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: Listing

```
                    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
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

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 Using OUTER JOIN Based on ID Number

Declare the PROCLIB library. The PROCLIB library is used in these examples to store created tables.

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

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=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: Listing

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                    .
```

Program Using COALESCE and LEFT JOIN

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: Listing

```
                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
```

Program to Subset the Query

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: Listing

```
                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
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-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: Listing

Current Summary Information for Each Job Category
Average Age Greater Than Or Equal to 30

Jobcode	Number	Average Age	Average 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

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

Flight	Date	Dest	Id 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

Supervisor Id	State	Job 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-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: Listing

```
                    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 1316 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-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: Listing

```
                    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
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 to Select Names That Sound Like 'Johnson'

Declare the PROCLIB library. The PROCLIB library is used in these examples to store created tables.

```
libname proclib 'SAS-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: Listing

```
              Employees Whose Last Name Sounds Like 'Johnson'                1

                   Id
                   Num                    Fname
                   ---------------------------------------
                   1411  JOHNSEN          JACK
                   1113  JOHNSON          LESLIE
                   1369  JONSON           ANTHONY
```

Program to Select Names that Sound Like 'Sanders'

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;
```

Output: Listing

```
              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

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	5	1
store2	5	3
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 whether 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: Listing

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 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-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: Listing

```
                        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: Listing

```
                        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: Listing

```
                          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, 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.

```
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
;
```

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: Listing

```
                    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: Listing

```
                  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
56
The STANDARD Procedure

Overview: STANDARD Procedure **1335**
What Does the STANDARD Procedure Do? **1335**
Standardizing Data **1335**
Syntax: STANDARD Procedure **1337**
PROC STANDARD Statement **1338**
BY Statement **1340**
FREQ Statement **1341**
VAR Statement **1342**
WEIGHT Statement **1342**
Results: STANDARD Procedure **1343**
Missing Values **1343**
Output Data Set **1343**
Statistical Computations: STANDARD Procedure **1343**
Examples: STANDARD Procedure **1344**
Example 1: Standardizing to a Given Mean and Standard Deviation **1344**
Example 2: Standardizing BY Groups and Replacing Missing Values **1346**

Overview: STANDARD Procedure

What Does the STANDARD Procedure Do?

The STANDARD procedure standardizes variables in a SAS data set to a given mean and standard deviation, and it creates a new SAS data set containing the standardized values.

Standardizing Data

The following output shows a simple standardization where the output data set contains standardized student exam scores. The statements that produce the output follow:

```
proc standard data=score mean=75 std=5
              out=stndtest;
run;

proc print data=stndtest;
run;
```

Output 56.1 Standardized Test Scores Using PROC STANDARD

```
                          The SAS System                          1

                Obs     Student       Test1

                  1     Capalleti    80.5388
                  2     Dubose       64.3918
                  3     Engles       80.9143
                  4     Grant        68.8980
                  5     Krupski      75.2816
                  6     Lundsford    79.7877
                  7     McBane       73.4041
                  8     Mullen       78.6612
                  9     Nguyen       74.9061
                 10     Patel        71.9020
                 11     Si           73.4041
                 12     Tanaka       77.9102
```

The following output shows a more complex example that uses BY-group processing. PROC STANDARD computes Z scores separately for two BY groups by standardizing life-expectancy data to a mean of 0 and a standard deviation of 1. The data are 1950 and 1993 life expectancies at birth for 16 countries. The birth rates for each country, classified as stable or rapid, form the two BY groups. The statements that produce the analysis also

- 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 1346.

Output 56.2 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: Basic Concepts in *SAS Output Delivery System: User's Guide* for details.

Tip: You can use the ATTRIB, FORMAT, LABEL, and WHERE statements. See Chapter 3, "Statements with the Same Function in Multiple Procedures," on page 35 for details. You can also use any global statements. See "Global Statements" on page 20 for a list.

PROC STANDARD <*option(s)*>;

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

FREQ *variable*;

VAR *variable(s)*;

WEIGHT *variable*;

Task	Statement
Standardize variables to a given mean and standard deviation	"PROC STANDARD Statement" on page 1338
Calculate separate standardized values for each BY group	"BY Statement" on page 1340
Identify a variable whose values represent the frequency of each observation	"FREQ Statement" on page 1341
Select the variables to standardize and determine the order in which they appear in the printed output	"VAR Statement" on page 1342
Identify a variable whose values weight each observation in the statistical calculations	"WEIGHT Statement" on page 1342

PROC STANDARD Statement

PROC STANDARD <*option(s)*>;

Task	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	

Task	Option
Print statistics for each variable to standardize	PRINT
Suppress all printed output	NOPRINT

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 20

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.

Alias: EXCLNPWGTS

MEAN=*mean-value*
standardizes variables to a mean of *mean-value*.

Default: mean of the input values

Featured in: Example 1 on page 1344

NOPRINT
suppresses the printing of the procedure output. NOPRINT is the default value.

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 1344

PRINT
prints the original frequency, mean, and standard deviation for each variable to standardize.

Featured in: Example 2 on page 1346

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 1346

STD=*std-value*
standardizes variables to a standard deviation of *std-value*.

Default: standard deviation of the input values

Featured in: Example 1 on page 1344

VARDEF=*divisor*
specifies the divisor to use in the calculation of variances and standard deviation. The following table shows the possible values for *divisor* and the associated divisors.

Table 56.1 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 41

Main discussion: "Keywords and Formulas" on page 1536

BY Statement

Calculates standardized values separately for each BY group.

Main discussion: "BY" on page 36

Featured in: Example 2 on page 1346

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 39

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 1344

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 41

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 1340 and the calculation of weighted statistics in "Keywords and Formulas" on page 1536 for more information.

Note: Before 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: Listing

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	Student Number	Section	Test1	Std Test1	Test2	Std Test2	Final
Capalleti	0545	1	94	80.54	91	80.86	87
Dubose	1252	2	51	64.39	65	71.63	91
Engles	1167	1	95	80.91	97	82.99	97
Grant	1230	2	63	68.90	75	75.18	80
Krupski	2527	2	80	75.28	69	73.05	71
Lundsford	4860	1	92	79.79	40	62.75	86
McBane	0674	1	75	73.40	78	76.24	72
Mullen	6445	2	89	78.66	82	77.66	93
Nguyen	0886	1	79	74.91	76	75.53	80
Patel	9164	2	71	71.90	77	75.89	83
Si	4915	1	75	73.40	71	73.76	73
Tanaka	8534	2	87	77.91	73	74.47	76

Example 2: Standardizing BY Groups and Replacing Missing Values

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 0 and standard deviation of 1
- replaces missing values with the given mean
- prints the mean and standard deviation for the variables to standardize
- prints the output data set.

Program

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
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;
```

---

* 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.

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: Listing

PROC STANDARD prints the variable name, mean, standard deviation, input frequency, and label of each variable to standardize for each BY group.

Life expectancies for Bangladesh, Mozambique, and Russia are no longer missing. The missing values are replaced with the given mean (0).

```
                      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

57

The SUMMARY Procedure

Overview: SUMMARY Procedure **1351**
Syntax: SUMMARY Procedure **1351**
PROC SUMMARY Statement **1352**
VAR Statement **1353**

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 33, "The MEANS Procedure," on page 609. 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: Basic Concepts in *SAS Output Delivery System: User's Guide* for details.

Tip: You can use the ATTRIB, FORMAT, LABEL, and WHERE statements. See Chapter 3, "Statements with the Same Function in Multiple Procedures," on page 35 for details. You can also use any global statements. See "Global Statements" on page 20 for a list.

Tip: Full syntax descriptions are in "Syntax: MEANS Procedure" on page 612.

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*;

Table 57.1

TASK	STATEMENT
Compute descriptive statistics for variables across all observations or within groups of observations	"PROC SUMMARY Statement" on page 1352
Calculate separate statistics for each BY group	"BY Statement" on page 621
Identify variables whose values define subgroups for the analysis	"CLASS Statement" on page 622
Identify a variable whose values represent the frequency of each observation	"FREQ Statement" on page 626
Include additional identification variables in the output data set	"ID Statement" on page 626
Create an output data set that contains specified statistics and identification variables	"OUTPUT Statement" on page 627
Identify specific combinations of class variables to use to subdivide the data	"TYPES Statement" on page 633
Identify the analysis variables and their order in the results	"VAR Statement" on page 1353
Specify the number of ways to make unique combinations of class variables	"WAYS Statement" on page 636
Identify a variable whose values weight each observation in the statistical calculations	"WEIGHT Statement" on page 636

Note: Full descriptions of the statements for PROC SUMMARY are in the documentation for Chapter 33, "The MEANS Procedure," on page 609. △

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.

Note: See "VAR Statement" on page 635 for a full description of the VAR statement. △

CHAPTER

58

The TABULATE Procedure

Overview: TABULATE Procedure **1356**
What Does the TABULATE Procedure Do? **1356**
Simple Tables **1356**
Complex Tables **1357**
PROC TABULATE and the Output Delivery System **1358**
Terminology: TABULATE Procedure **1359**
Syntax: TABULATE Procedure **1362**
PROC TABULATE Statement **1363**
BY Statement **1373**
CLASS Statement **1374**
CLASSLEV Statement **1378**
FREQ Statement **1378**
KEYLABEL Statement **1379**
KEYWORD Statement **1379**
TABLE Statement **1380**
VAR Statement **1389**
WEIGHT Statement **1391**
Concepts: TABULATE Procedure **1391**
Statistics That Are Available in PROC TABULATE **1392**
Formatting Class Variables **1393**
Formatting Values in Tables **1393**
How Using BY-Group Processing Differs from Using the Page Dimension **1394**
Calculating Percentages **1395**
Calculating the Percentage of the Value in a Single Table Cell **1395**
Using PCTN and PCTSUM **1395**
Specifying a Denominator for the PCTN Statistic **1396**
Specifying a Denominator for the PCTSUM Statistic **1397**
Using Style Elements in PROC TABULATE **1398**
What Are Style Elements? **1398**
Using the STYLE= Option **1399**
Applying Style Attributes to Table Cells **1400**
Using a Format to Assign a Style Attribute **1400**
In-Database Processing for PROC TABULATE **1400**
Results: TABULATE Procedure **1401**
Missing Values **1401**
How PROC TABULATE Treats Missing Values **1401**
No Missing Values **1403**
A Missing Class Variable **1404**
Including Observations with Missing Class Variables **1405**
Formatting Headings for Observations with Missing Class Variables **1406**
Providing Headings for All Categories **1407**

Providing Text for Cells That Contain Missing Values **1408**
Providing Headings for All Values of a Format **1409**
Understanding the Order of Headings with ORDER=DATA **1411**
Portability of ODS Output with PROC TABULATE **1412**
Examples: TABULATE Procedure **1413**
Example 1: Creating a Basic Two-Dimensional Table **1413**
Example 2: Specifying Class Variable Combinations to Appear in a Table **1415**
Example 3: Using Preloaded Formats with Class Variables **1417**
Example 4: Using Multilabel Formats **1423**
Example 5: Customizing Row and Column Headings **1425**
Example 6: Summarizing Information with the Universal Class Variable ALL **1427**
Example 7: Eliminating Row Headings **1429**
Example 8: Indenting Row Headings and Eliminating Horizontal Separators **1431**
Example 9: Creating Multipage Tables **1434**
Example 10: Reporting on Multiple-Response Survey Data **1436**
Example 11: Reporting on Multiple-Choice Survey Data **1441**
Example 12: Calculating Various Percentage Statistics **1448**
Example 13: Using Denominator Definitions to Display Basic Frequency Counts and Percentages **1451**
Example 14: Specifying Style Elements for ODS Output **1465**
Example 15: Style Precedence **1470**
References **1474**

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

The following output shows a simple table that was produced by PROC TABULATE. The data set "ENERGY" on page 1608 contains data on expenditures of energy by two types of customers, residential and business, in individual states in the Northeast (1) and West (4) regions of the United States. The table sums expenditures for states within a geographic division. (The RTS option provides enough space to display the column headings without hyphenating them.)

```
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 58.1 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

The following output is a more complicated table using the same data set that was used to create Output 58.1. The statements that create this report

- customize column and row headings
- apply a format to all table cells
- sum expenditures for residential and business customers
- compute subtotals for each division
- compute totals for all regions.

For an explanation of the program that produces this report, see Example 6 on page 1427.

Output 58.2 Complex Table Produced by PROC TABULATE

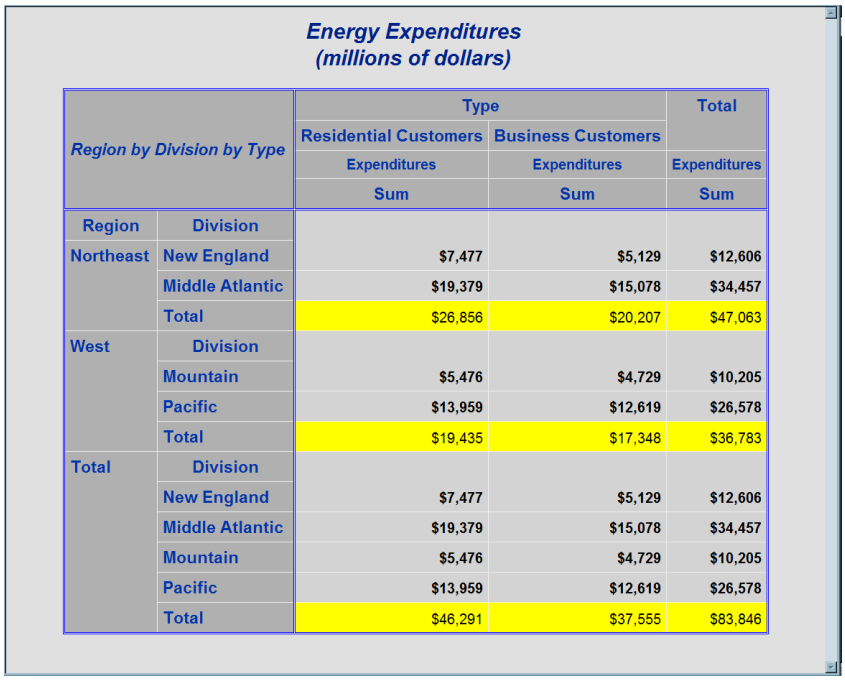
Energy Expenditures for Each Region 2
(millions of dollars)

		Customer Base		
		Residential Customers	Business Customers	All Customers
Region	Division			
Northeast	New England	7,477	5,129	12,606
	Middle Atlantic	19,379	15,078	34,457
	Subtotal	26,856	20,207	47,063
West	Division			
	Mountain	5,476	4,729	10,205
	Pacific	13,959	12,619	26,578
	Subtotal	19,435	17,348	36,783
Total for All Regions		$46,291	$37,555	$83,846

PROC TABULATE and the Output Delivery System

The following display shows a table that is created in Hypertext Markup Language (HTML). You can use the Output Delivery System with PROC TABULATE to create customized output in HTML, Rich Text Format (RTF), Portable Document Format (PDF), and other output formats. For an explanation of the program that produces this table, see Example 14 on page 1465.

Display 58.1 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 58.1 Parts of a PROC TABULATE Table

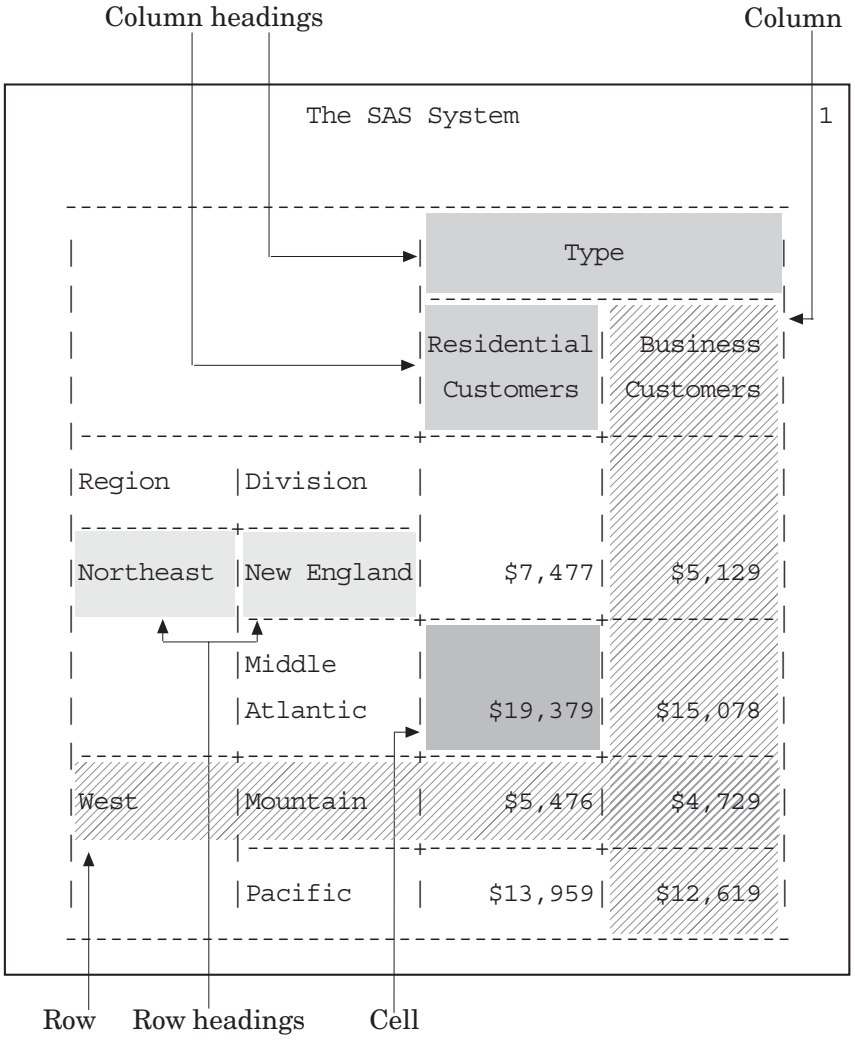
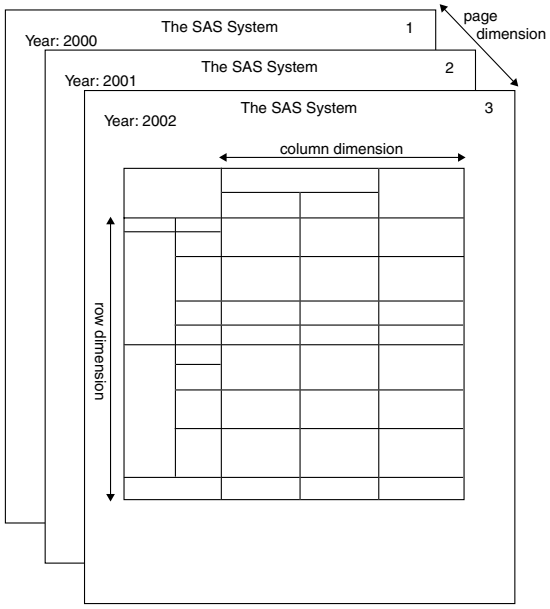


Figure 58.2 PROC TABULATE Table Dimensions



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 58.1 on page 1360 contains three class variables: Region, Division, and Type. These class variables form the eight categories listed in the following table. (For convenience, the categories are described in terms of their formatted values.)

Table 58.1 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 58.1 on page 1360, 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 58.2 on page 1361, 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 1369 in "What is the Output Delivery System?" in the *SAS Output Delivery System: User's Guide*.

subtable
the group of cells that is produced by crossing a single element from each dimension of the TABLE statement when one or more dimensions contain concatenated elements.

Figure 58.1 on page 1360 contains no subtables. For an illustration of a table that consists of multiple subtables, see Figure 58.18 on page 1456.

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 "How Does ODS Work?" in *SAS Output Delivery System: User's Guide* for details.

Tip: You can use the ATTRIB, FORMAT, LABEL, and WHERE statements. See Chapter 3, "Statements with the Same Function in Multiple Procedures," on page 35 for details. You can also use any global statements. See "Global Statements" on page 20 for a list.

PROC 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*;

Task	Statement
Display descriptive statistics in tabular format	"PROC TABULATE Statement" on page 1363
Create a separate table for each BY group	"BY Statement" on page 1373
Identify variables in the input data set as class variables	"CLASS Statement" on page 1374
Specify a style for class variable level value headings	"CLASSLEV Statement" on page 1378
Identify a variable in the input data set whose values represent the frequency of each observation	"FREQ Statement" on page 1378
Specify a label for a keyword	"KEYLABEL Statement" on page 1379
Specify a style for keyword headings	"KEYWORD Statement" on page 1379
Describe the table to create	"TABLE Statement" on page 1380
Identify variables in the input data set as analysis variables	"VAR Statement" on page 1389
Identify a variable in the input data set whose values weight each observation in the statistical calculations	"WEIGHT Statement" on page 1391

PROC TABULATE Statement

PROC TABULATE <*option(s)*>;

Task	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	

Task	Option
Specify the confidence level for the confidence limits	ALPHA=
Exclude observations with nonpositive weights	EXCLNPWGT
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 1415

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 20

EXCLNPWGT
excludes observations with nonpositive weight values (zero or negative) from the analysis. By default, PROC TABULATE treats observations with negative weights like observations with zero weights and counts them in the total number of observations.

Alias: EXCLNPWGTS

See also: WEIGHT= on page 1390 and "WEIGHT Statement" on page 1391

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 1415

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 1413 and Example 6 on page 1427

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 58.2 on page 1366 shows the formatting characters that PROC TABULATE uses. Figure 58.3 on page 1367 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 in which they are

listed. For example, the following option assigns the asterisk (*) to the third formatting character, the pound sign (#) to the seventh character, and does not alter the remaining characters:

```
formchar(3,7)='*#'
```

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

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 **x** after the closing quotation mark. For example, the following option assigns the hexadecimal character 2D to the third formatting character, assigns the hexadecimal character 7C to the seventh character, and does not alter the remaining characters:

```
formchar(3,7)='2D7C'x
```

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)
```

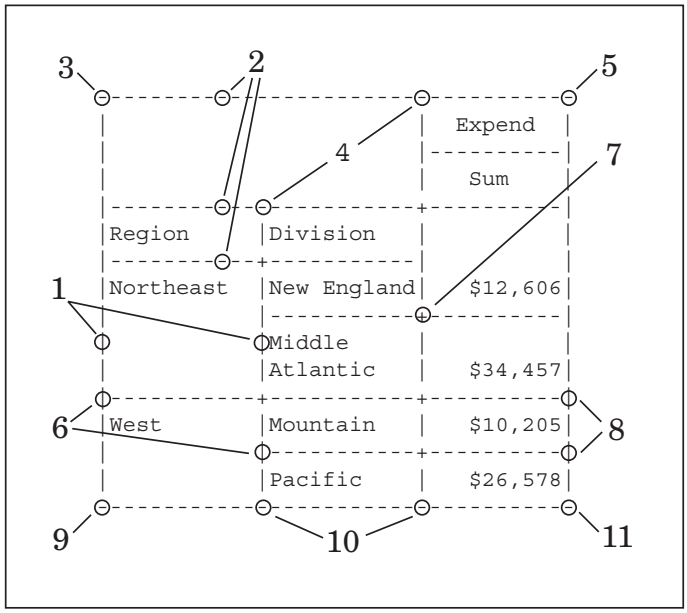
See also: For more information about formatting output, see Chapter 5, "Controlling the Table's Appearance," in the *SAS Guide to TABULATE Processing*.

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

Table 58.2 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 58.3 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 1405

See also: "Special Missing Values" in *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 headings.

Restriction: The NOSEPS option affects only the traditional SAS monospace output destination.

Tip: If you want to replace the separator lines with blanks rather than remove them, then use option FORMCHAR= on page 1365.

Featured in: Example 8 on page 1431

NOTHREADS
See THREADS | NOTHREADS on page 1372.

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 in which 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 1411

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. Otherwise, the position contains a 0. In simple PROC TABULATE steps that do not use the universal class variable ALL, all values of _TYPE_ contain only 1s because the only categories that are being considered involve all class variables. If you use the variable ALL, then your tables will contain data for categories that do not include all the class variables, and positions of _TYPE_ will, therefore, include both 1s and 0s.

PAGE
The logical page that contains the observation.

TABLE
The number of the table that contains the observation.

statistics
statistics that are calculated for each observation in the data set.

Featured in: Example 3 on page 1417

PCTLDEF=
See QNTLDEF= on page 1369.

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 643

QMETHOD=OS|P2|HIST
specifies the method PROC TABULATE uses to process the input data when it computes quantiles. If the number of observations is less than or equal to the QMARKERS= value and QNTLDEF=5, then both methods produce the same results.

OS
uses order statistics. PROC UNIVARIATE uses this technique.

Note: This technique can be very memory-intensive. △

P2|HIST
uses the P^2 method to approximate the quantile.

Default: OS

Restriction: When QMETHOD=P2, PROC TABULATE will not compute the following items:

- MODE
- weighted quantiles

Tip: When QMETHOD=P2, reliable estimates of some quantiles (P1, P5, P95, P99) might not be possible for some types of data.

Main Discussion: "Quantiles" on page 643

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 1541

STYLE=<*style-element-name*|<PARENT>>[*style-attribute-name=style-attribute-value*<... *style-attribute-name=style-attribute-value*>]

specifies the style element to use for the data cells of a table when it is used in the PROC TABULATE statement. For example, the following statement specifies that the background color for data cells be red:

```
proc tabulate data=one style=[backgroundcolor=red];
```

Note: This option can be used in other statements, or in dimension expressions, to specify style elements for other parts of a table. △

Note: You can use braces ({ and }) instead of square brackets ([and]). △

style-element-name
is the name of a style element that is part of a style definition that is registered with the Output Delivery System. SAS provides some style definitions. You can create your own style definitions with PROC TEMPLATE.

Default: If you do not specify a style element, then PROC TABULATE uses Data.

See also: "Concepts: Style Definitions and the TEMPLATE Procedure in *SAS Output Delivery System: User's Guide* for information about PROC TEMPLATE and the default style definitions. For information about the style elements, see "ODS Style Elements" in *SAS Output Delivery System: User's Guide*.

<PARENT>
specifies that the data cell use the style element of its parent heading. The parent style element of a data cell is one of the following:

- the style element of the leaf heading above the column that contains the data cell, if the table specifies no row dimension, or if the table specifies the style element in the column dimension expression.
- the style element of the leaf heading above the row that contains the cell, if the table specifies the style element in the row dimension expression.
- the Beforecaption style element, if the table specifies the style element in the page dimension expression.
- undefined, otherwise.

Note: In this usage, the angle brackets around the word PARENT are required. Curly braces or square brackets cannot be substituted in the syntax. △

Note: The parent of a heading (not applicable to STYLE= in the PROC TABULATE statement) is the heading under which the current heading is nested. △

style-attribute-name
specifies the attribute to change. The following table shows attributes that you can set or change with the STYLE= option in the PROC TABULATE statement (or in any other PROC TABULATE statement that uses the STYLE= option, except for the TABLE statement). Note that not all attributes are valid in all destinations.

Table 58.3 Style Attributes for PROC REPORT and PROC TABULATE

Attribute	PROC REPORT STATEMENT: REPORT Area	PROC REPORT Areas: CALLDEF, COLUMN, HEADER, LINES, SUMMARY	PROC TABULATE STATEMENT: TABLE	PROC TABULATE STATEMENTS: VAR, CLASS, BOX Opt, CLASSLEV, KEYWORD
ASIS=	X	X		X
BACKGOUNDCOLOR=	X	X	X	X

Attribute	PROC REPORT STATEMENT: REPORT Area	PROC REPORT Areas: CALLDEF, COLUMN, HEADER, LINES, SUMMARY	PROC TABULATE STATEMENT: TABLE	PROC TABULATE STATEMENTS: VAR, CLASS, BOX Opt, CLASSLEV, KEYWORD
BACKGOUNDIMAGE=	X	X	X	X
BORDERBOTTOMCOLOR=	X	X		X
BORDERBOTTOMSTYLE=	X	X	X	X
BORDERBOTTOMWIDTH=	X	X	X	X
BORDERCOLOR=	X	X		X
BORDERCOLORDARK=	X	X	X	X
BORDERCOLORLIGHT=	X	X	X	X
BORDERTOPCOLOR=	X	X		X
BORDERTOPSTYLE=	X	X	X	X
BORDERTOPWIDTH=	X	X	X	X
BORDERWIDTH=	X	X	X	X
CELLPADDING=	X		X	
CELLSPACING=	X		X	
CLASS=	X	X	X	X
COLOR=	X	X	X	
FLYOVER=	X	X		X
FONT=	X	X	X	X
FONTFAMILY=	X	X	X	X
FONTSIZE=	X	X	X	X
FONTSTYLE=	X	X	X	X
FONTWEIGHT=	X	X	X	X
FONTWIDTH=	X	X	X	X
FRAME=	X		X	
HEIGHT=	X	X		X
HREFTARGET=		X		X
HTMLSTYLE=	X	X	X	X
NOBREAKSPACE=	X	X		X
POSTHTML=	X	X	X	X
POSTIMAGE=	X	X	X	X
POSTTEXT=	X	X	X	X
PREHTML=	X	X	X	X
PREIMAGE=	X	X	X	X
PRETEXT=	X	X	X	X

Attribute	PROC REPORT STATEMENT: REPORT Area	PROC REPORT Areas: CALLDEF, COLUMN, HEADER, LINES, SUMMARY	PROC TABULATE STATEMENT: TABLE	PROC TABULATE STATEMENTS: VAR, CLASS, BOX Opt, CLASSLEV, KEYWORD
PROTECTSPECIALCHARS=		X		X
RULES=	X		X	
TAGATTR=	X	X		X
TEXTALIGN=	X	X	X	X
URL=		X		X
VERTICALALIGN=		X		X
WIDTH=	X	X	X	X

See also: Style Attributes and Their Values in *SAS Output Delivery System: User's Guide*

style-attribute-value
specifies a value for the attribute. Each attribute has a different set of valid values. See "Style Attributes and Their Values" in *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 1398

Featured in: Example 14 on page 1465

THREADS | NOTHREADS
enables or disables parallel processing of the input data set. This option overrides the SAS system option THREADS | NOTHREADS unless the system option is restricted. (See Restriction.) See "Support For Parallel Processing" in *SAS Language Reference: Concepts* for more information.

Default: value of SAS system option THREADS | NOTHREADS.

Restriction: Your site administrator can create a restricted options table. A restricted options table specifies SAS system option values that are established at startup and cannot be overridden. If the THREADS | NOTHREADS system option is listed in the restricted options table, any attempt to set these system options is ignored and a warning message is written to the SAS log.

Interaction: PROC TABULATE uses the value of the SAS system option THREADS except when a BY statement is specified or the value of the SAS system option CPUCOUNT is less than 2. In those cases, you can specify the THREADS option in the PROC TABULATE statement to force PROC TABULATE to use parallel processing.

Note: When multi-threaded processing, also known as parallel processing, is in effect, observations might be returned in an unpredictable order. However, the observations are sorted correctly if a BY statement is specified. △

TRAP

enables floating point exception (FPE) recovery during data processing beyond the recovery that is provided by normal SAS FPE handling. Note that without the TRAP option, normal SAS FPE handling is still in effect so that PROC TABULATE terminates in the case of math exceptions.

VARDEF=*divisor*
specifies the divisor to use in the calculation of the variance and standard deviation. The following table shows the possible values for *divisor* and the associated divisors.

Table 58.4 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 43

BY Statement

Creates a separate table on a separate page for each BY group.

Main discussion: "BY" on page 36

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. For example, the observations are grouped in chronological order.

The requirement for ordering or indexing observations according to the values of BY variables is suspended for BY-group processing when you use the NOTSORTED option. In fact, the procedure does not use an index if you specify NOTSORTED. The procedure defines a BY group as a set of contiguous observations that have the same values for all BY variables. If observations with the same values for the BY variables are not contiguous, then the procedure treats each contiguous set as a separate BY group.

CLASS Statement

Identifies class variables for the table. Class variables determine the categories that PROC TABULATE uses to calculate statistics.

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 ones 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.

Interaction: If a variable name and a statistic name are the same, enclose the statistic name in single or double quotation marks.

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 1417

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: "Special Missing Values" in *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 might 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 532 in the VALUE statement of the FORMAT procedure.

Featured in: Example 4 on page 1423

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 in which you define them.

FORMATTED
orders values by their ascending formatted values. This order depends on your operating environment.

Alias: FMT | EXTERNAL

FREQ
orders values by descending frequency count.

Interaction: Use the ASCENDING option to order values by ascending frequency count.

UNFORMATTED
orders values by their unformatted values, which yields the same order as PROC SORT. This order depends on your operating environment. This sort sequence is particularly useful for displaying dates chronologically.

Alias: UNFMT | INTERNAL

Default: UNFORMATTED

Interaction: If you use the PRELOADFMT option in the CLASS statement, then PROC TABULATE orders the levels by the order of the values in the user-defined format.

Tip: By default, all orders except FREQ are ascending. For descending orders, use the DESCENDING option.

Featured in: “Understanding the Order of Headings with ORDER=DATA” on page 1411

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 might not make sense. △

Featured in: Example 3 on page 1417

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 1369 in the PROC TABULATE statement.

Note: The use of STYLE= in the CLASS statement differs slightly from its use in the PROC TABULATE statement. In the CLASS statement, inheritance is different for rows and columns. For rows, the parent heading is located to the left of the current heading. For columns, the parent heading is located above the current heading. △

Note: If a page dimension expression contains multiple nested elements, then the Beforecaption style element is the style element of the first element in the nesting. △

Alias: S=

Restriction: This option affects only the HTML, RTF, and Printer destinations.

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 1465

How PROC TABULATE Handles Missing Values for Class Variables

By default, if an observation contains a missing value for any class variable, then PROC TABULATE excludes that observation from all tables that it creates. CLASS statements apply to all TABLE statements in the PROC TABULATE step. Therefore, if you define a variable as a class variable, then PROC TABULATE omits observations that have missing values for that variable from every table even if the variable does not appear in the TABLE statement for one or more tables.

If you specify the MISSING option in the PROC TABULATE statement, then the procedure considers missing values as valid levels for all class variables. If you specify the MISSING option in a CLASS statement, then PROC TABULATE considers missing values as valid levels for the class variables that are specified in that CLASS statement.

CLASSLEV Statement

Specifies a style element for class variable level value headings.

Restriction: This statement affects only the HTML, RTF, and Printer destinations.

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 about the arguments of this option and how it is used, see STYLE= on page 1369 in the PROC TABULATE statement.

Note: The use of STYLE= in the CLASSLEV statement differs slightly from its use in the PROC TABULATE statement. In the CLASSLEV statement, inheritance is different for rows and columns. For rows, the parent heading is located to the left of the current heading. For columns, the parent heading is located above the current heading. △

Alias: S=

Restriction: This option affects only the HTML, RTF, and Printer destinations.

Tip: To override a style element that is specified in the CLASSLEV statement, you can specify a style element in the related TABLE statement dimension expression.

Featured in: Example 14 on page 1465

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 39.

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 1392 or is the universal class variable ALL. (See "Elements That You Can Use in a Dimension Expression" on page 1386.)

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 1392 or is the universal class variable ALL. (See "Elements That You Can Use in a Dimension Expression" on page 1386.)

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 about the arguments of this option and how it is used, see STYLE= on page 1369 in the PROC TABULATE statement.

Note: The use of STYLE= in the KEYWORD statement differs slightly from its use in the PROC TABULATE statement. In the KEYWORD statement, inheritance is different for rows and columns. For rows, the parent heading is located to the left of the current heading. For columns, the parent heading is located above the current heading. △

Alias: S=

Restriction: This option affects only the HTML, RTF, and Printer destinations.

Tip: To override a style element that is specified in the KEYWORD statement, you can specify a style element in the related TABLE statement dimension expression.

Featured in: Example 14 on page 1465

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: To create several tables use multiple TABLE statements.

Tip: Use of variable name list shortcuts is now supported within the TABLE statement. For more information, refer to "Shortcuts for Specifying Lists of Variable Names" on page 25.

TABLE <<*page-expression*,> *row-expression*,>
column-expression </ *table-option(s)*>;

Required Arguments

column-expression
defines the columns in the table. For information about constructing dimension expressions, see "Constructing Dimension Expressions" on page 1386.

Restriction: A column dimension is the last dimension in a TABLE statement. A row dimension or a row dimension and a page dimension can precede a column dimension.

Options

Task	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=
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 1369 in the PROC TABULATE statement.

Featured in: Example 9 on page 1434 and Example 14 on page 1465

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 1434

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 1431 (with crossings) and Example 9 on page 1434 (without crossings)

MISSTEXT='*text*'

MISSTEXT={<*label*= '*text*'> <**STYLE**=<*style-element-name*>
[*style-attribute-name=style-attribute-value*<…
style-attribute-name=style-attribute-value>]>}
supplies up to 256 characters of text to print and specifies a style element for table cells that contain missing values. For details about the arguments of the STYLE= option and how it is used, see STYLE= on page 1369 in the PROC TABULATE statement.

Interaction: A style element that is specified in a dimension expression overrides a style element that is specified in the MISSTEXT= option for any given cells.

Featured in: "Providing Text for Cells That Contain Missing Values" on page 1408 and Example 14 on page 1465

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 about constructing dimension expressions, see "Constructing Dimension Expressions" on page 1386.

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 1434

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 1364

Featured in: "Providing Headings for All Categories" on page 1407

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 1429

row-expression

defines the rows in the table. For information about constructing dimension expressions, see "Constructing Dimension Expressions" on page 1386.

Restriction: A row dimension is the next to last dimension in a table statement. A column dimension must follow a row dimension. A page dimension can precede a row dimension.

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 1413

STYLE=<*style-element-name*> [*style-attribute-name=style-attribute-value*<... *style-attribute-name=style-attribute-value*>]
specifies a style element to use for parts of the table other than table cells. See STYLE= on page 1369 in the PROC TABULATE statement for information about the style element arguments.

Note: You can use braces ({ and }) instead of square brackets ([and]). △

The following table shows the attributes that you can set or change with the STYLE= option in the TABLE statement. Most of these attributes apply to parts of the table other than cells (for example, table borders and the lines between columns and rows). Attributes that you apply in the PROC TABULATE statement and in other locations in the PROC TABULATE step apply to cells within the table. Note that not all attributes are valid in all destinations. See "Style Attributes and Their Values" in *SAS Output Delivery System: User's Guide* for more information about these style attributes, their valid values, and their applicable destinations.

Table 58.5 Style Attributes for PROC REPORT and PROC TABULATE

Attribute	PROC REPORT STATEMENT: REPORT Area	PROC REPORT Areas: CALLDEF, COLUMN, HEADER, LINES, SUMMARY	PROC TABULATE STATEMENT: TABLE	PROC TABULATE STATEMENTS: VAR, CLASS, BOX Opt, CLASSLEV, KEYWORD
ASIS=	X	X		X
BACKGOUNDCOLOR=	X	X	X	X
BACKGOUNDIMAGE=	X	X	X	X
BORDERBOTTOMCOLOR=	X	X		X
BORDERBOTTOMSTYLE=	X	X	X	X

Attribute	PROC REPORT STATEMENT: REPORT Area	PROC REPORT Areas: CALLDEF, COLUMN, HEADER, LINES, SUMMARY	PROC TABULATE STATEMENT: TABLE	PROC TABULATE STATEMENTS: VAR, CLASS, BOX Opt, CLASSLEV, KEYWORD
BORDERBOTTOMWIDTH=	X	X	X	X
BORDERCOLOR=	X	X		X
BORDERCOLORDARK=	X	X	X	X
BORDERCOLORLIGHT=	X	X	X	X
BORDERTOPCOLOR=	X	X		X
BORDERTOPSTYLE=	X	X	X	X
BORDERTOPWIDTH=	X	X	X	X
BORDERWIDTH=	X	X	X	X
CELLPADDING=	X		X	
CELLSPACING=	X		X	
CLASS=	X	X	X	X
COLOR=	X	X	X	
FLYOVER=	X	X		X
FONT=	X	X	X	X
FONTFAMILY=	X	X	X	X
FONTSIZE=	X	X	X	X
FONTSTYLE=	X	X	X	X
FONTWEIGHT=	X	X	X	X
FONTWIDTH=	X	X	X	X
FRAME=	X		X	
HEIGHT=	X	X		X
HREFTARGET=		X		X
HTMLSTYLE=	X	X	X	X
NOBREAKSPACE=	X	X		X
POSTHTML=	X	X	X	X
POSTIMAGE=	X	X	X	X
POSTTEXT=	X	X	X	X
PREHTML=	X	X	X	X
PREIMAGE=	X	X	X	X
PRETEXT=	X	X	X	X
PROTECTSPECIALCHARS=		X		X
RULES=	X		X	
TAGATTR=	X	X		X
TEXTALIGN=	X	X	X	X

Attribute	PROC REPORT STATEMENT: REPORT Area	PROC REPORT Areas: CALLDEF, COLUMN, HEADER, LINES, SUMMARY	PROC TABULATE STATEMENT: TABLE	PROC TABULATE STATEMENTS: VAR, CLASS, BOX Opt, CLASSLEV, KEYWORD
URL=		X		X
VERTICALALIGN=		X		X
WIDTH=	X	X	X	X

Note: The list of attributes that you can set or change with the STYLE= option in the TABLE statement differs from the list of attributes of the PROC TABULATE statement. △

Alias: S=

Restriction: This option affects only the HTML, RTF, and Printer destinations.

Tip: To override a style element specification that is made as an option in the TABLE statement, specify STYLE= in a dimension expression of the TABLE statement.

Featured in: Example 14 on page 1465

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

Featured in: Example 15 on page 1470

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 consists of one or more elements and operators.

Elements That You Can Use in a Dimension Expression

analysis variables
(See “VAR Statement” on page 1389.)

class variables
(See “CLASS Statement” on page 1374.)

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 1427, Example 9 on page 1434, and Example 13 on page 1451

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 1392 for a list of available statistics. Use the asterisk (*) operator to associate a statistic keyword with a variable. The N statistic (number of nonmissing values) can be specified in a dimension expression without associating it with a variable.

Default: For analysis variables, the default statistic is SUM. Otherwise, the default statistic is N.

Examples:

```
n
Region*n
Sales*max
```

Restriction: Statistic keywords other than N must be associated with an analysis variable.

Interaction: Statistical keywords should be enclosed by single or double quotation marks to ensure that the keyword element is treated as a statistical keyword and not treated as a variable. By default, SAS treats these keywords as variables.

Featured in: Example 10 on page 1436 and Example 13 on page 1451

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 about specifying formats in tables, see "Formatting Values in Tables" on page 1393.

Featured in: Example 6 on page 1427

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 1425 and Example 7 on page 1429

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 1388.

Operators That You Can Use in a Dimension Expression

asterisk *
creates categories from the combination of values of the class variables and constructs the appropriate headings for the dimension. If one of the elements is an analysis variable, then the statistics for the analysis variable are calculated for the categories that are created by the class variables. This process is called *crossing*.

Examples:

```
Region*Division
Quarter*Sales*f=dollar8.2
```

Featured in: Example 1 on page 1413

(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 1427

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 1427

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 1395.

Featured in: Example 10 on page 1436 and Example 13 on page 1451

Specifying Style Elements in Dimension Expressions

You can specify a style element in a dimension expression to control the appearance in HTML, RTF, and Printer output of the following table elements:

analysis variable name headings
class variable name headings
class variable level value headings
data cells
keyword headings
page dimension text

Specifying a style element in a dimension expression is useful when you want to override a style element that you have specified in another statement, such as the PROC TABULATE, CLASS, CLASSLEV, KEYWORD, TABLE, or VAR statements.

The syntax for specifying a style element in a dimension expression is

[STYLE<(CLASSLEV)>=<*style-element-name* | PARENT>[*style-attribute-name=style-attribute-value*<… *style-attribute-name=style-attribute-value*>]]

Some examples of style elements in dimension expressions are

```
dept={label='Department'
      style=[color=red]}, N

dept*[style=MyDataStyle], N

dept*[format=12.2 style=MyDataStyle], N
```

Note: When used in a dimension expression, the STYLE= option must be enclosed within square brackets ([and]) or braces ({ and }). △

With the exception of (CLASSLEV), all arguments are described in STYLE= on page 1369 in the PROC TABULATE statement.

(CLASSLEV)
assigns a style element to a class variable level value heading. For example, the following TABLE statement specifies that the level value heading for the class variable, DEPT, has a foreground color of yellow:

```
table dept=[style(classlev)=
            [color=yellow]]*sales;
```

Note: This option is used only in dimension expressions. △

For an example that shows how to specify style elements within dimension expressions, see Example 14 on page 1465.

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.

Interaction: If a variable name and a statistic name are the same, enclose the statistic name in single or double quotation marks.

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 more information about the arguments of this option, see STYLE= on page 1369 in the PROC TABULATE statement.

Note: The use of STYLE= in the VAR statement differs slightly from its use in the PROC TABULATE statement. In the VAR statement, inheritance is different for rows and columns. For rows, the parent heading is located to the left of the current heading. For columns, the parent heading is located above the current heading. △

Alias: S=

Restriction: This option affects only the HTML, RTF, and Printer destinations.

Tip: To override a style element that is specified in the VAR statement, you can specify a style element in the related TABLE statement dimension expression.

Featured in: Example 14 on page 1465

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 1373.)

Tip: Use the WEIGHT option in multiple VAR statements to specify different weights for the analysis variables.

Note: Before 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 about calculating weighted statistics and for an example that uses the WEIGHT statement, see "Calculating Weighted Statistics" on page 42

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.

Restriction: PROC TABULATE will not compute MODE when a weight variable is active. Instead, try using PROC UNIVARIATE when MODE needs to be computed and a weight variable is active.

Interaction: If you use the WEIGHT= option in a VAR statement to specify a weight variable, then PROC TABULATE uses this variable instead to weight those VAR statement variables.

Tip: When you use the WEIGHT statement, consider which value of the VARDEF= option is appropriate. See the discussion of VARDEF= on page 1373 and the calculation of weighted statistics in the "Keywords and Formulas" on page 1536 section of this document.

Note: Before 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.

Note: If a variable name (class or analysis) and a statistic name are the same, then enclose the statistic name in single quotation marks (for example, **'MAX'**).
△

Descriptive statistic keywords

COLPCTN	PCTSUM
COLPCTSUM	RANGE
CSS	REPPCTN
CV	REPPCTSUM
KURTOSIS | KURT	ROWPCTN
LCLM	ROWPCTSUM
MAX	SKEWNESS | SKEW
MEAN	STDDEV | STD
MIN	STDERR
MODE	SUM
N	SUMWGT
NMISS	UCLM
PAGEPCTN	USS
PAGEPCTSUM	VAR
PCTN	

Quantile statistic keywords

MEDIAN | P50	Q3 | P75
P1	P90
P5	P95
P10	P99
Q1|P25	QRANGE

Hypothesis testing keywords

PROBT | PRT	T

These statistics, the formulas that are used to calculate them, and their data requirements are discussed in the Keywords and Formulas"Keywords and Formulas" on page 1536 section of this document.

To compute standard error of the mean (STDERR) or Student's *t*-test, you must use the default value of the VARDEF= option, which is DF. The VARDEF= option is specified in the PROC TABULATE statement.

To compute weighted quantiles, you must use QMETHOD=OS in the PROC TABULATE statement.

Use both LCLM and UCLM to compute a two-sided confidence limit for the mean. Use only LCLM or UCLM to compute a one-sided confidence limit. Use the ALPHA= option in the PROC TABULATE statement to specify a confidence level.

Formatting Class Variables

Use the FORMAT statement to assign a format to a class variable for the duration of a PROC TABULATE step. When you assign a format to a class variable, PROC TABULATE uses the formatted values to create categories, and it uses the formatted values in headings. If you do not specify a format for a class variable, and the variable does not have any other format assigned to it, then the default format, BEST12., is used, unless the GROUPINTERNAL option is specified.

User-defined formats are particularly useful for grouping values into fewer categories. For example, if you have a class variable, Age, with values ranging from 1 to 99, then you could create a user-defined format that groups the ages so that your tables contain a manageable number of categories. The following PROC FORMAT step creates a format that condenses all possible values of age into six groups of values.

```
proc format;
   value agefmt  0-29='Under 30'
                 30-39='30-39'
                 40-49='40-49'
                 50-59='50-59'
                 60-69='60-69'
                 other='70 or over';
run;
```

For information about creating user-defined formats, see Chapter 25, "The FORMAT Procedure," on page 511.

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" on page 1380 . For example, if you specify FORMAT_PRECEDENCE=ROW and specify a format modifier in the row dimension, then that format overrides all other specified formats for the table cells.

How Using BY-Group Processing Differs from Using the Page Dimension

Using the page-dimension expression in a TABLE statement can have an effect similar to using a BY statement.

The following table contrasts the two methods.

Table 58.6 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 1427.)
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 1395.)
Titles	You can use the #BYVAL, #BYVAR, and #BYLINE specifications in TITLE statements to customize the titles for each BY group. (See "Creating Titles That Contain BY-Group Information" on page 21.)	The BOX= option in the TABLE statement customizes the page headings, but you must use the same title on each page.
Ordering class variables	ORDER=DATA and ORDER=FREQ order each BY group independently.	The order of class variables is the same on every page.

Issue	PROC TABULATE with a BY statement	PROC TABULATE with a page dimension in the TABLE statement
Obtaining uniform headings	You might need to insert dummy observations into BY groups that do not have all classes represented.	The PRINTMISS option ensures that each page of the table has uniform headings.
Multiple ranges with the same format	PROC TABULATE produces a table for each range.	PROC TABULATE combines observations from the two ranges.

1 You can use the BY statement without sorting the data set if the data set has an index for the BY variable.

Calculating Percentages

Calculating the Percentage of the Value in a Single Table Cell

The following statistics print the percentage of the value in a single table cell in relation to the total of the values in a group of cells. No denominator definitions are required. However, an analysis variable can be used as a denominator definition for percentage sum statistics.

REPPCTN and REPPCTSUM statistics—print the percentage of the value in a single table cell in relation to the total of the values in the report.

COLPCTN and COLPCTSUM statistics—print the percentage of the value in a single table cell in relation to the total of the values in the column.

ROWPCTN and ROWPCTSUM statistics—print the percentage of the value in a single table cell in relation to the total of the values in the row.

PAGEPCTN and PAGEPCTSUM statistics—print the percentage of the value in a single table cell in relation to the total of the values in the page.

These statistics calculate the most commonly used percentages. See Example 12 on page 1448 for an example.

Using PCTN and PCTSUM

PCTN and PCTSUM statistics can be used to calculate these same percentages. They enable you to manually define denominators. PCTN and PCTSUM statistics print the percentage of the value in a single table cell in relation to the value (used in the denominator of the calculation of the percentage) in another table cell or to the total of the values in a group of cells. By default, PROC TABULATE summarizes the values in all N cells (for PCTN) or all SUM cells (for PCTSUM) and uses the summarized value for the denominator. You can control the value that PROC TABULATE uses for the denominator with a denominator definition.

You place a denominator definition in angle brackets (< and >) next to the PCTN or PCTSUM statistic. The denominator definition specifies which categories to sum for the denominator.

This section illustrates how to specify denominator definitions in a simple table. Example 13 on page 1451 illustrates how to specify denominator definitions in a table that consists 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 1608.

```
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 the following figure.) Each occurrence of PCTN uses a different denominator definition.

Figure 58.4 Three Different Uses of the PCTN Statistic with Frequency Counts Highlighted

Number of Users in Each Division

		Type	
		1	2
Division			
1	Number of customers	6.00	6.00
	% of row 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 the following figure.) Each occurrence of PCTSUM uses a different denominator definition.

Figure 58.5 Three Different Uses of the PCTSUM Statistic with Sums Highlighted

Expenditures in Each Division

		Type	
		1	2
		Expenditu-res	Expenditu-res
Division			
1	Expenditures	$7,477.00	$5,129.00
	% of row 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 on. See "Understanding Style Definitions, Style Elements, and Style Attributes" in *SAS Output Delivery System: User's Guide* for more information. See "ODS Style Elements" in *SAS Output Delivery System: User's Guide* for a comprehensive list of style elements.

The following table lists the default styles for various regions of a table.

Table 58.7 Default Styles for Table Regions

Region	Style
column headings	Heading
box	Heading
page dimension text	Beforecaption
row headings	Rowheading
data cells	Data
table	Table

Using the STYLE= Option

You specify style elements for PROC TABULATE with the STYLE= option. The following table shows where you can use this option. Specifications in the TABLE statement override the same specification in the PROC TABULATE statement. However, any style attributes that you specify in the PROC TABULATE statement and that you do not override in the TABLE statement are inherited. For example, if you specify a blue background and a white foreground for all data cells in the PROC TABULATE statement, and you specify a gray background for the data cells of a particular crossing in the TABLE statement, then the background for those data cells is gray, and the foreground is white (as specified in the PROC TABULATE statement).

Detailed information about STYLE= is provided in the documentation for individual statements.

Table 58.8 Using the STYLE= Option in PROC TABULATE

To set the style element for	Use STYLE in this statement
data cells	"PROC TABULATE Statement" on page 1363 or dimension expression(s)
page dimension text and class variable name headings	"CLASS Statement" on page 1374
class level value headings	"CLASSLEV Statement" on page 1378
keyword headings	"KEYWORD Statement" on page 1379
table borders, rules, and other parts that are not specified elsewhere	"TABLE Statement" on page 1380
box text	"TABLE Statement" on page 1380 BOX= option
missing values	"TABLE Statement" on page 1380 MISSTEXT= option
analysis variable name headings	"VAR Statement" on page 1389

Applying Style Attributes to Table Cells

PROC TABULATE determines the style attributes to use for a particular cell from the following default order of precedence for styles:

1. If no other style attributes are specified, then PROC TABULATE uses the default style attributes from the default style (Data).
2. The STYLE= option in the PROC TABULATE statement changes the default style attributes. If no other STYLE= option specifications affect a cell, then PROC TABULATE uses these style attributes for that cell.
3. A STYLE= option that is specified in the page dimension applies to all the table cells on the logical page unless you specify another STYLE= option for a cell in the row or column dimension.
4. A STYLE= option that is specified in the row dimension applies to all the table cells in the row unless you specify another STYLE= option for a cell in the column dimension.
5. A STYLE= option that is specified in the column dimension applies to all the table cells in the column.

You can change this order of precedence by using the STYLE_PRECEDENCE= option in the "TABLE Statement" on page 1380. For example, if you specify STYLE_PRECEDENCE=ROW and specify a STYLE= option in the row dimension, then those style attribute values override all others that are specified for the table cells.

Using a Format to Assign a Style Attribute

You can use a format to assign a style attribute value to any cell whose content is determined by values of a class or analysis variable. For example, the following code assigns a red background to cells whose values are less than 10,000, a yellow background to cells whose values are at least 10,000 but less than 20,000, and a green background to cells whose values are at least 20,000:

```
proc format;
   value expfmt low-<10000='red'
                     10000-<20000='yellow'
                     20000-high='green';
run;

ods html body='external-HTML-file';
proc tabulate data=energy style=[backgroundcolor=expfmt.];
   class region division type;
   var expenditures;
   table (region all)*(division all),
           type*expenditures;
run;
ods html close;
```

In-Database Processing for PROC TABULATE

When the DATA= input data set is stored as a table or view in a database management system (DBMS), the PROC TABULATE procedure can use in-database processing to perform most of its work within the database. In-database processing can provide the advantages of faster processing and reduced data transfer between the database and SAS software.

PROC TABULATE performs in-database processing by using SQL implicit pass-through. The procedure generates SQL queries that are based on the classifications and the statistics that you specify in the TABLE statement. The database executes these SQL queries to construct initial summary tables, which are then transmitted to PROC TABULATE.

If class variables are specified, the procedure creates an SQL GROUP BY clause that represents the n-way type. Only the n-way class tree is generated on the DBMS. The result set that is created when the aggregation query executes in the database is read by SAS into the internal PROC TABULATE data structure.

When SAS format definitions have been deployed in the database, formatting of class variables occurs in the database. If the SAS format definitions have not been deployed in the database, the in-database aggregation occurs on the raw values, and the relevant formats are applied by SAS as the results' set is merged into the PROC TABULATE internal structures. Multi-label formatting is always done by SAS using the initially aggregated result set that is returned by the database.

The following statistics are supported for in-database processing: N, NMISS, MIN, MAX, RANGE, SUM, SUMWGT, CSS, USS, VAR, STD, STDERR, UCLM, LCLM, and CV.

The SQLGENERATION system option or LIBNAME statement option controls whether and how in-database procedures are run inside the database. By default, the in-database procedures are run inside the database when possible. There are many data set options that will prevent in-database processing. For a complete listing, refer to "Overview of In-Database Procedures" in *SAS/ACCESS for Relational Databases: Reference*.

Results: TABULATE Procedure

Missing Values

How PROC TABULATE Treats Missing Values

How a missing value for a variable in the input data set affects your output depends on how you use the variable in the PROC TABULATE step. The following table summarizes how the procedure treats missing values.

Table 58.9 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

If …	PROC TABULATE, by default, …	To override the default …
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 the following figure.)

```
proc format;
   value cntryfmt 1='United States'
                  2='Japan';
   value compfmt  1='Supercomputer'
                  2='Mainframe'
                  3='Midrange'
                  4='Workstation'
                  5='Personal Computer'
                  6='Laptop';
run;


data comprev;
   input Country Computer Rev90 Rev91 Rev92;
   datalines;
1 1 788.8 877.6 944.9
1 2 12538.1 9855.6 8527.9
1 3 9815.8 6340.3 8680.3
1 4 3147.2 3474.1 3722.4
1 5 18660.9 18428.0 23531.1
2 1 469.9 495.6 448.4
2 2 5697.6 6242.4 5382.3
2 3 5392.1 5668.3 4845.9
2 4 1511.6 1875.5 1924.5
2 5 4746.0 4600.8 4363.7
```

```
;


proc print data=comprev noobs;
   format country cntryfmt. computer compfmt.;
   title 'The Data Set COMPREV';
run;
```

Figure 58.6 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 the following figure:

```
proc tabulate data=comprev;
   class country computer;
   var rev90 rev91 rev92;
   table computer*country,rev90 rev91 rev92 /
         rts=32;
   format country cntryfmt. computer compfmt.;
   title 'Revenues from Computer Sales';
   title2 'for 1990 to 1992';
run;
```

Figure 58.7 Computer Sales Data: No Missing Values

Because the data set contains no missing values, the table includes all observations. All headings and cells contain nonmissing values.

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
	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 58.8 on page 1405 is the same as the program that produces Figure 58.7 on page 1404. 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 58.8 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 less row than Figure 58.7 on page 1404.

```
                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

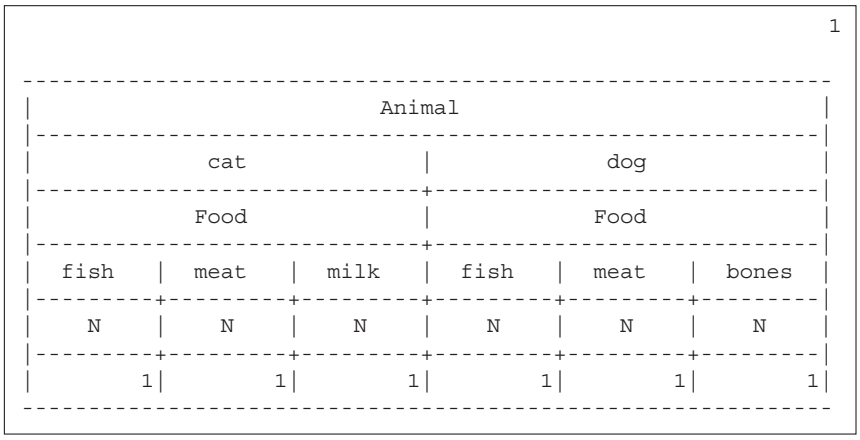
This program adds the MISSING option to the previous program. MISSING is available either in the PROC TABULATE statement or in the CLASS statement. If you want MISSING to apply only to selected class variables, but not to others, then specify MISSING in a separate CLASS statement with the selected variables. The MISSING option includes observations with missing values of a class variable in the report. (See the following figure.)

```
proc tabulate data=compmiss missing;
   class country computer;
   var rev90 rev91 rev92;
   table computer*country,rev90 rev91 rev92 /
         rts=32;
   format country cntryfmt. computer compfmt.;
   title 'Revenues from Computer Sales';
```

```
   title2 'for 1990 to 1992';
run;
```

Figure 58.9 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 58.9 on page 1406, PROC TABULATE displays missing values of a class variable as one of the standard SAS characters for missing values (a period, a blank, an underscore, or one of the letters A through Z). If you want to display something else instead, then you must assign a format to the class variable that has missing values, as shown in the following program. (See the following figure.)

```
proc format;
   value misscomp 1='Supercomputer'
                  2='Mainframe'
                  3='Midrange'
                  4='Workstation'
                  5='Personal Computer'
                  6='Laptop'
                  .='No type given';
run;


proc tabulate data=compmiss missing;
   class country computer;
   var rev90 rev91 rev92;
   table computer*country,rev90 rev91 rev92 /
         rts=32;
   format country cntryfmt. computer misscomp.;
   title 'Revenues for Computer Sales';
   title2 'for 1990 to 1992';
run;
```

Figure 58.10 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 58.10 on page 1407 does not include a row for **`No type given`** and for **`United States`** or for **`Midrange`** and for **`Japan`** because there are no data in these categories. If you want the table to represent all possible categories, then use the PRINTMISS option in the TABLE statement, as shown in the following program. (See the following figure.)

```
proc tabulate data=compmiss missing;
   class country computer;
   var rev90 rev91 rev92;
   table computer*country,rev90 rev91 rev92 /
         rts=32 printmiss;
   format country cntryfmt. computer misscomp.;
   title 'Revenues for Computer Sales';
   title2 'for 1990 to 1992';
run;
```

Figure 58.11 Computer Sales Data: Missing Statistics Values

This table contains a row for the category **No type given**, **United States** and the category **Midrange, Japan**. Because there are no data in these categories, the values for the statistics are all missing.

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 the following figure.)

```
proc tabulate data=compmiss missing;
   class country computer;
```

```
  var rev90 rev91 rev92;
  table computer*country,rev90 rev91 rev92 /
        rts=32 printmiss misstext='NO DATA!';
  format country cntryfmt. computer misscomp.;
  title 'Revenues for Computer Sales';
  title2 'for 1990 to 1992';
run;
```

Figure 58.12 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 1417 for another example that uses PRELOADFMT.
- Use the CLASSDATA= option in the PROC TABULATE statement. See Example 2 on page 1415 for an example that uses the CLASSDATA= option.
- Add dummy values to the input data set so that each value that the format handles appears at least once in the data set.

The following program adds the PRELOADFMT option to a CLASS statement that contains the relevant variable.

The results are shown in the following figure.

```
proc tabulate data=compmiss missing;
   class country;
   class computer / preloadfmt;
   var rev90 rev91 rev92;
   table computer*country,rev90 rev91 rev92 /
         rts=32 printmiss misstext='NO DATA!';
   format country cntryfmt. computer compfmt.;
   title 'Revenues for Computer Sales';
   title2 'for 1990 to 1992';
run;
```

Figure 58.13 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 Computer	United States	18660.90	18428.00	23531.10
	Japan	4746.00	4600.80	4363.70
Laptop	United States	NO DATA!	NO DATA!	NO DATA!
	Japan	NO DATA!	NO DATA!	NO DATA!

Understanding the Order of Headings with ORDER=DATA

The ORDER= option applies to all class variables. Occasionally, you want to order the headings for different variables differently. One method for reordering the headings is to group the data as you want them to appear and to specify ORDER=DATA.

For this technique to work, the first value of the first class variable must occur in the data with all possible values of all the other class variables. If this criterion is not met, then the order of the headings might surprise you.

The following program creates a simple data set in which the observations are ordered first by the values of Animal, then by the values of Food. The ORDER= option in the PROC TABULATE statement orders the heading for the class variables by the order of their appearance in the data set. (See the following figure.) Although **bones** is

the first value for Food in the group of observations where Animal=**dog**, all other values for Food appear before **bones** in the data set because **bones** never appears when Animal=**cat**. Therefore, the heading for **bones** in the table in the following figure is not in alphabetical order.

In other words, PROC TABULATE maintains for subsequent categories the order that was established by earlier categories. If you want to re-establish the order of Food for each value of Animal, then use BY-group processing. PROC TABULATE creates a separate table for each BY group, so that the ordering can differ from one BY group to the next.

```
data foodpref;
   input Animal $ Food $;
   datalines;
cat fish
cat meat
cat milk
dog bones
dog fish
dog meat
;

proc tabulate data=foodpref format=9.
              order=data;
   class animal food;
   table animal*food;
run;
```

Figure 58.14 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

The following example program

- 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 1608 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 Customers	Business Customers
		Expenditures	Expenditures
		Sum	Sum
Region	Division		
Northeast	New England	$7,477	$5,129
	Middle Atlantic	$19,379	$15,078
West	Mountain	$5,476	$4,729
	Pacific	$13,959	$12,619

Example 2: Specifying Class Variable Combinations to Appear in a Table

Procedure features:

PROC TABULATE Statement options:

CLASSDATA=
EXCLUSIVE

Data set: "ENERGY" on page 1608

Formats: REGFMT., DIVFMT., and USETYPE. on page 1414

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 1413.

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" on page 1608

Formats: REGFMT., DIVFMT., and USETYPE. on page 1414

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 1413.

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
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)

Obs	Region	Division	Type	_TYPE_	_PAGE_	_TABLE_	Expenditures_Sum
1	Northeast	New England	Residential Customers	111	1	1	7477
2	Northeast	New England	Business Customers	111	1	1	5129
3	Northeast	Middle Atlantic	Residential Customers	111	1	1	19379
4	Northeast	Middle Atlantic	Business Customers	111	1	1	15078
5	West	Mountain	Residential Customers	111	1	1	5476
6	West	Mountain	Business Customers	111	1	1	4729
7	West	Pacific	Residential Customers	111	1	1	13959
8	West	Pacific	Business Customers	111	1	1	12619

Example 4: Using Multilabel Formats

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 532. 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 58.3

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” on page 1608

Formats: REGFMT., DIVFMT., and USETYPE. on page 1414

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" on page 1608

Formats: REGFMT., DIVFMT., and USETYPE. on page 1414

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.

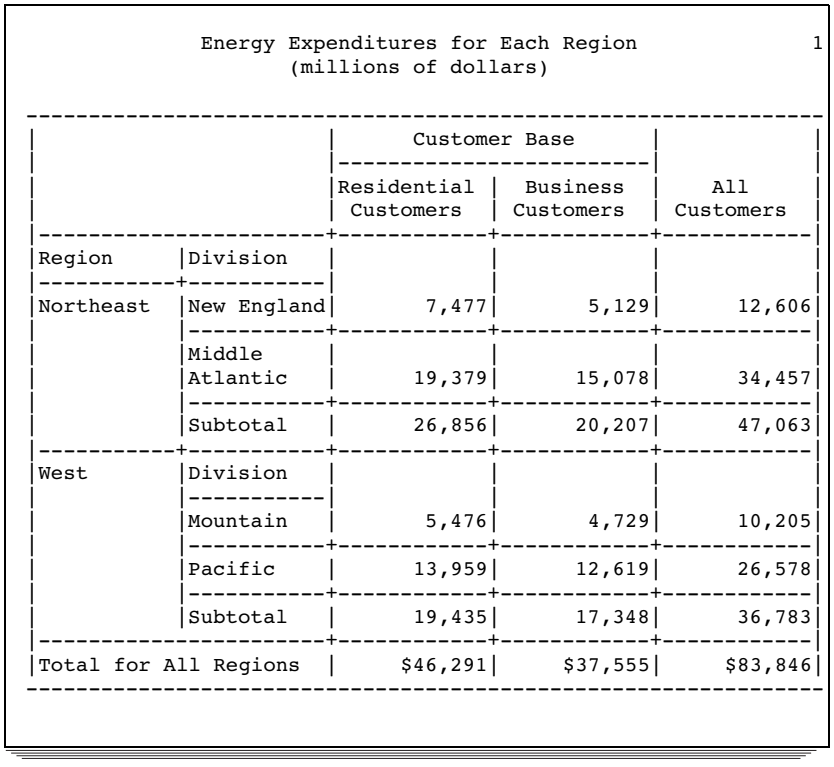
```
    format region regfmt. division divfmt. type usetype.;
```

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)

Region	Division	Customer Base		
		Residential Customers	Business Customers	All Customers
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" on page 1608

Formats: REGFMT., DIVFMT., and USETYPE. on page 1414

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 1425. 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 Customers	Business Customers
Region	Division		
Northeast	New England	$7,477	$5,129
	Middle Atlantic	$19,379	$15,078
West	Mountain	$5,476	$4,729
	Pacific	$13,959	$12,619

Example 8: Indenting Row Headings and Eliminating Horizontal Separators

Procedure features:

PROC TABULATE statement options:
NOSEPS
TABLE statement options:
INDENT=

Data set: "ENERGY" on page 1608

Formats: REGFMT., DIVFMT., and USETYPE. on page 1414

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" on page 1608

Formats: REGFMT., DIVFMT., and USETYPE. on page 1414

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;
```

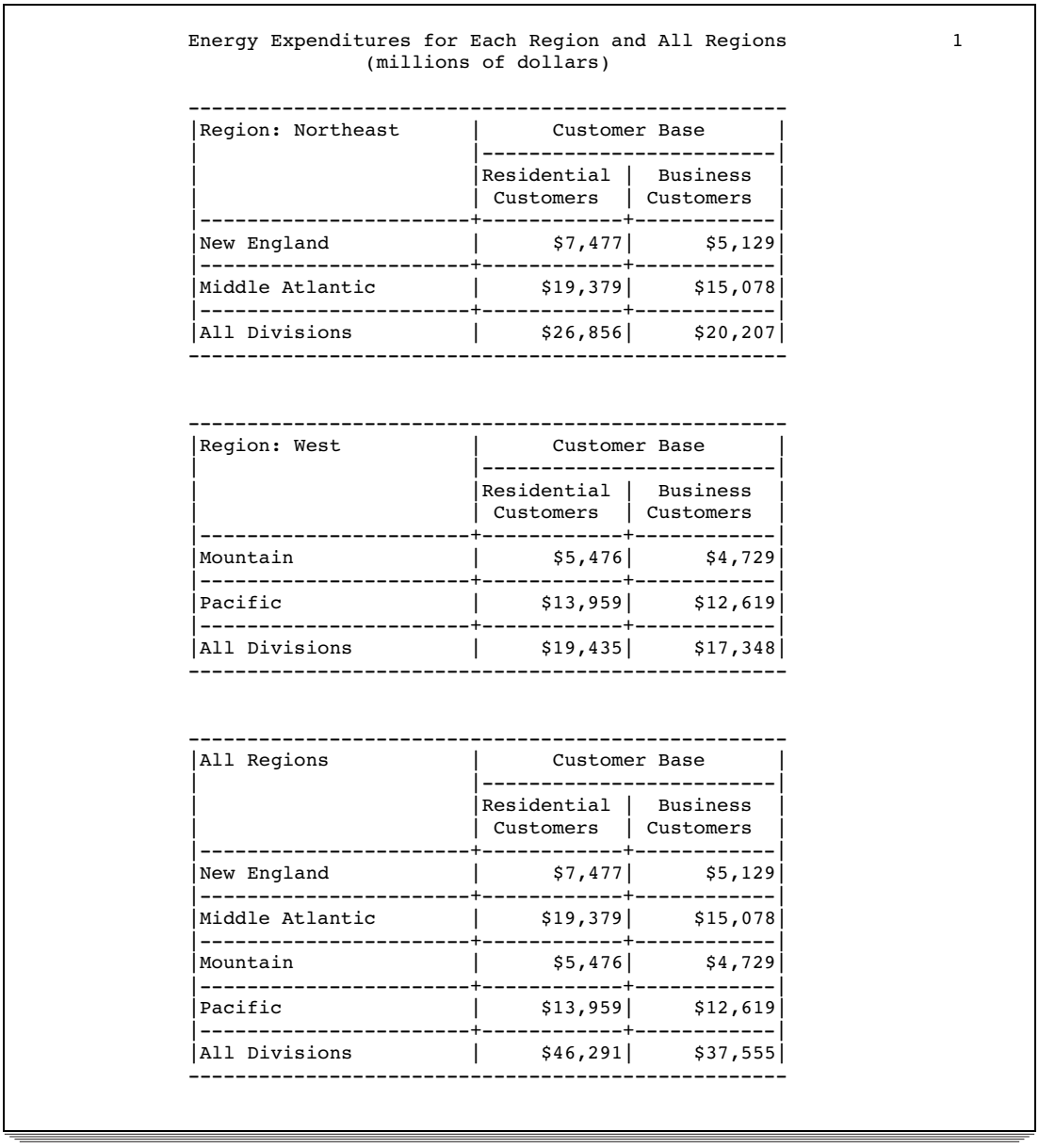
Format the output. The FORMAT statement assigns formats to the variables Region, Division, and Type.

```
    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

The following figure shows the survey form that is used to collect data.

Figure 58.15 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 sale 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 1602 creates the data set. Using missing values rather than 0s is crucial for calculating frequency counts in PROC TABULATE.

```
data customer_response;
   input Customer Factor1-Factor4 Source1-Source3
         Quality1-Quality3;
   datalines;
1 . . 1 1 1 1 . 1 . .
2 1 1 . 1 1 1 . 1 1 .
3 . . 1 1 1 1 . . . .

. . . more data lines . . .

119 . . . 1 . . . 1 . .
120 1 1 . 1 . . . . 1 .
;
```

Store the number of observations in a macro variable. The SET statement reads the descriptor portion of CUSTOMER_RESPONSE at compile time and stores the number of observations (the number of respondents) in COUNT. The SYMPUT routine stores the value of COUNT in the macro variable NUM. This variable is available for use by other procedures and DATA steps for the remainder of the SAS session. The IF 0 condition, which is always false, ensures that the SET statement, which reads the observations, never executes. (Reading observations is unnecessary.) The STOP statement ensures that the DATA step executes only once.

```
data _null_;
   if 0 then set customer_response nobs=count;
   call symput('num',left(put(count,4.)));
   stop;
run;
```

Create the PCTFMT. format. The FORMAT procedure creates a format for percentages. The PCTFMT. format writes all values with at least one digit to the left of the decimal point and with one digit to the right of the decimal point. A blank and a percent sign follow the digits.

```
proc format;
   picture pctfmt low-high='009.9 %';
run;
```

Create the report and use the default table options.

```
proc tabulate data=customer_response;
```

Specify the analysis variables. The VAR statement specifies that PROC TABULATE calculate statistics on the Factor1, Factor2, Factor3, Factor4, and Customer variables. The variable Customer must be listed because it is used to calculate the Percent column that is defined in the TABLE statement.

```
   var factor1-factor4 customer;
```

Define the table rows and columns. The TABLE statement creates a row for each factor, a column for frequency counts, and a column for the percentages. Text in quotation marks supplies headings for the corresponding row or column. The format modifiers F=7. and F=PCTFMT9. provide formats for values in the associated cells and extend the column widths to accommodate the column headings.

```
   table factor1='Cost'
         factor2='Performance'
         factor3='Reliability'
         factor4='Sales Staff',
         (n='Count'*f=7. pctn<customer>='Percent'*f=pctfmt9.) ;
```

Specify the titles.

```
   title 'Customer Survey Results: Spring 1996';
   title3 'Factors Influencing the Decision to Buy';
run;
```

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

The following figure shows the survey form that is used to collect data.

Figure 58.16 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 questions 5. Use codes 0-8 for 6-19.

0 Do not listen at that time

1	Rock	5	Classical
2	Top 40	6	Easy Listening
3	Country	7	News/Information/Talk
4	Jazz	8	Other

5. ____What style of music or radio programming do you most often listen to?

On a typical WEEKDAY, what kind of radio programming do you listen to	On a typical WEEKEND-DAY, what kind of radio programming do you listen to
6. ____from 6-9 a.m.?	13. ____from 6-9 a.m.?
7. ____from 9 a.m. to noon?	14. ____from 9 a.m. to noon?
8. ____from noon to 1 p.m.?	15. ____from noon to 1 p.m.?
9. ____from 1-4 p.m.?	16. ____from 1-4 p.m.?
10. ____from 4-6 p.m.?	17. ____from 4-6 p.m.?
11. ____from 6-10 p.m.?	18. ____from 6-10 p.m.?
12. ____from 10 p.m. to 2 a.m.?	19. ____from 10 p.m. to 2 a.m.?

An external file on page 1629 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 58.16 on page 1442.) 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 1445 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. The following figure illustrates the transposition. PROC TABULATE uses this new data set to create the crosstabular report.

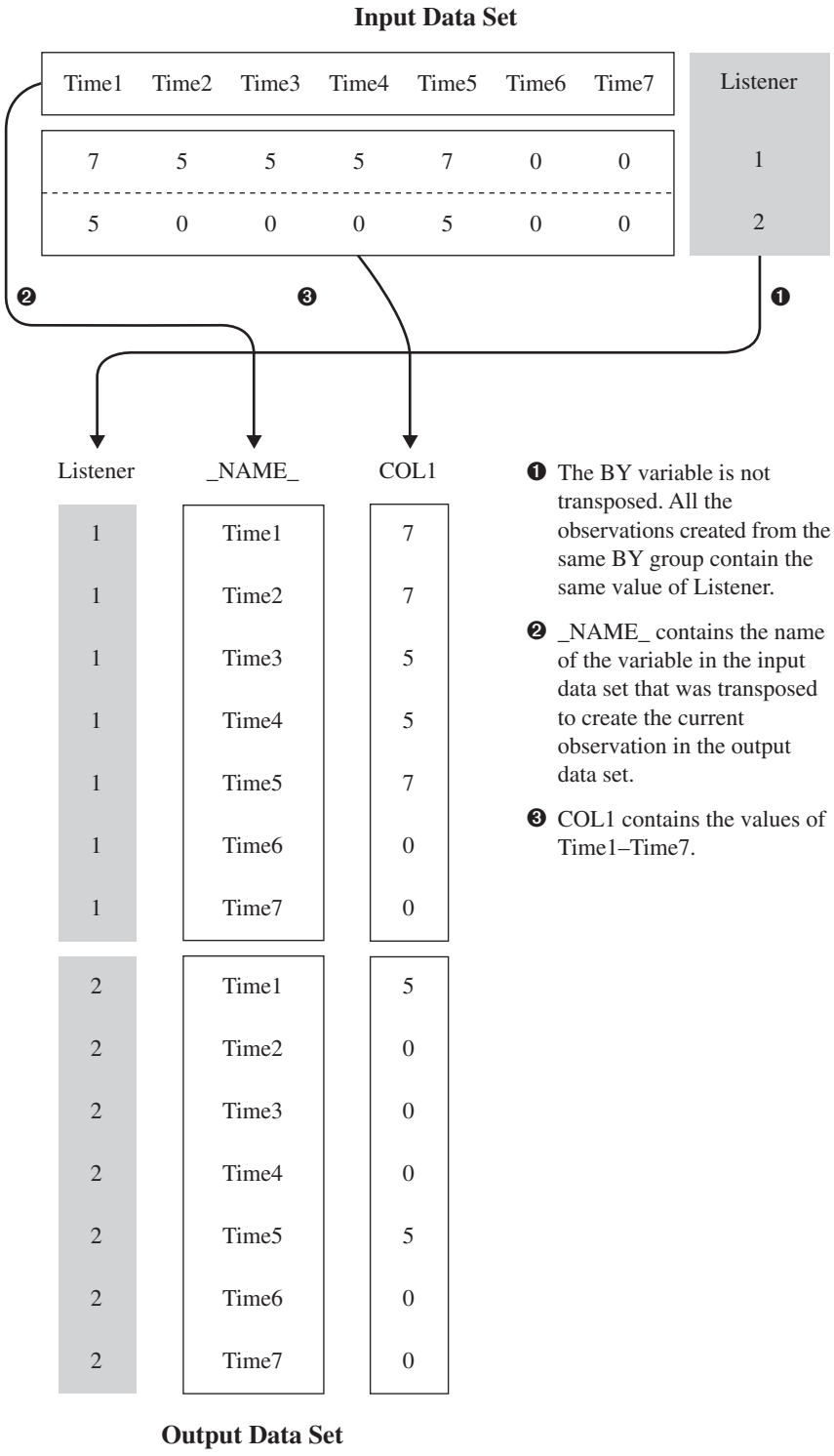
PROC TRANSPOSE restructures data so that values that were stored in one observation are written to one variable. You can specify which variables you want to

transpose. This section illustrates how PROC TRANSPOSE reshapes the data. The following section explains the PROC TRANSPOSE step in this example.

When you transpose with BY processing, as this example does, you create from each BY group one observation for each variable that you transpose. In this example, Listener is the BY variable. Each observation in the input data set is a BY group because the value of Listener is unique for each observation.

This example transposes seven variables, Time1 through Time7. Therefore, the output data set has seven observations from each BY group (each observation) in the input data set.

Figure 58.17 Transposing Two Observations



Understanding the PROC TRANSPOSE Step

Here is a detailed explanation of the PROC TRANSPOSE step that reshapes the data:

```
proc transpose data=radio  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.

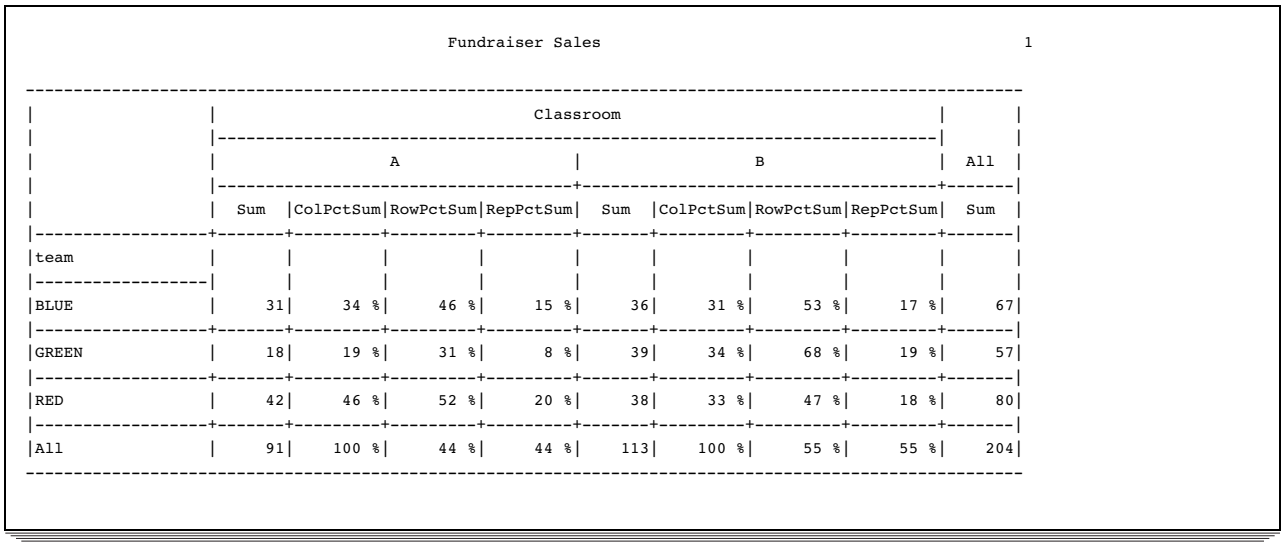
The last column of the report summarizes sales for the row for all classrooms.

```
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 1
1 2  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 3
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 4  1 4  1 4  1 4
1 4  1 4  1 4  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 2  1 3  1 3  1 3  1 3  1 4  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 1  2 2  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 4  2 1  2 3  2 3
2 3  2 3  2 3  2 3  2 4  2 4  2 4  2 4
2 4  2 4  2 1  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 2
2 3  2 3  2 4  2 4  2 4  2 1  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 heading 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 1454.

```
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 1
within Job Classes

		Gender		All
		Female	Male	Employees
Job Class				
Technical	Number of employees	16	18	34
	Percent of row total	47.06	52.94	100.00
	Percent of column total	26.23	29.03	27.64
	Percent of total	13.01	14.63	27.64
Manager/Supervisor	Number of employees	20	15	35
	Percent of row total	57.14	42.86	100.00
	Percent of column total	32.79	24.19	28.46
	Percent of total	16.26	12.20	28.46
Clerical	Number of employees	14	14	28
	Percent of row total	50.00	50.00	100.00
	Percent of column total	22.95	22.58	22.76
	Percent of total	11.38	11.38	22.76
Administrative	Number of employees	11	15	26
	Percent of row total	42.31	57.69	100.00
	Percent of column total	18.03	24.19	21.14
	Percent of total	8.94	12.20	21.14
All Jobs	Number of employees	61	62	123
	Percent of row total	49.59	50.41	100.00
	Percent of column total	100.00	100.00	100.00
	Percent of total	49.59	50.41	100.00

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';
```

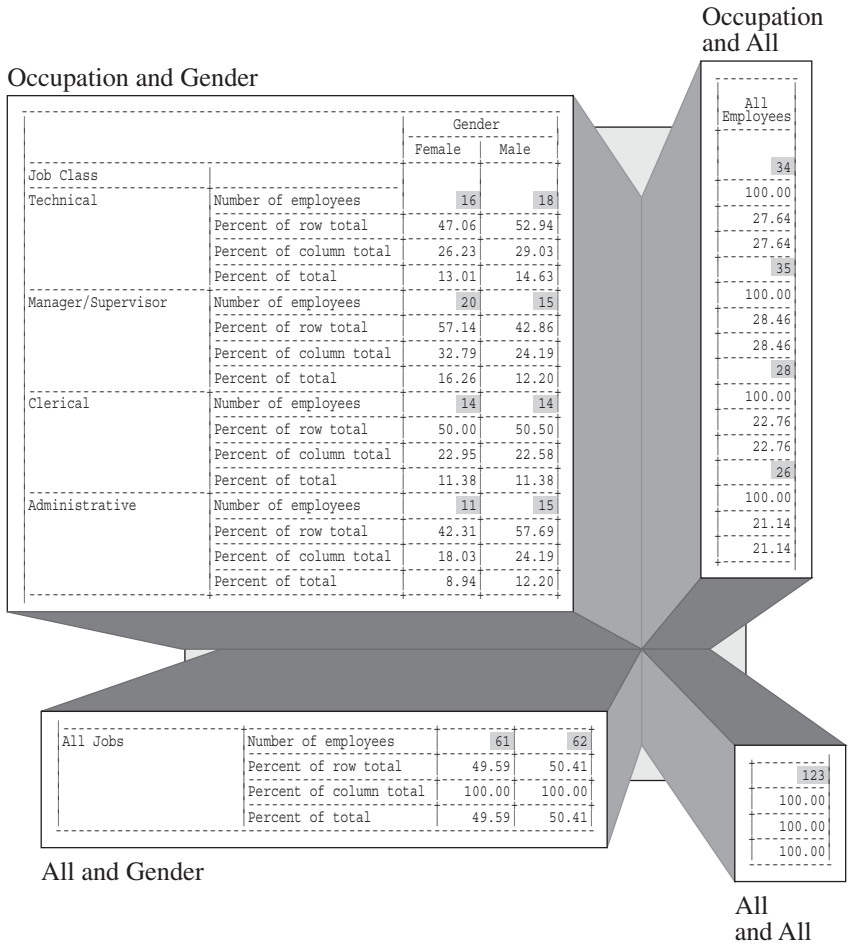
The table is a concatenation of four subtables. In this report, each subtable is a crossing of one class variable in the row dimension and one class variable in the column dimension. Each crossing establishes one or more categories. A *category* is a combination of unique values of class variables, such as **`female, technical`** or **`all, clerical`**. The following table describes each subtable.

Table 58.10 Contents of Subtables

Class variables contributing to the subtable	Description of frequency counts	Number of categories
Occupation and Gender	number of females in each job or number of males in each job	8
All and Gender	number of females or number of males	2
Occupation and All	number of people in each job	4
All and All	number of people in all jobs	1

The following figure highlights these subtables and the frequency counts for each category.

Figure 58.18 Illustration of the Four Subtables



		Gender	
		Female	Male
Job Class			
Technical	Number of employees	16	18
	Percent of row total	47.06	52.94
	Percent of column total	26.23	29.03
	Percent of total	13.01	14.63
Manager/Supervisor	Number of employees	20	15
	Percent of row total	57.14	42.86
	Percent of column total	32.79	24.19
	Percent of total	16.26	12.20
Clerical	Number of employees	14	14
	Percent of row total	50.00	50.50
	Percent of column total	22.95	22.58
	Percent of total	11.38	11.38
Administrative	Number of employees	11	15
	Percent of row total	42.31	57.69
	Percent of column total	18.03	24.19
	Percent of total	8.94	12.20

All Employees
34
100.00
27.64
27.64
35
100.00
28.46
28.46
28
100.00
22.76
22.76
26
100.00
21.14
21.14

All Jobs	Number of employees	61	62
	Percent of row total	49.59	50.41
	Percent of column total	100.00	100.00
	Percent of total	49.59	50.41

123
100.00
100.00
100.00

Interpreting Denominator Definitions

The following fragment of the TABLE statement defines the denominator definitions for this report. The PCTN keyword and the denominator definitions are highlighted.

```
table (occupation='Job Class' all='All Jobs')
          *(n='Number of employees'*f=5.
            pctn<gender all>='Row percent'
            pctn<occupation all>='Column percent'
            pctn='Percent of total'),
```

Each use of PCTN nests a row of statistics within each value of Occupation and All. Each denominator definition tells PROC TABULATE which frequency counts to sum for the denominators in that row. This section explains how PROC TABULATE interprets these denominator definitions.

Row Percentages

The part of the TABLE statement that calculates the row percentages and that labels the row is

```
pctn<gender all>='Row percent'
```

Consider how PROC TABULATE interprets this denominator definition for each subtable.

Figure 58.19 Subtable 1: Occupation and Gender

Gender Distribution
within Job Classes

		Gender		All Employees
		Female	Male	
Job Class				
Technical	Number of employees	16	18	34
	Percent of row total	47.06	52.94	100.00
	Percent of column total	26.23	29.03	27.64
	Percent of total	13.01	14.63	27.64
Manager/Supervisor	Number of employees	20	15	35
	Percent of row total	57.14	42.86	100.00
	Percent of column total	32.79	24.19	28.46
	Percent of total	16.26	12.20	28.46
Clerical	Number of employees	14	14	28
	Percent of row total	50.00	50.00	100.00
	Percent of column total	22.95	22.58	22.76
	Percent of total	11.38	11.38	22.76
Administrative	Number of employees	11	15	26
	Percent of row total	42.31	57.69	100.00
	Percent of column total	18.03	24.19	21.14
	Percent of total	8.94	12.20	21.14
All Jobs	Number of employees	61	62	123
	Percent of row total	49.59	50.41	100.00
	Percent of column total	100.00	100.00	100.00
	Percent of total	49.59	50.41	100.00

PROC TABULATE looks at the first element in the denominator definition, Gender, and asks 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.

Figure 58.20 Subtable 2: All and Gender

Gender Distribution
within Job Classes

		Gender		All Employees
		Female	Male	
Job Class				
Technical	Number of employees	16	18	34
	Percent of row total	47.06	52.94	100.00
	Percent of column total	26.23	29.03	27.64
	Percent of total	13.01	14.63	27.64
Manager/Supervisor	Number of employees	20	15	35
	Percent of row total	57.14	42.86	100.00
	Percent of column total	32.79	24.19	28.46
	Percent of total	16.26	12.20	28.46
Clerical	Number of employees	14	14	28
	Percent of row total	50.00	50.00	100.00
	Percent of column total	22.95	22.58	22.76
	Percent of total	11.38	11.38	22.76
Administrative	Number of employees	11	15	26
	Percent of row total	42.31	57.69	100.00
	Percent of column total	18.03	24.19	21.14
	Percent of total	8.94	12.20	21.14
All Jobs	Number of employees	61	62	123
	Percent of row total	49.59	50.41	100.00
	Percent of column total	100.00	100.00	100.00
	Percent of total	49.59	50.41	100.00

PROC TABULATE looks at the first element in the denominator definition, Gender, and asks 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.

Figure 58.21 Subtable 3: Occupation and All

Gender Distribution
within Job Classes

		Gender		All Employees
		Female	Male	
Job Class				
Technical	Number of employees	16	18	34
	Percent of row total	47.06	52.94	100.00
	Percent of column total	26.23	29.03	27.64
	Percent of total	13.01	14.63	27.64
Manager/Supervisor	Number of employees	20	15	35
	Percent of row total	57.14	42.86	100.00
	Percent of column total	32.79	24.19	28.46
	Percent of total	16.26	12.20	28.46
Clerical	Number of employees	14	14	28
	Percent of row total	50.00	50.00	100.00
	Percent of column total	22.95	22.58	22.76
	Percent of total	11.38	11.38	22.76
Administrative	Number of employees	11	15	26
	Percent of row total	42.31	57.69	100.00
	Percent of column total	18.03	24.19	21.14
	Percent of total	8.94	12.20	21.14
All Jobs	Number of employees	61	62	123
	Percent of row total	49.59	50.41	100.00
	Percent of column total	100.00	100.00	100.00
	Percent of total	49.59	50.41	100.00

PROC TABULATE looks at the first element in the denominator definition, Gender, and asks 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. △

Figure 58.22 Subtable 4: All and All

Gender Distribution
within Job Classes

		Gender		All Employees
		Female	Male	
Job Class				
Technical	Number of employees	16	18	34
	Percent of row total	47.06	52.94	100.00
	Percent of column total	26.23	29.03	27.64
	Percent of total	13.01	14.63	27.64
Manager/Supervisor	Number of employees	20	15	35
	Percent of row total	57.14	42.86	100.00
	Percent of column total	32.79	24.19	28.46
	Percent of total	16.26	12.20	28.46
Clerical	Number of employees	14	14	28
	Percent of row total	50.00	50.00	100.00
	Percent of column total	22.95	22.58	22.76
	Percent of total	11.38	11.38	22.76
Administrative	Number of employees	11	15	26
	Percent of row total	42.31	57.69	100.00
	Percent of column total	18.03	24.19	21.14
	Percent of total	8.94	12.20	21.14
All Jobs	Number of employees	61	62	123
	Percent of row total	49.59	50.41	100.00
	Percent of column total	100.00	100.00	100.00
	Percent of total	49.59	50.41	100.00

PROC TABULATE looks at the first element in the denominator definition, Gender, and asks 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.

Figure 58.23 Subtable 1: Occupation and Gender

Gender Distribution
within Job Classes

		Gender		All Employees
		Female	Male	
Job Class				
Technical	Number of employees	16	18	34
	Percent of row total	47.06	52.94	100.00
	Percent of column total	26.23	29.03	27.64
	Percent of total	13.01	14.63	27.64
Manager/Supervisor	Number of employees	20	15	35
	Percent of row total	57.14	42.86	100.00
	Percent of column total	32.79	24.19	28.46
	Percent of total	16.26	12.20	28.46
Clerical	Number of employees	14	14	28
	Percent of row total	50.00	50.00	100.00
	Percent of column total	22.95	22.58	22.76
	Percent of total	11.38	11.38	22.76
Administrative	Number of employees	11	15	26
	Percent of row total	42.31	57.69	100.00
	Percent of column total	18.03	24.19	21.14
	Percent of total	8.94	12.20	21.14
All Jobs	Number of employees	61	62	123
	Percent of row total	49.59	50.41	100.00
	Percent of column total	100.00	100.00	100.00
	Percent of total	49.59	50.41	100.00

PROC TABULATE looks at the first element in the denominator definition, Occupation, and asks 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.

Figure 58.24 Subtable 2: All and Gender

Gender Distribution
within Job Classes

		Gender		All Employees
		Female	Male	
Job Class				
Technical	Number of employees	16	18	34
	Percent of row total	47.06	52.94	100.00
	Percent of column total	26.23	29.03	27.64
	Percent of total	13.01	14.63	27.64
Manager/Supervisor	Number of employees	20	15	35
	Percent of row total	57.14	42.86	100.00
	Percent of column total	32.79	24.19	28.46
	Percent of total	16.26	12.20	28.46
Clerical	Number of employees	14	14	28
	Percent of row total	50.00	50.00	100.00
	Percent of column total	22.95	22.58	22.76
	Percent of total	11.38	11.38	22.76
Administrative	Number of employees	11	15	26
	Percent of row total	42.31	57.69	100.00
	Percent of column total	18.03	24.19	21.14
	Percent of total	8.94	12.20	21.14
All Jobs	Number of employees	61	62	123
	Percent of row total	49.59	50.41	100.00
	Percent of column total	100.00	100.00	100.00
	Percent of total	49.59	50.41	100.00

PROC TABULATE looks at the first element in the denominator definition, Occupation, and asks 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. △

Figure 58.25 Subtable 3: Occupation and All

Gender Distribution
within Job Classes

		Gender		All Employees
		Female	Male	
Job Class				
Technical	Number of employees	16	18	34
	Percent of row total	47.06	52.94	100.00
	Percent of column total	26.23	29.03	27.64
	Percent of total	13.01	14.63	27.64
Manager/Supervisor	Number of employees	20	15	35
	Percent of row total	57.14	42.86	100.00
	Percent of column total	32.79	24.19	28.46
	Percent of total	16.26	12.20	28.46
Clerical	Number of employees	14	14	28
	Percent of row total	50.00	50.00	100.00
	Percent of column total	22.95	22.58	22.76
	Percent of total	11.38	11.38	22.76
Administrative	Number of employees	11	15	26
	Percent of row total	42.31	57.69	100.00
	Percent of column total	18.03	24.19	21.14
	Percent of total	8.94	12.20	21.14
All Jobs	Number of employees	61	62	123
	Percent of row total	49.59	50.41	100.00
	Percent of column total	100.00	100.00	100.00
	Percent of total	49.59	50.41	100.00

PROC TABULATE looks at the first element in the denominator definition, Occupation, and asks 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.

Figure 58.26 Subtable 4: All and All

Gender Distribution
within Job Classes

		Gender		All Employees
		Female	Male	
Job Class				
Technical	Number of employees	16	18	34
	Percent of row total	47.06	52.94	100.00
	Percent of column total	26.23	29.03	27.64
	Percent of total	13.01	14.63	27.64
Manager/Supervisor	Number of employees	20	15	35
	Percent of row total	57.14	42.86	100.00
	Percent of column total	32.79	24.19	28.46
	Percent of total	16.26	12.20	28.46
Clerical	Number of employees	14	14	28
	Percent of row total	50.00	50.00	100.00
	Percent of column total	22.95	22.58	22.76
	Percent of total	11.38	11.38	22.76
Administrative	Number of employees	11	15	26
	Percent of row total	42.31	57.69	100.00
	Percent of column total	18.03	24.19	21.14
	Percent of total	8.94	12.20	21.14
All Jobs	Number of employees	61	62	123
	Percent of row total	49.59	50.41	100.00
	Percent of column total	100.00	100.00	100.00
	Percent of total	49.59	50.41	100.00

PROC TABULATE looks at the first element in the denominator definition, Occupation, and asks 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. The following table summarizes the process for all subtables in this example.

Table 58.11 Denominators for Total Percentages

Class variables contributing to the subtable	Frequency counts	Total
Occupat and Gender	16, 18, 20, 15 14, 14, 11, 15	123
Occupat and All	34, 35, 28, 26	123

Class variables contributing to the subtable	Frequency counts	Total
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 1608

Formats: REGFMT, DIVFMT, and USETYPE. on page 1414

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=[fontweight=bold];
```

Specify subgroups for the analysis. The STYLE= option in the CLASS statement specifies the style element for the class variable name headings.

```
class region division type / style=[textalign=center];
```

Specify the style attributes for the class variable value headings. The STYLE= option in the CLASSLEV statement specifies the style element for the class variable level value headings.

```
classlev region division type / style=[textalign=left];
```

Specify the analysis variable and its style attributes. The STYLE= option in the VAR statement specifies a style element for the variable name headings.

```
var expenditures / style=[fontsize=3];
```

Specify the style attributes for keywords, and label the "all" keyword. The STYLE= option in the KEYWORD statement specifies a style element for keywords. The KEYLABEL statement assigns a label to the keyword.

```
keyword all sum / style=[fontwidth=wide];
keylabel all="Total";
```

Define the table rows and columns and their style attributes. The STYLE= option in the dimension expression overrides any other STYLE= specifications in PROC TABULATE that specify attributes for table cells. The STYLE= option after the slash (/) specifies attributes for parts of the table other than table cells.

```
table (region all)*(division all*[style=[backgroundcolor=yellow]]),
      (type all)*(expenditures*f=dollar10.) /
      style=[bordercolor=blue]
```

Specify the style attributes for cells with missing values. The STYLE= option in the MISSTEXT option of the TABLE statement specifies a style element to use for the text in table cells that contain missing values.

```
      misstext=[label="Missing" style=[fontweight=light]]
```

Specify the style attributes for the box above the row titles. The STYLE= option in the BOX option of the TABLE statement specifies a style element to use for text in the box above the row titles.

```
      box=[label="Region by Division by Type"
         style=[fontstyle=italic]];
```

Format the class variable values. The FORMAT statement assigns formats to Region, Division, and Type.

```
format region regfmt. division divfmt. type usetype.;
```

Specify the titles.

```
   title 'Energy Expenditures';
   title2 '(millions of dollars)';
run;
```

Close the ODS destinations.

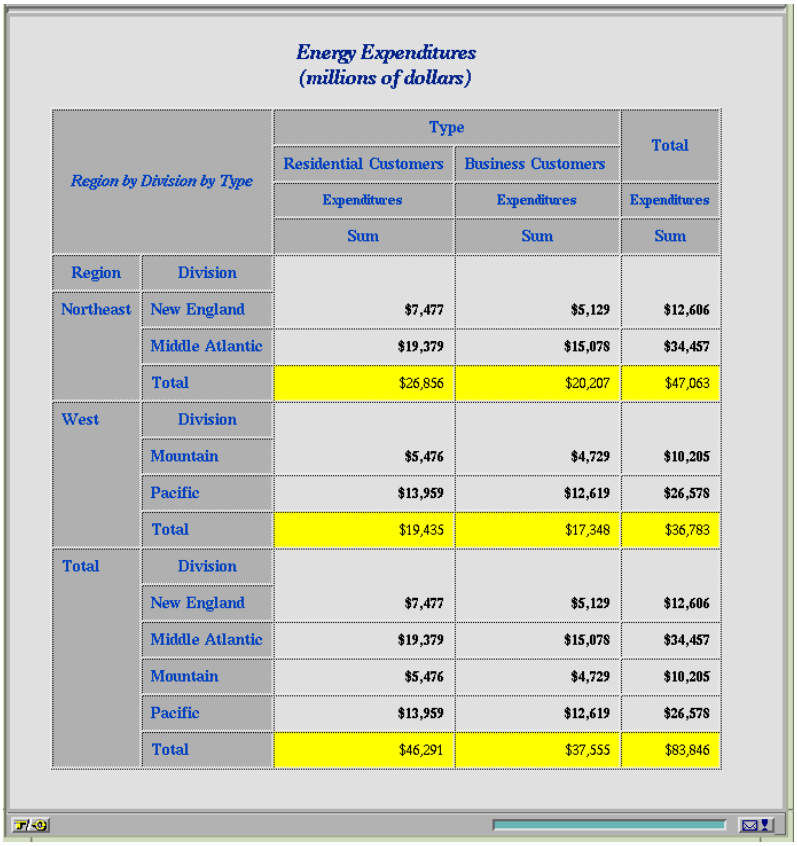
```
ods html close;
ods pdf close;
ods rtf close;
```

Listing Output

Energy Expenditures (millions of dollars) 1

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

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	Business Customers	Total
		Expenditures	Expenditures	Expenditures
		Sum	Sum	Sum
Region	Division			
Northeast	New England	$7,477	$5,129	$12,606
	Middle Atlantic	$19,379	$15,078	$34,457
	Total	$26,856	$20,207	$47,063
West	Division			
	Mountain	$5,476	$4,729	$10,205
	Pacific	$13,959	$12,619	$26,578
	Total	$19,435	$17,348	$36,783
Total	Division			
	New England	$7,477	$5,129	$12,606
	Middle Atlantic	$19,379	$15,078	$34,457
	Mountain	$5,476	$4,729	$10,205
	Pacific	$13,959	$12,619	$26,578
	Total	$46,291	$37,555	$83,846

Example 15: Style Precedence

Procedure features:

PROC TABULATE statement options:

FORMAT=

STYLE= option in

CLASSLEV statement
TABLE statement

TABLE statement

crossing (*) operator
STYLE_PRECEDENCE= option

Other features:

ODS HTML statement

FORMAT statement

Data set: “SALES” on page 1641

This example

- creates a category for each sales type, retail or wholesale, in each region
- applies the dollar format to all cells in the table
- applies an italic font style for each region and sales type
- applies a style (background = red, yellow, or orange) color based on the SYTLE_PRECEDENCE = option
- generates ODS HTML output

Program

Create the SALETYPEFMT. formats. PROC FORMAT creates formats for SALETYPE.

```
proc format;
      value $saletypefmt 'R'='Retail'
                         'W'='WholeSale';
run;
```

Specify the ODS output filename. The ODS HTML statement produces output that is written in HTML.

```
ods html file=''stylePrecedence.html'';
```

Specify the titles of the tables to be produced. Two tables will be generated. The First Table will show no style precedence whereas the Second Table will show that the color that takes precedence is based on what is specified by the STYLE_PRECEDENCE option.

```
title ''Style Precedence'';
title2 ''First Table: no precedence, Orange '';
title3 ''Second Table: style_precedence=page, Yellow'';
```

Specify the table options. The FORMAT= option specifies DOLLAR10. as the default format for the value in each table cell.

```
proc tabulate data=sales format=dollar10.;
```

Specify subgroups for the analysis. The CLASS statement separates the analysis by values of Product, Region, and SaleType.

```
class product region saletype;
```

Specify styles for the subgroups. The CLASSLEV statement specifies a style for the Region and Saletype elements.

```
classlev region saletype / style={font_style=italic};
```

Specify the analysis variable. The VAR statement specifies that PROC TABULATE calculate statistics on the Netsales variable.

```
var netsales;
```

Specify labels. The LABEL statement renames the Netsales variable to Net Sales.

```
label netsales=''Net Sales'';
```

Specify Keylabel. The KEYLABEL statement labels the universal class variable ALL to Total.

```
keylabel all=''Total'';
```

Define the table rows and columns. The TABLE statement creates a table per product per page. In this example there is one product, A100. The TABLE statement also creates a row for each formatted value of Region and creates a column for each formatted value of SaleType. Each cell that is created by these rows and columns contains the sum of the analysis variable Net Sales for all observations that contribute to that cell. The STYLE= option in the dimension expression overrides any other STYLE= specifications in PROC TABULATE that specify attributes for the table cells. In this first table, the column expression is the default and the style associated with column takes precedence. Therefore, orange will be the default color of the background.

```
table product *{style={background=red}},
region*{style={background=yellow}},
saletype*{style={background=orange}};
```

Define the table rows and columns using the STYLE_PRECEDENCE option. The TABLE statement creates a table per product per page, A100. The TABLE statement also creates a row for each formatted value of Region and creates a column for each formatted value of SaleType. Each cell that is created by these rows and columns contains the sum of the analysis variable Net Sales for all observations that contribute to that cell. The STYLE= option in the dimension expression overrides any other STYLE= specifications in PROC TABULATE that specify attributes for the table cells. In this second table, the STYLE_PRECEDENCE option is specfied on the page expression. Therefore, the style that applies to the background is yellow.

```
table product *{style={background=red}},
region*{style={background=yellow}},
saletype*{style={background=orange}} / style_precedence=page;
```

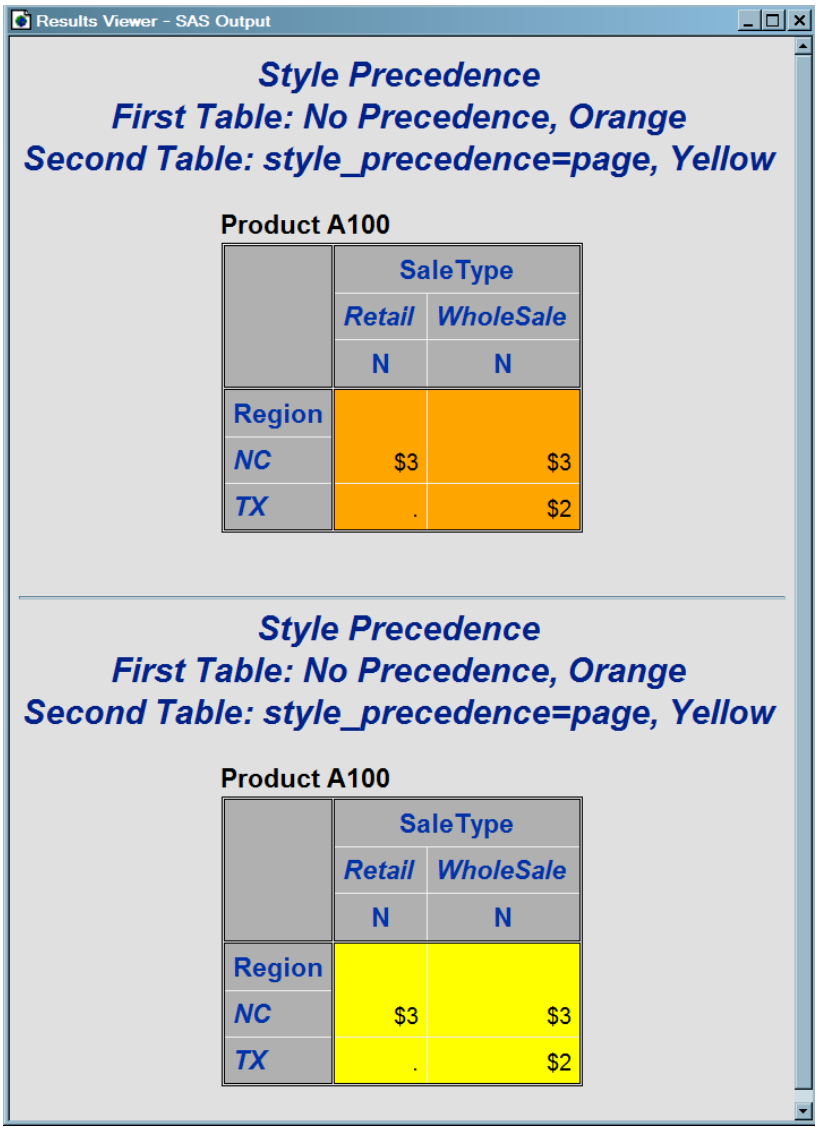
Format the output. The FORMAT statement assigns formats to the SaleType variable.

```
format saletype $saletypefmt.;
```

Run the program and close the ODS destination. Close the ODS HTML destination.

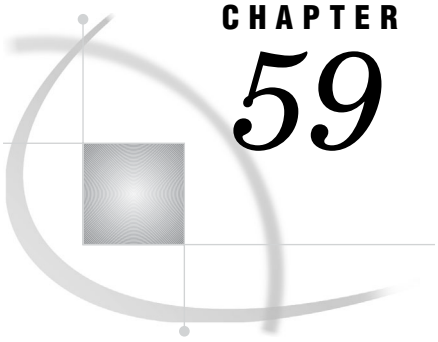
```
run;
ods html close;
```

HTML Output



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 59

The TEMPLATE Procedure

Information about the TEMPLATE Procedure **1475**

Information about the TEMPLATE Procedure

See: For complete documentation about the TEMPLATE procedure, see *SAS Output Delivery System: User's Guide*.

CHAPTER

60

The TIMEPLOT Procedure

Overview: TIMEPLOT Procedure **1477**
Syntax: TIMEPLOT Procedure **1479**
PROC TIMEPLOT Statement **1480**
BY Statement **1481**
CLASS Statement **1481**
ID Statement **1482**
PLOT Statement **1483**
Results: TIMEPLOT Procedure **1487**
Data Considerations **1487**
Procedure Output **1487**
Page Layout **1487**
Contents of the Listing **1488**
ODS Table Names **1488**
Missing Values **1488**
Examples: TIMEPLOT Procedure **1489**
Example 1: Plotting a Single Variable **1489**
Example 2: Customizing an Axis and a Plotting Symbol **1491**
Example 3: Using a Variable for a Plotting Symbol **1493**
Example 4: Superimposing Two Plots **1495**
Example 5: Showing Multiple Observations on One Line of a Plot **1497**

Overview: TIMEPLOT Procedure

The TIMEPLOT procedure plots one or more variables over time intervals. A listing of variable values accompanies the plot. Although the plot and the listing are similar to the ones produced by the PLOT and PRINT procedures, PROC TIMEPLOT output has these distinctive features:

- The vertical axis always represents the sequence of observations in the data set; thus, if the observations are in order of date or time, then the vertical axis represents the passage of time.
- The horizontal axis represents the values of the variable that you are examining. Like PROC PLOT, PROC TIMEPLOT can overlay multiple plots on one set of axes so that each line of the plot can contain values for more than one variable.
- A plot produced by PROC TIMEPLOT can occupy more than one page.
- Each observation appears sequentially on a separate line of the plot; PROC TIMEPLOT does not hide observations as PROC PLOT sometimes does.
- The listing of the plotted values can include variables that do not appear in the plot.

The following output illustrates a simple report that you can produce with PROC TIMEPLOT. This report shows sales of refrigerators for two sales representatives during the first six weeks of the year. The statements that produce the output follow. A DATA stepExample 1 on page 1489 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 60.1 Simple Report Created with PROC TIMEPLOT

```
                Weekly Sales of Refrigerators                1
                           for the
                  First Six Weeks of the Year

Month     Week      Icebox      min                          max
                                2520.04                   3550.43
                              *-------------------------------*
    1        1     3450.94     |                              I  |
    1        1     2520.04     |I                                |
    1        2     3240.67     |                        I        |
    1        2     2675.42     |     I                           |
    1        3     3160.45     |                    I            |
    1        3     2805.35     |          I                      |
    1        4     3400.24     |                             I   |
    1        4     2870.61     |            I                    |
    2        1     3550.43     |                               I |
    2        1     2730.09     |       I                         |
    2        2     3385.74     |                             I   |
    2        2     2670.93     |     I                           |
                              *-------------------------------*
```

The following output is a more complicated report of the same data set that is used to create Output 60.1. 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 1497.

Output 60.2 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: Basic Concepts in *SAS Output Delivery System: User's Guide* for details.

Tip: You can use the ATTRIB, FORMAT, LABEL, and WHERE statements. See Chapter 3, "Statements with the Same Function in Multiple Procedures," on page 35 for details. You can also use any global statements. See "Global Statements" on page 20 for a list.

PROC TIMEPLOT <*option(s)*>;

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

CLASS *variable(s)*;

ID *variable(s)*;

PLOT *plot-request(s)/option(s)*;

Task	Statement
Requests that the plots be produced	"PROC TIMEPLOT Statement" on page 1480
Produce a separate plot for each BY group	"BY Statement" on page 1481
Group data according to the values of the class variables	"CLASS Statement" on page 1481
Print in the listing the values of the variables that you identify	"ID Statement" on page 1482
Specify the plots to produce	"PLOT Statement" on page 1483

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 1495

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 1486).

BY Statement

Produces a separate plot for each BY group.

Main discussion: "BY" on page 36

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 1497

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 1484). 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 1489

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)*;

The following table summarizes the options that are available in the PLOT statement.

Table 60.1 Summary of Options for the PLOT Statement

Task	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. (Optional) Also specifies the plotting symbol to use. By default, each plot request produces a separate plot.

A plot request can have the following forms. You can mix different forms of requests in one PLOT statement (see Example 4 on page 1495).

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 1489

(*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 1491

(*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 1493

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 1486) 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 1491

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 1493

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 1497

NPP
suppresses the listing of the values of the variables that appear in the PLOT statement.

Featured in: Example 3 on page 1493

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 1495

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 1497

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 1484).

See also: "Page Layout" on page 1487

Featured in: Example 1 on page 1489

REF=*reference-value(s)*
draws lines on the plot that are perpendicular to the specified values on the horizontal axis. The values for *reference-value(s)* can be constants, or you can use the form

MEAN(*variable(s)*)

If you use this form of REF=, then PROC TIMEPLOT evaluates the mean for each variable that you list and draws a reference line for each mean.

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 1493 and Example 4 on page 1495

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 1493

REVERSE
orders the values on the horizontal axis with the largest value in the leftmost position.

Featured in: Example 4 on page 1495

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 1481.)

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 you use a CLASS statement. If you do not use a CLASS statement (see Example 1 on page 1489), then PROC TIMEPLOT prints (and plots) each observation on a separate line. If you do use a CLASS statement, then the form of the output varies depending on whether you specify a symbol variable (see "Using a Symbol Variable" on page 1482).

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 "ODS Output Object Table Names" in *SAS Output Delivery System: User's Guide*.

Table 60.2 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 headings). Plot variables and symbol variables can also appear in the plot.

Observations with missing values of a class variable form a class of observations.

In the listing, missing values appear as a period (.), a blank, or a special missing value (the letters A through Z and the underscore (_) character).

In the plot, PROC TIMEPLOT handles different variables in different ways:

- An observation or class of observations with a missing value of the plot variable does not appear in the plot.

- If you use a symbol variable (see the discussion of plot requests on page 1484), 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 headings in the listing are the variables' names. The plot uses the default plotting symbol, which is the first character of the plot variable's name.

```
                          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 1489

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 heading in the listing
- creates and uses a permanent format.

Program

Declare the PROCLIB SAS library.

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

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page. FMTSEARCH= adds the SAS 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 heading in the listing.

```
   label icebox='Refrigerator';
```

Apply the MONTHFMT. format to the Month variable. The FORMAT statement assigns a format to use for Month in the report.

```
   format month monthfmt.;
```

Specify the titles.

```
   title 'Weekly Sales of Refrigerators';
   title2 'for the';
   title3 'First Six Weeks of the Year';
run;
```

Output

The column headings in the listing are the variables' names (for Month and Week, which have no labels) and the variable's label (for Icebox, which has a label). The plotting symbol is **R** (for Refrigerator).

```
                        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 1489

Formats: MONTHFMT. on page 1491

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 library.

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

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page. FMTSEARCH= adds the SAS 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 1489

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 heading in the listing.

```
    label icebox='Refrigerators';
```

Specify the titles.

```
    title 'Weekly Sales of Stoves and Refrigerators';
    title2 'for the';
    title3 'First Six Weeks of the Year';
run;
```

Output

The column heading for the variable Icebox in the listing is the variable's label (Refrigerators). One plot uses the first letter of the value of Seller as the plotting symbol. The other plot uses the letter **R**.

```
                  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 1489
Formats: MONTHFMT. on page 1491

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 library.

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

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page. FMTSEARCH= adds the SAS 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

61

The TRANSPOSE Procedure

Overview: TRANSPOSE Procedure **1501**
What Does the TRANSPOSE Procedure Do? **1501**
What Types of Transpositions Can PROC TRANSPOSE Perform? **1502**
Syntax: TRANSPOSE Procedure **1504**
PROC TRANSPOSE Statement **1504**
BY Statement **1505**
COPY Statement **1507**
ID Statement **1508**
IDLABEL Statement **1509**
VAR Statement **1510**
Results: TRANSPOSE Procedure **1510**
Output Data Set **1510**
Output Data Set Variables **1510**
Attributes of Transposed Variables **1511**
Names of Transposed Variables **1511**
Examples: TRANSPOSE Procedure **1512**
Example 1: Performing a Simple Transposition **1512**
Example 2: Naming Transposed Variables **1513**
Example 3: Labeling Transposed Variables **1514**
Example 4: Transposing BY Groups **1516**
Example 5: Naming Transposed Variables When the ID Variable Has Duplicate Values **1518**
Example 6: Transposing Data for Statistical Analysis **1520**

Overview: TRANSPOSE Procedure

What Does the TRANSPOSE Procedure Do?

The TRANSPOSE procedure creates an output data set by restructuring the values in a SAS data set, transposing selected variables into observations. The TRANSPOSE procedure can often eliminate the need to write a lengthy DATA step to achieve the same result. Further, the output data set can be used in subsequent DATA or PROC steps for analysis, reporting, or further data manipulation.

PROC TRANSPOSE does not produce printed output. To print the output data set from the PROC TRANSPOSE step, use PROC PRINT, PROC REPORT, or another SAS reporting tool.

To create *transposed variable*, the procedure transposes the values of an observation in the input data set into values of a variable in the output data set.

What Types of Transpositions Can PROC TRANSPOSE Perform?

Simple Transposition

The following example illustrates a simple transposition. In the input data set, each *variable* represents the scores from one tester. In the output data set, each *observation* now represents the scores from one tester. Each value of _NAME_ is the name of a variable in the input data set that the procedure transposed. Thus, the value of _NAME_ identifies the source of each observation in the output data set. For example, the values in the first observation in the output data set come from the values of the variable Tester1 in the input data set. The statements that produce the output follow.

```
proc print data=proclib.product noobs;
   title 'The Input Data Set';
run;

proc transpose data=proclib.product
               out=proclib.product_transposed;
run;

proc print data=proclib.product_transposed noobs;
   title 'The Output Data Set';
run;
```

Output 61.1 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 61.2 A Transposition with BY Groups

Input Data Set 1

Location	Date	Length1	Weight1	Length2	Weight2	Length3	Weight3	Length4	Weight4
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 1516.

Syntax: TRANSPOSE Procedure

Tip: Does not support the Output Delivery System

Tip: You can use the ATTRIB, FORMAT, LABEL, and WHERE statements. See Chapter 3, "Statements with the Same Function in Multiple Procedures," on page 35 for details. You can also use any global statements. See "Global Statements" on page 20 for a list.

PROC TRANSPOSE <DATA=*input-data-set*> <DELIMITER=*delimiter*>
<LABEL=*label*> <LET>
<NAME=*name*> <OUT=*output-data-set*> <PREFIX=*prefix*> <SUFFIX=*suffix*>;
BY <DESCENDING> *variable-1*
<…<DESCENDING> *variable-n*>
<NOTSORTED>;
COPY *variable(s)*;
ID *variable*;
IDLABEL *variable*;
VAR *variable(s)*;

Task	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

Tip: You can use data set options with the DATA= and OUT= options. See "Data Set Options" on page 19 for a list.

PROC TRANSPOSE <DATA=*input-data-set*> <DELIMITER=*delimiter*>
<LABEL=*label*> <LET>
<NAME=*name*> <OUT=*output-data-set*> <PREFIX=*prefix*> <SUFFIX=*suffix*>;

Options

DATA=*input-data-set*
names the SAS data set to transpose.

Default: most recently created SAS data set

DELIMITER= *delimiter*
specifies a delimiter to use in constructing names for transposed variables in the output data set. If specified, the delimiter will be inserted between variable values if more than one variable has been specified on the ID statement.

Alias: DELIM=

See Also: "ID Statement" on page 1508

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 1518

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 1513

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 1512

PREFIX= *prefix*
specifies a prefix to use in constructing names for transposed variables in the output data set. For example, if PREFIX=VAR, then the names of the variables are VAR1, VAR2, …,VAR*n*.

Interaction: When you use PREFIX= with an ID statement, the variable name begins with the prefix value followed by the ID value.

Featured in: Example 2 on page 1513

SUFFIX= *suffix*
specifies a suffix to use in constructing names for transposed variables in the output data set.

Interaction: When you use SUFFIX= with an ID statement, the value is appended to the ID value.

BY Statement

Defines BY groups.

Restriction: Do not use PROC TRANSPOSE with a BY statement or an ID statement if another user is updating the data set at the same time.

Main discussion: "BY" on page 36

Featured in: Example 4 on page 1516

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. The procedure does not use an index if you specify NOTSORTED. The procedure defines a BY group as a set of contiguous observations that have the same values for all BY variables. If observations with the same values for the BY variables are not contiguous, then the procedure treats each contiguous set as a separate BY group.

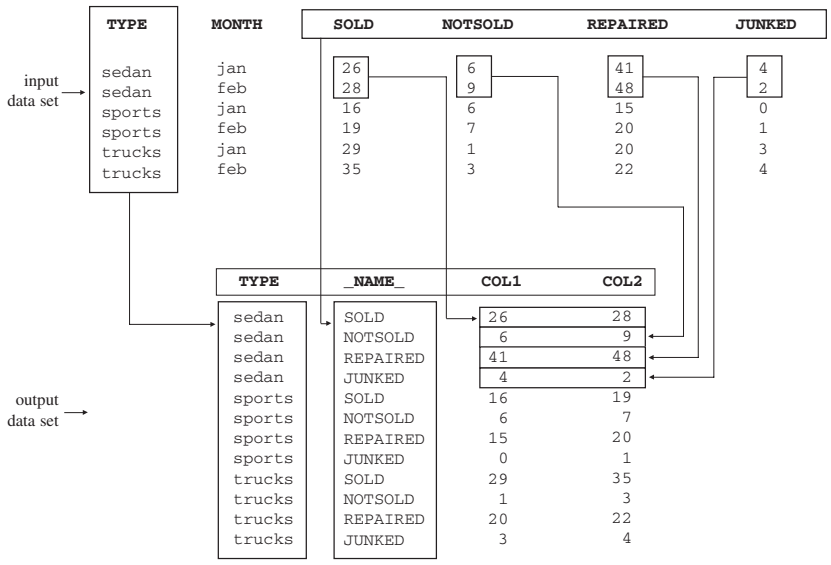
The NOBYSORTED system option disables observation sequence checking system-wide and applies to all procedures and BY statements. See the BYSORTED system option in the *SAS Language Reference: Dictionary*.

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.

The following figure shows what happens when you transpose a data set with BY groups. TYPE is the BY variable, and SOLD, NOTSOLD, REPAIRED, and JUNKED are the variables to transpose.

Figure 61.1 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 1520

COPY *variable(s)*;

Required Argument

variable(s)
names one or more variables that the COPY statement copies directly from the input data set to the output data set without transposing them.

Details

Because the COPY statement copies variables directly to the output data set, the number of observations in the output data set is equal to the number of observations in the input data set.

The procedure pads the output data set with missing values if the number of observations in the input data set is not equal to the number of variables that it transposes.

ID Statement

Specifies one or more variables in the input data set whose formatted values name the transposed variables in the output data set. When a variable name is being formed in the transposed (output) data set, the formatted values of all listed ID variables will be concatenated in the same order that the variables are listed on the ID statement. The PREFIX=, DELIMITER=, and SUFFIX= options can be used to modify the formed variable name. The PREFIX= option specifies a common character or character string to appear at the beginning of the formed variable names. The DELIMITER= option specifies a common character or character string to be inserted between the values of the ID variables. The SUFFIX= option specifies a common character or character string to be appended to the end of each formed variable name.

Restriction: You cannot use PROC TRANSPOSE with an ID statement or a BY statement with an engine that supports concurrent access if another user is updating the data set at the same time.

Tip: If the value of any ID variable is missing, then PROC TRANSPOSE writes a warning message to the log. The procedure does not transpose observations that have a missing value for any ID variable.

Featured in: Example 2 on page 1513

ID *variable(s)*;

Required Argument

variable(s)
names one or more variables whose formatted values are used to form the names of 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. It transposes the observation that contains the last occurrence of the duplicate ID value.

When multiple ID variables are specified: (or if a BY statement is used within a BY group) the combination of formatted ID variable values should be unique within the data set. If the combination is not unique, then PROC TRANSPOSE will issue a warning message and stop processing unless the LET option has been specified.

Making Variable Names out of Numeric Values

When you use a numeric variable as an ID variable, PROC TRANSPOSE changes the formatted ID value into a valid SAS name.

SAS variable names cannot begin with a number. When the first character of the formatted value is numeric, the procedure prefixes an underscore to the value, this action truncates the last character of a 32-character value. Remaining invalid characters are replaced by underscores. The procedure truncates to 32 characters any ID value that is longer than 32 characters when the procedure uses that value to name a transposed variable.

If the formatted value looks like a numeric constant, then PROC TRANSPOSE changes the characters +, –, and . to P, N, and D, respectively. If the formatted value has characters that are not numeric, then PROC TRANSPOSE changes the characters +, –, and . to underscores.

Note: If the value of the VALIDVARNAME system option is V6, then PROC TRANSPOSE truncates transposed variable names to 8 characters. △

Missing Values

If you use an ID variable that contains a missing value, then PROC TRANSPOSE writes a warning message to the log. The procedure does not transpose observations that have a missing value for one or more ID variables.

IDLABEL Statement

Creates labels for the transposed variables.

Restriction: Must appear after an ID statement.

Featured in: Example 3 on page 1514

IDLABEL *variable*;

Required Argument

variable
names the variable whose values the procedure uses to label the variables that the ID statement names. *variable* can be character or numeric.

Note: To see the effect of the IDLABEL statement, print the output data set with the PRINT procedure by using the LABEL option. You can also print the contents of the output data set by using the CONTENTS statement in the DATASETS procedure. △

VAR Statement

Lists the variables to transpose.

Featured in:

Example 4 on page 1516
Example 6 on page 1520

VAR *variable(s)*;

Required Argument

variable(s)
names one or more variables to transpose.

Details

- If you omit the VAR statement, then the TRANSPOSE procedure transposes all numeric variables in the input data set that are not listed in another statement.
- You must list character variables in a VAR statement if you want to transpose them.

  Note: If the procedure is transposing any character variable, then all transposed variables will be character variables. △

Results: TRANSPOSE Procedure

Output Data Set

The TRANSPOSE procedure always produces an output data set, regardless of whether you specify the OUT= option in the PROC TRANSPOSE statement. PROC TRANSPOSE does not print the output data set. Use PROC PRINT, PROC REPORT, or some other SAS reporting tool to print the output data set.

Output Data Set Variables

The output data set contains the following variables:

- variables that result from transposing the values of each variable into an observation.
- a variable that PROC TRANSPOSE creates to identify the source of the values in each observation in the output data set. This variable is a character variable whose values are the names of the variables that are transposed from the input data set. By default, PROC TRANSPOSE names this variable _NAME_. To override the default name, use the NAME= option. The label for the _NAME_ variable is **NAME OF FORMER VARIABLE**.

- variables that PROC TRANSPOSE copies from the input data set when you use either the BY or COPY statement. These variables have the same names and values as they do in the input data set. These variables also have the same attributes (for example: type, length, label, informat, and format).
- a character variable whose values are the variable labels of the variables that are being transposed (if any of the variables that the procedure is transposing have labels). Specify the name of the variable by using the LABEL= option. The default is **_LABEL_**.

  Note: If the value of the LABEL= option or the NAME= option is the same as a variable that appears in a BY or COPY statement, then the output data set does not contain a variable whose values are the names or labels of the transposed variables. △

Attributes of Transposed Variables

- All transposed variables are the same type and length.
- If all variables that the procedure is transposing are numeric, then the transposed variables are numeric. Thus, if the numeric variable has a character string as a formatted value, then its unformatted numeric value is transposed.
- If any variable that the procedure is transposing is character, then all transposed variables are character. If you are transposing a numeric variable that has a character string as a formatted value, then the formatted value is transposed.
- The length of the transposed variables is equal to the length of the longest variable that is being transposed.

Names of Transposed Variables

PROC TRANSPOSE uses the following rules to name transposed variables:

1. An ID statement specifies a variable or variables in the input data set whose formatted values become names for the transposed variables. If multiple ID variables are specified, the name of the transposed variable is the concatenation of the values of the ID variables. If the DELMITER= option is specified, its value is inserted between the formatted values of the ID variables when the names of the transposed variables are formed.
2. The PREFIX= option specifies a prefix to use in constructing the names of transposed variables. The SUFFIX= option also specifies a suffix to append to the names of the transposed variables.
3. If you do not use an ID statement, PREFIX= option, or the SUFFIX= option, then PROC TRANSPOSE looks for an input variable called _NAME_ to get the names of the transposed variables.
4. If you do not use an ID statement or the PREFIX= option, and if the input data set does not contain a variable named _NAME_, then PROC TRANSPOSE assigns the names COL1, COL2, …, COL*n* to the transposed variables.

Examples: TRANSPOSE Procedure

Example 1: Performing a Simple Transposition

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 1512

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 data set 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 1512

This example uses the values of the variable in the IDLABEL statement to label transposed variables.

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;
```

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;
```

Display the IDLABEL variable names and label. PROC CONTENTS displays the variable names and labels.

```
proc contents data=idlabel;
run;
```

Output 1

This data set is the output data set, IDLABEL.

```
                              Student Test Scores                               1

NAME OF
 FORMER
VARIABLE    Capalleti    Dubose    Engles    Grant    Krupski    Lundsford    McBane

 Test1          94         51        95        63        80          92          75
 Test2          91         65        97        75        76          40          78
 Final          87         91        97        80        71          86          72
```

Program 2

Display the variable and label names. PROC CONTENTS will display the variable names and the labels used in the first program.

```
proc contents data=idlabel;
run;
```

Ouput 2

PROC CONTENTS displays the variables and labels.

```
Student Test Scores                                       2

                            The CONTENTS Procedure

                  Alphabetic List of Variables and Attributes

             #    Variable    Type    Len    Label

             1    Test        Char      8    NAME OF FORMER VARIABLE
             2    sn0545      Num       8    Capalleti
             8    sn0674      Num       8    McBane
             4    sn1167      Num       8    Engles
             5    sn1230      Num       8    Grant
             3    sn1252      Num       8    Dubose
             6    sn2527      Num       8    Krupski
             7    sn4860      Num       8    Lundsford
```

Example 4: Transposing BY Groups

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 data set is the output data set, FISHLENGTH. For each BY group in the original data set, PROC TRANSPOSE creates four observations, one for each variable that it is transposing. Missing values appear for the variable Measurement (renamed from COL1) when the variables that are being transposed have no value in the input data set for that BY group. Several observations have a missing value for Measurement. For example, in the last observation, a missing value appears because the input data contained no value for Length4 on 04AUG95 at Eagle Lake.

```
                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 data set 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
62

The TRANTAB Procedure

Information about the TRANTAB Procedure **1525**

Information about the TRANTAB Procedure

See: For documentation about the TRANTAB procedure, see *SAS National Language Support (NLS): Reference Guide*.

CHAPTER
63

The UNIVARIATE Procedure

Information about the UNIVARIATE Procedure **1527**

Information about the UNIVARIATE Procedure

See: The documentation for the UNIVARIATE procedure has moved to the *Base SAS Procedures Guide: Statistical Procedures*.

CHAPTER

64

The XSL Procedure (Preproduction)

Overview: XSL Procedure **1529**
What Does the Extensible Style Sheet Language (XSL) Procedure Do? **1529**
Understanding XSL **1529**
Syntax: XSL Procedure **1530**
PROC XSL Statement **1530**
Examples: XSL Procedure **1531**
Example 1: Transforming an XML Document into Another XML Document **1531**

Overview: XSL Procedure

What Does the Extensible Style Sheet Language (XSL) Procedure Do?

PROC XSL transforms an XML document into another format, such as HTML, text, or another XML document type. PROC XSL reads an input XML document, transforms it by using an XSL style sheet, and then writes an output file.

To transform the XML document, PROC XSL uses the XSLT processor Xalan-Java, which is open source software from the Apache Xalan Project. The XSLT processor implements the XSLT 1.0 standard. For information about Xalan, see the Web site **`http://xml.apache.org/xalan-j/`**. For specific information about improving transformation performance, see the Web site **`http://xml.apache.org/xalan-j/faq.html`**.

Understanding XSL

XSL is a family of transformation languages that enables you to describe how to convert files that are encoded in XML. The languages include the following:

- XSL Transformations (XSLT) for transforming an XML document
- XSL Formatting Objects (XSL-FO) for specifying the visual presentation of an XML document
- XML Path Language (XPath), which is used by XSLT, for selecting parts of an XML document

For information about XSLT standards, see the Web site **`http://www.w3.org/TR/xslt`**.

Syntax: XSL Procedure

Table of Contents: Chapter 64, “The XSL Procedure (Preproduction),” on page 1529

PROC XSL IN=*fileref* | *'external-file'*
OUT=*fileref* | *'external-file'*
XSL=*fileref* | *'external-file'*;

PROC XSL Statement

Transforms an XML document.

PROC XSL IN=*fileref* | *'external-file'*
OUT=*fileref* | *'external-file'*
XSL=*fileref* |*'external-file'*;

Task	Argument
Specify the input file	IN=
Specify the output file	OUT=
Specify the XSL style sheet	XSL=

Required Arguments

IN=*fileref* | *'external-file'*
specifies the input file. The file must be a well-formed XML document.

fileref
specifies the SAS fileref that is assigned to the input XML document. To assign a fileref, use the FILENAME statement.

'external-file'
is the physical location of the input XML document. Include the complete pathname and the filename. Enclose the physical name in single or double quotation marks. The maximum length is 200 characters.

Featured in: Example 1 on page 1531

OUT=*fileref* | *'external-file'*
specifies the output file.

fileref

specifies the SAS fileref that is assigned to the output file. To assign a fileref, use the FILENAME statement.

'external-file'
is the physical location of the output file. Include the complete pathname and the filename. Enclose the physical name in single or double quotation marks. The maximum length is 200 characters.

Featured in: Example 1 on page 1531

XSL=*fileref* | *'external-file'*
specifies the XSL style sheet to transform the XML document. The XSL style sheet is a file that describes how to transform the XML document by using the XSLT language. The XSL style sheet must be a well-formed XML document.

fileref
specifies the SAS fileref that is assigned to the XSL style sheet. To assign a fileref, use the FILENAME statement.

'external-file'
is the physical location of the XSL style sheet. Include the complete pathname and the filename. Enclose the physical name in single or double quotation marks. The maximum length is 200 characters.

Alias: XSLT

Featured in: Example 1 on page 1531

Examples: XSL Procedure

Example 1: Transforming an XML Document into Another XML Document

The following example transforms an XML document into another XML document.

This is the input XML document named XMLInput.xml, which contains data about vehicles. Each second-level repeating element describes a particular car, with the nested elements that contain information about the model and year. The make information is an attribute on the second-level repeating element.

```
<?xml version="1.0" ?>
<vehicles>
  <car make="Ford">
    <model>Mustang</model>
    <year>1965</year>
  </car>
  <car make="Chevrolet">
    <model>Nova</model>
    <year>1967</year>
  </car>
</vehicles>
```

This is the XSL style sheet named XSLTransform.xsl that describes how to transform the XML. The conversion creates <root> as the root-enclosing element and <model> as the second-level repeating element. Each <model> element in the output XML

document will include the values from the <car> element and the make= attribute from the input XML document.

```
<?xml version="1.0" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="xml" indent="yes"/>

<xsl:template match="/vehicles">
       <root> <xsl:apply-templates select="car"/> </root>
</xsl:template>

<xsl:template match="car">
 <model make="{@make}">
    <xsl:value-of select="model" />
 </model>
</xsl:template>

</xsl:stylesheet>
```

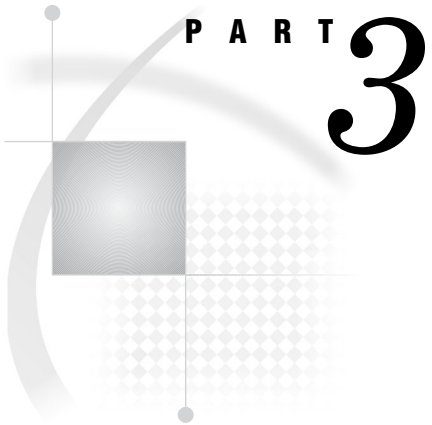
The following SAS program transforms the XML document. The procedure specifies the input XML document, the XSL style sheet, and the output XML document.

```
proc xsl
   in='C:\XMLInput.xml'
   xsl='C:\XSLTransform.xsl'
   out='C:\XMLOutput.xml';
run;
```

Here is the resulting output XML document named XMLOutput.xml.

Output 64.1 Output XML Document

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
<model make="Ford">Mustang</model>
<model make="Chevrolet">Nova</model>
</root>
```



PART 3

Appendixes

Appendix 1 . . . . . **SAS Elementary Statistics Procedures** *1535*

Appendix 2 . . . . . **Operating Environment-Specific Procedures** *1571*

Appendix 3 . . . . . **Raw Data and DATA Steps** *1573*

Appendix 4 . . . . . **ICU License** *1643*

Appendix 5 . . . . . **Recommended Reading** *1645*

APPENDIX

1

SAS Elementary Statistics Procedures

Overview **1535**
Keywords and Formulas **1536**
Simple Statistics **1536**
Descriptive Statistics **1538**
Quantile and Related Statistics **1541**
Hypothesis Testing Statistics **1543**
Confidence Limits for the Mean **1543**
Using Weights **1544**
Data Requirements for Summarization Procedures **1544**
Statistical Background **1544**
Populations and Parameters **1544**
Samples and Statistics **1545**
Measures of Location **1546**
The Mean **1546**
The Median **1546**
The Mode **1546**
Percentiles **1546**
Quantiles **1546**
Measures of Variability **1550**
The Range **1550**
The Interquartile Range **1551**
The Variance **1551**
The Standard Deviation **1551**
Coefficient of Variation **1551**
Measures of Shape **1551**
Skewness **1551**
Kurtosis **1552**
The Normal Distribution **1552**
Sampling Distribution of the Mean **1555**
Testing Hypotheses **1565**
Defining a Hypothesis **1565**
Significance and Power **1566**
Student's t Distribution **1567**
Probability Values **1568**
References **1569**

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 1537 lists the most common statistics and the procedures that compute them.

Keywords and Formulas

Simple Statistics

The base SAS procedures use a standardized set of keywords to refer to statistics. You specify these keywords in SAS statements to request the statistics to be displayed or stored in an output data set.

In the following notation, summation is over observations that contain nonmissing values of the analyzed variable and, except where shown, over nonmissing weights and frequencies of one or more:

x_i
is the nonmissing value of the analyzed variable for observation i.

f_i
is the frequency that is associated with x_i if you use a FREQ statement. If you omit the FREQ statement, then $f_i = 1$ for all i.

w_i
is the weight that is associated with x_i if you use a WEIGHT statement. The base procedures automatically exclude the values of x_i with missing weights from the analysis.

By default, the base procedures treat a negative weight as if it is equal to zero. However, if you use the EXCLNPWGT option in the PROC statement, then the procedure also excludes those values of x_i with nonpositive weights. Note that most SAS/STAT procedures, such as PROC TTEST and PROC GLM, exclude values with nonpositive weights by default.

If you omit the WEIGHT statement, then $w_i = 1$ for all i.

n
is the number of nonmissing values of x_i, $\sum f_i$. If you use the EXCLNPWGT option and the WEIGHT statement, then n is the number of nonmissing values with positive weights.

$\bar{x}$
is the mean

$$\sum w_i x_i / \sum w_i$$

s^2
is the variance

$$\frac{1}{d} \sum w_i (x_i - \bar{x})^2$$

where d is the variance divisor (the VARDEF= option) that you specify in the PROC statement. Valid values are as follows:

When VARDEF=	d equals . . .
N	n
DF	$n - 1$
WEIGHT	$\sum w_i$
WDF	$\sum w_i - 1$

The default is DF.

z_i
is the standardized variable

$$(x_i - \bar{x}) / s$$

The standard keywords and formulas for each statistic follow. Some formulas use keywords to designate the corresponding statistic.

Table 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	X	X	X		
Percentiles/Deciles/ Quartiles	X	X	X	X		
t test						
for mean=0	X	X	X	X		X
for mean=μ_0		X				
Nonparametric tests for location		X				
Tests for normality		X				
Correlation coefficients					X	
Cronbach's alpha					X	

Descriptive Statistics

The keywords for descriptive statistics are

CSS
is the sum of squares corrected for the mean, computed as

$$\sum w_i \left(x_i - \bar{x}\right)^2$$

CV
is the percent coefficient of variation, computed as

$$\left(100s\right)/\bar{x}$$

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.

Note: When QMETHOD=P2, PROC REPORT, PROC MEANS, and PROC TABULATE do not compute MODE. △

N
is the number of x_i values that are not missing. Observations with f_i less than one and w_i equal to missing or $w_i \leq 0$ (when you use the EXCLNPWGT option) are excluded from the analysis and are not included in the calculation of N.

NMISS

is the number of x_i values that are missing. Observations with f_i less than one and w_i equal to missing or $w_i \leq 0$ (when you use the EXCLNPWGT option) are excluded from the analysis and are not included in the calculation of NMISS.

NOBS
is the total number of observations and is calculated as the sum of N and NMISS. However, if you use the WEIGHT statement, then NOBS is calculated as the sum of N, NMISS, and the number of observations excluded because of missing or nonpositive weights.

RANGE
is the range and is calculated as the difference between maximum value and minimum value.

SKEWNESS | SKEW
is skewness, which measures the tendency of the deviations to be larger in one direction than in the other. When VARDEF=DF, the skewness is computed as

$$c_{3_n} \sum z_i^3$$

where c_{3_n} is $\frac{n}{(n-1)(n-2)}$. The weighted skewness is computed as

$$= c_{3_n} \sum \left(\left(x_i - \overline{x} \right) / \hat{\sigma_j} \right)^3$$
$$= c_{3_n} \sum w_i^{3/2} \left(\left(x_i - \overline{x} \right) / \hat{\sigma} \right)^3$$

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 1st percentile.

P5
is the 5th percentile.

P10
is the 10th percentile.

P90
is the 90th percentile.

P95
is the 95th percentile.

P99
is the 99th percentile.

Q1
is the lower quartile (25th percentile).

Q3
is the upper quartile (75th percentile).

QRANGE
is interquartile range and is calculated as

$$Q_3 - Q_1$$

You use the QNTLDEF= option (PCTLDEF= in PROC UNIVARIATE) to specify the method that the procedure uses to compute percentiles. Let n be the number of nonmissing values for a variable, and let $x_1, x_2, \ldots, x_n$ represent the ordered values of the variable such that x_1 is the smallest value, x_2 is next smallest value, and x_n is the largest value. For the tth percentile between 0 and 1, let $p = t/100$. Then define j as the integer part of np and g as the fractional part of np or $(n+1)\,p$, so that

$$\begin{array}{ll} np = j + g & \text{when QNTLDEF} = 1, 2, 3, \text{or } 5 \\ (n+1)\,p = j + g & \text{when QNTLDEF} = 4 \end{array}$$

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

$$y = \begin{cases} \frac{1}{2}\left(x_i + x_{i+1}\right) & \text{if } \sum_{j=1}^{i} w_j = pW \\ x_{i+1} & \text{if } \sum_{j=1}^{i} w_j < pW < \sum_{j=1}^{i+1} w_j \end{cases}$$

where w_j is the weight associated with x_i and $W = \sum_{i=1}^{n} w_i$ is the sum of the weights. When the observations have identical weights, the weighted percentiles are the same as the unweighted percentiles with QNTLDEF=5.

Table A1.2 Methods for Computing Quantile Statistics

QNTLDEF=	Description	Formula	
1	weighted average at x_{np}	$y = (1-g)\,x_j + g x_{j+1}$	
		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 $g > 0$

QNTLDEF=	Description	Formula	
4	weighted average aimed at $x_{(n+1)p}$	$y = (1-g)\,x_j + g x_{j+1}$	
		where x_{n+1} is taken to be x_n	
5	empirical distribution function with averaging	$y = \frac{1}{2}(x_j + x_{j+1})$	if $g = 0$
		$y = x_{j+1}$	if $g > 0$

Hypothesis Testing Statistics

The keywords for hypothesis testing statistics are

T
is the Student's *t* statistic to test the null hypothesis that the population mean is equal to 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 | PRT
is the two-tailed *p*-value for Student's *t* statistic, T, with $n-1$ degrees of freedom. This value is the probability under the null hypothesis of obtaining a more extreme value of T than is observed in this sample.

Confidence Limits for the Mean

The keywords for confidence limits are

CLM
is the two-sided confidence limit for the mean. A two-sided $100\,(1-\alpha)$percent confidence interval for the mean has upper and lower limits

$$\overline{x} \pm t_{(1-\alpha/2;n-1)} \frac{s}{\sqrt{\sum w_i}}$$

where s is $\sqrt{\frac{1}{n-1}\sum (x_i - \overline{x})^2}$, $t_{(1-\alpha/2;n-1)}$ is the $(1-\alpha/2)$ critical value of the Student's *t* statistics with $n-1$ degrees of freedom, and α is the value of the ALPHA= option which by default is 0.05. Unless you use VARDEF=DF, which is the default variance divisor, CLM is set to missing.

LCLM

is the one-sided confidence limit below the mean. The one-sided $100\,(1-\alpha)$percent confidence interval for the mean has the lower limit

$$\overline{x} - t_{(1-\alpha;n-1)}\frac{s}{\sqrt{\sum w_i}}$$

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 41.

Data Requirements for Summarization Procedures

The following are the minimal data requirements to compute unweighted statistics and do not describe recommended sample sizes. Statistics are reported as missing if VARDEF=DF (the default) and the following requirements are not met:

- N and NMISS are computed regardless of the number of missing or nonmissing observations.
- SUM, MEAN, MAX, MIN, RANGE, USS, and CSS require at least one nonmissing observation.
- VAR, STD, STDERR, CV, T, PRT, and PROBT require at least two nonmissing observations.
- SKEWNESS requires at least three nonmissing observations.
- KURTOSIS requires at least four nonmissing observations.
- SKEWNESS, KURTOSIS, T, PROBT, and PRT require that STD is greater than zero.
- CV requires that MEAN is not equal to zero.
- CLM, LCLM, UCLM, STDERR, T, PRT, and PROBT require that VARDEF=DF.

Statistical Background

Populations and Parameters

Usually, there is a clearly defined set of elements in which you are interested. This set of elements is called the *universe*, and a set of values associated with these elements

is called a *population* of values. The statistical term *population* has nothing to do with people 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 might be expedient to describe a population in terms of a few measures that summarize interesting features of the distribution. One such measure, computed from the population values, is called a *parameter*. Many different parameters can be defined to measure different aspects of a distribution.

The most commonly used parameter is the (arithmetic) *mean*. If the population contains a finite number of values, then the population mean is computed as the sum of all the values in the population divided by the number of elements in the population. For an infinite population, the concept of the mean is similar but requires more complicated mathematics.

E(x) denotes the mean of a population of values symbolized by x, such as height, where E stands for *expected value*. You can also consider expected values of derived functions of the original values. For example, if x represents height, then $\mathrm{E}\left(x^2\right)$ 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 might be different from that of the most commonly used estimator. The most commonly used estimator is not necessarily the best estimator in all applications.

Measures of Location

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 $\mu = \mathrm{E}(x)$ is usually estimated by the sample mean $\bar{x}$.

The Median

The population median is the central value, lying above and below half of the population values. The sample median is the middle value when the data are arranged in ascending or descending order. For an even number of observations, the midpoint between the two middle values is usually reported as the median.

The Mode

The mode is the value at which the density of the population is at a maximum. Some densities have more than one local maximum (peak) and are said to be *multimodal*. The sample mode is the value that occurs most often in the sample. By default, PROC UNIVARIATE reports the lowest such value if there is a tie for the most-often-occurring sample value. PROC UNIVARIATE lists all possible modes when you specify the MODES option in the PROC statement. If the population is continuous, then all sample values occur once, and the sample mode has little use.

Percentiles

Percentiles, including quantiles, quartiles, and the median, are useful for a detailed study of a distribution. For a set of measurements arranged in order of magnitude, the pth percentile is the value that has p percent of the measurements below it and (100–p) percent above it. The median is the 50th percentile. Because it might not be possible to divide your data so that you get exactly the desired percentile, the UNIVARIATE procedure uses a more precise definition.

The upper quartile of a distribution is the value below which 75 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.

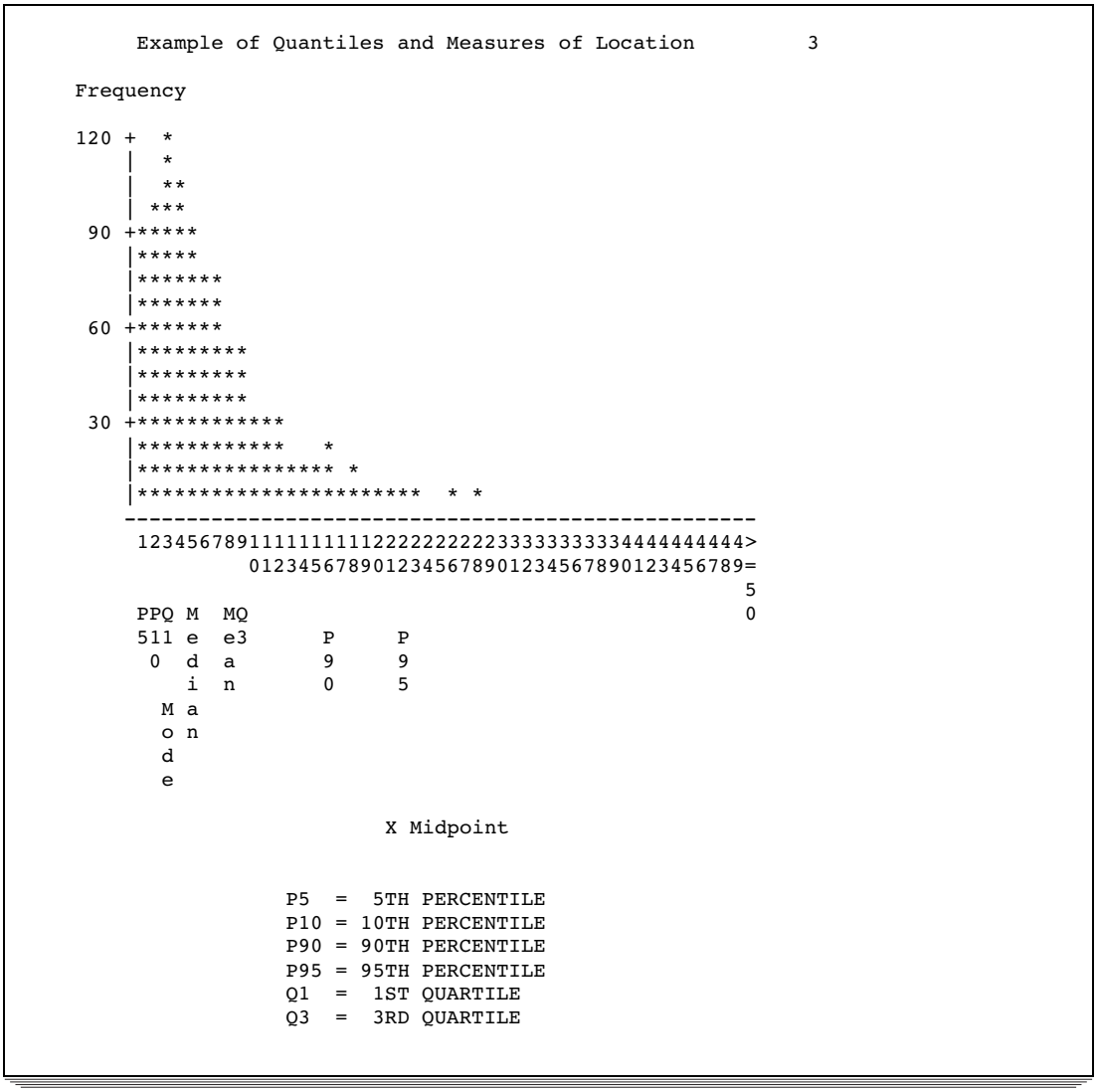
```
options nodate pageno=1 linesize=80 pagesize=52;

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
```



Measures of Variability

Another group of statistics is important in studying the distribution of a population. These statistics measure the *variability*, also called the spread, of values. In the definitions given in the sections that follow, notice that if the entire sample is changed by the addition of a fixed amount to each observation, then the values of these statistics are unchanged. If each observation in the sample is multiplied by a constant, however, then the values of these statistics are appropriately rescaled.

The Range

The sample range is the difference between the largest and smallest values in the sample. For many populations, at least in statistical theory, the range is infinite, so the sample range might not tell you much about the population. The sample range tends to increase as the sample size increases. If all sample values are multiplied by a constant, then the sample range is multiplied by the same constant.

The Interquartile Range

The interquartile range is the difference between the upper and lower quartiles. If all sample values are multiplied by a constant, then the sample interquartile range is multiplied by the same constant.

The Variance

The population variance, usually denoted by σ^2, is the expected value of the squared difference of the values from the population mean:

$$\sigma^2 = \mathrm{E}(x - \mu)^2$$

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 might tend to be larger in magnitude than negative deviations, or vice versa. *Skewness* is a measure of the tendency of the deviations to be larger in one direction than in the other. For example, the data in the last example are skewed to the right.

Population skewness is defined as

$$\mathrm{E}\,(x-\mu)^3\,/\sigma^3$$

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 might merit attention even in small samples because such values indicate that statistical methods that are based on normality assumptions might be inappropriate.

The Normal Distribution

One especially important family of theoretical distributions is the *normal* or *Gaussian* distribution. A normal distribution is a smooth symmetric function often referred to as "bell-shaped." Its skewness and kurtosis are both zero. A normal distribution can be completely specified by only two parameters: the mean and the standard deviation. Approximately 68 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 bell-like, normal density.

```
options nodate pageno=1 linesize=80 pagesize=52;

title '10000 Obs Sample from a Normal Distribution';
title2 'with Mean=50 and Standard Deviation=10';

data normaldat;
   drop n;
   do n=1 to 10000;
      X=10*rannor(53124)+50;
      output;
   end;
run;

proc univariate data=normaldat nextrobs=0 normal
                           mu0=50 loccount;
   var x;
run;


proc format;
   picture msd
      20='20 3*Std' (noedit)
      30='30 2*Std' (noedit)
      40='40 1*Std' (noedit)
      50='50 Mean ' (noedit)
      60='60 1*Std' (noedit)
      70='70 2*Std' (noedit)
      80='80 3*Std' (noedit)
   other=' ';
run;
options linesize=80 pagesize=42;

proc chart;
   vbar x / midpoints=20 to 80 by 2;
   format x msd.;
run;
```

```
          10000 Obs Sample from a Normal Distribution          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.

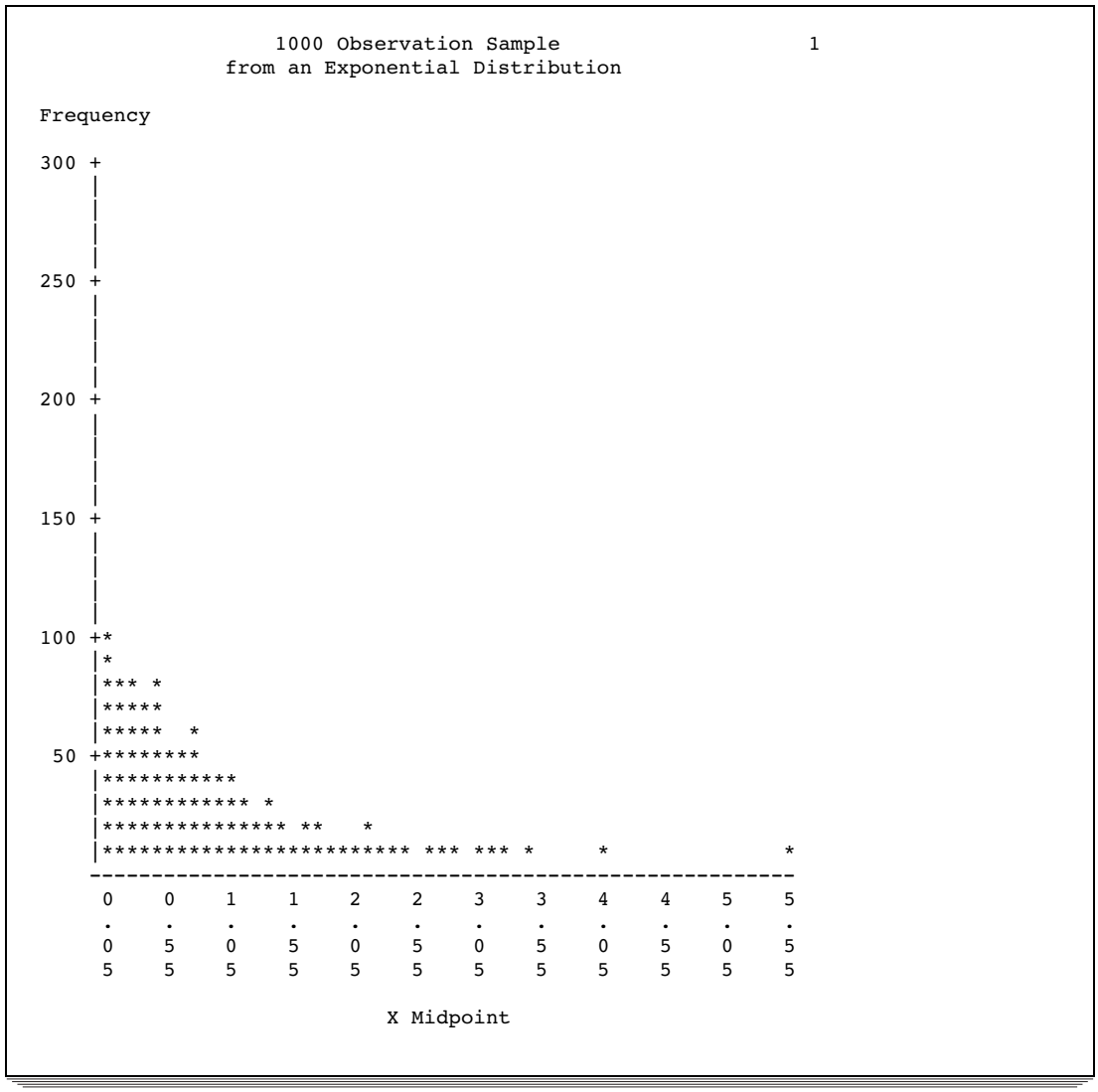
The following example consists of three separate programs that show how the sampling distribution of the mean can be approximated by a normal distribution as the sample size increases. The first DATA step uses the RANEXP function to create a sample of 1000 observations from an exponential distribution. 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.

The following example is an example of a nonnormal distribution. The population skewness is 2.00, which is close to the sample skewness of 1.97. The population kurtosis is 6.00, but the sample kurtosis is only 4.80.

```
options nodate pageno=1 linesize=80 pagesize=42;

title '1000 Observation Sample';
title2 'from an Exponential Distribution';

data expodat;
   drop n;
   do n=1 to 1000;
      X=ranexp(18746363);
      output;
   end;
run;
proc format;
    value axisfmt
      .05='0.05'
      .55='0.55'
     1.05='1.05'
     1.55='1.55'
     2.05='2.05'
     2.55='2.55'
     3.05='3.05'
     3.55='3.55'
     4.05='4.05'
     4.55='4.55'
     5.05='5.05'
     5.55='5.55'
     other=' ';
run;

proc chart data=expodat ;
   vbar x / axis=300
            midpoints=0.05 to 5.55 by .1;
   format x axisfmt.;
run;
```

```
options pagesize=64;

proc univariate data=expodat noextrobs=0 normal
                mu0=1;
   var x;
```

```
run;
```



1000 Observation Sample 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 because the sampling distribution is closer to a normal distribution than is the original exponential distribution. The CHART procedure displays a histogram of the 1000-sample means. The shape of the histogram is much closer to a bell-like, normal density, but it is still distinctly lopsided.

```
options nodate pageno=1 linesize=80 pagesize=48;

title '1000 Sample Means with 10 Obs per Sample';
title2 'Drawn from an Exponential Distribution';

data samp10;
   drop n;
   do Sample=1 to 1000;
      do n=1 to 10;
```

```
         X=ranexp(433879);
         output;
       end;
   end;

proc means data=samp10 noprint;
   output out=mean10 mean=Mean;
   var x;
   by sample;
run;


 proc format;
     value axisfmt
       .05='0.05'
       .55='0.55'
      1.05='1.05'
      1.55='1.55'
      2.05='2.05'
      other=' ';
 run;

proc chart data=mean10;
   vbar mean/axis=300
            midpoints=0.05 to 2.05 by .1;
   format mean axisfmt.;
run;


options pagesize=64;
proc univariate data=mean10 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, $H_1: \mu < 0$, and that the students are overweight on the average, $H_2: \mu > 0$.

The null hypothesis is so called because in many situations it corresponds to the assumption of "no effect" or "no difference." However, this interpretation is not appropriate for all testing problems. The null hypothesis is like a straw man that can be toppled by statistical evidence. You decide between the alternative hypotheses according to which way the straw man falls.

A naive way to approach this problem would be to look at the sample mean $\bar{x}$ and decide among the three hypotheses according to the following rule:

- If $\bar{x} < 0$, then decide on $H_1: \mu < 0$.
- If $\bar{x} = 0$, then decide on $H_0: \mu = 0$.
- If $\bar{x} > 0$, then decide on $H_2: \mu > 0$.

The trouble with this approach is that there might be a high probability of making an incorrect decision. If H_0 is true, then you are nearly certain to make a wrong decision because the chances of $\bar{x}$ being exactly zero are almost nil. If μ is slightly less than zero, so that H_1 is true, then there might be nearly a 50 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 might be to reserve judgment until you can obtain more evidence.

The question is, how far from zero must $\bar{x}$ be for you to be able to make a confident decision? The answer can be obtained by considering the sampling distribution of $\bar{x}$. If X has an approximately normal distribution, then $\bar{x}$ has an approximately normal sampling distribution. The mean of the sampling distribution of $\bar{x}$ is μ. Assume temporarily that σ, the standard deviation of X, is known to be 12. Then the standard error of $\bar{x}$ for samples of nine observations is $\sigma/\sqrt{n} = 12/\sqrt{9} = 4$.

You know that about 95 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 $H_1: \mu < 0$.
- If $-8 \leq \bar{x} \leq 8$, then reserve judgment.
- If $\bar{x} > 8$, then decide on $H_2: \mu > 0$.

If H_0 is true, then in about 95 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 that the population mean is 4. The power for detecting H_2 is the probability of getting a sample mean greater than 8. The critical value 8 is one standard error higher than the population mean 4. The chance of getting a value at least one standard deviation greater than the mean from a normal distribution is about 16 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 eight degrees of freedom.

```
data devnorm;
   title 'Deviations from Normal Weight';
   input X @@;
   datalines;
-7 -2 1 3 6 10 15 21 30
;

proc means data=devnorm maxdec=3 n mean
           std stderr t probt;
run;

title 'Student''s t Critical Value';

data _null_;
   file print;
   t=tinv(.975,8);
   put t 5.3;
run;
```

```
                Deviations from Normal Weight                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 eight 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 directions specified by the alternative hypothesis as is the value actually observed. A two-tailed p-value for a t statistic is the probability of obtaining an absolute t value that is greater than the observed absolute t value. A one-tailed p-value for a t statistic for the alternative hypothesis $\mu > \mu_0$ is the probability of obtaining a t

value greater than the observed t value. Once the p-value is computed, you can perform a hypothesis test by comparing the p-value with the desired significance level. If the p-value is less than or equal to the type I error rate of the test, then the null hypothesis can be rejected. The two-tailed p-value, labeled **`Pr > |t|`** in the PROC MEANS output, is .0606, so the null hypothesis could be rejected at the 10 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

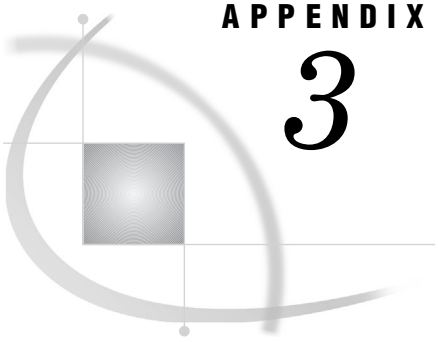
Descriptions of Operating Environment-Specific Procedures **1571**

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 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 **1574**
CENSUS **1574**
CHARITY **1575**
CONTROL Library **1577**
- *CONTROL.ALL* **1578**
- *CONTROL.BODYFAT* **1579**
- *CONTROL.CONFOUND* **1579**
- *CONTROL.CORONARY* **1579**
- *CONTROL.DRUG1* **1580**
- *CONTROL.DRUG2* **1581**
- *CONTROL.DRUG3* **1581**
- *CONTROL.DRUG4* **1581**
- *CONTROL.DRUG5* **1582**
- *CONTROL.GROUP* **1582**
- *CONTROL.MLSCL* **1585**
- *CONTROL.NAMES* **1586**
- *CONTROL.OXYGEN* **1586**
- *CONTROL.PERSONL* **1587**
- *CONTROL.PHARM* **1590**
- *CONTROL.POINTS* **1590**
- *CONTROL.PRENAT* **1590**
- *CONTROL.RESULTS* **1593**
- *CONTROL.SLEEP* **1593**
- *CONTROL.SYNDROME* **1596**
- *CONTROL.TENSION* **1597**
- *CONTROL.TEST2* **1597**
- *CONTROL.TRAIN* **1597**
- *CONTROL.VISION* **1598**
- *CONTROL.WEIGHT* **1598**
- *CONTROL.WGHT* **1600**

CUSTOMER_RESPONSE **1602**
DJIA **1604**
EDUCATION **1605**
EMPDATA **1606**
ENERGY **1608**
EXP Library **1609**
- *EXP.RESULTS* **1609**
- *EXP.SUR* **1609**

EXPREV **1610**
GROC **1611**
MATCH_11 **1612**

PROCLIB.DELAY **1613**
PROCLIB.EMP95 **1614**
PROCLIB.EMP96 **1615**
PROCLIB.INTERNAT **1616**
PROCLIB.LAKES **1616**
PROCLIB.MARCH **1617**
PROCLIB.PAYLIST2 **1618**
PROCLIB.PAYROLL **1618**
PROCLIB.PAYROLL2 **1621**
PROCLIB.SCHEDULE **1622**
PROCLIB.STAFF **1625**
PROCLIB.SUPERV **1628**
RADIO **1629**
SALES **1641**

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
;
```

CONTROL Library

The following are the contents of the CONTROL library that is used in the DATASETS procedure section.

```
                                  Directory

      Libref         CONTROL
      Engine         V9
      Physical Name  \myfiles\control
      File Name      \myfiles\control


              Member   Obs, Entries                          File
 #  Name      Type      or Indexes   Vars  Label              Size  Last Modified

 1  A1        CATALOG       23                             62464  04Jan02:14:20:12
 2  A2        CATALOG        1                             17408  04Jan02:14:20:12
 3  ALL       DATA          23        17                  13312  04Jan02:14:20:12
 4  BODYFAT   DATA           1         2                   5120  04Jan02:14:20:12
 5  CONFOUND  DATA           8         4                   5120  04Jan02:14:20:12
 6  CORONARY  DATA          39         4                   5120  04Jan02:14:20:12
 7  DRUG1     DATA           6         2   JAN95 Data      5120  04Jan02:14:20:12
 8  DRUG2     DATA          13         2   MAY95 Data      5120  04Jan02:14:20:12
 9  DRUG3     DATA          11         2   JUL95 Data      5120  04Jan02:14:20:12
10  DRUG4     DATA           7         2   JAN92 Data      5120  04Jan02:14:20:12
11  DRUG5     DATA           1         2   JUL92 Data      5120  04Jan02:14:20:12
12  ETEST1    CATALOG        1                             17408  04Jan02:14:20:16
13  ETEST2    CATALOG        1                             17408  04Jan02:14:20:16
14  ETEST3    CATALOG        1                             17408  04Jan02:14:20:16
15  ETEST4    CATALOG        1                             17408  04Jan02:14:20:16
16  ETEST5    CATALOG        1                             17408  04Jan02:14:20:16
17  ETESTS    CATALOG        1                             17408  04Jan02:14:20:16
18  FORMATS   CATALOG        6                             17408  04Jan02:14:20:16
19  GROUP     DATA         148        11                  25600  04Jan02:14:20:16
20  MLSCL     DATA          32         4   Multiple         5120  04Jan02:14:20:16
                                         Sclerosis Data
```

```
21  NAMES     DATA           7         4                      5120  04Jan02:14:20:16
22  OXYGEN    DATA          31         7                      9216  04Jan02:14:20:16
23  PERSONL   DATA         148        11                     25600  04Jan02:14:20:16
24  PHARM     DATA           6         3   Sugar Study        5120  04Jan02:14:20:16
25  POINTS    DATA           6         6                      5120  04Jan02:14:20:16
26  PRENAT    DATA         149         6                     17408  04Jan02:14:20:16
27  RESULTS   DATA          10         5                      5120  04Jan02:14:20:16
28  SLEEP     DATA         108         6                      9216  04Jan02:14:20:16
29  SYNDROME  DATA          46         8                      9216  04Jan02:14:20:16
30  TENSION   DATA           4         3                      5120  04Jan02:14:20:16
31  TEST2     DATA          15         5                      5120  04Jan02:14:20:16
32  TRAIN     DATA           7         2                      5120  04Jan02:14:20:16
33  VISION    DATA          16         3                      5120  04Jan02:14:20:16
34  WEIGHT    DATA          83        13   California        13312  04Jan02:14:20:16
                                          Results
35  WGHT      DATA          83        13   California        13312  04Jan02:14:20:16
                                          Results
16
```

The following are the raw data and DATA steps for all the data files in the CONTROL library.

CONTROL.ALL

```
data control.all;
    input FMTNAME $8. START $9. END $8. LABEL $20. MIN best4. MAX best4.
         DEFAULT best4. LENGTH best4. FUZZ best8. PREFIX $2. MULT best8.
         FILL $1. NOEDIT best4.  TYPE $2. SEXCL $2. EEXCL $2. HLO $7. ;
    label FMTNAME='Format name'
          START='Starting value for format'
          END='Ending value for format'
          LABEL='Format value label'
          MIN='Minimum length'
          MAX='Maximum length'
          DEFAULT='Default length'
          LENGTH='Format length'
          FUZZ='Fuzz value'
          PREFIX='Prefix characters'
          MULT='Multiplier'
          FILL='Fill character'
          NOEDIT='Is picture string noedit?'
          TYPE='Type of format'
          SEXCL='Start exclusion'
          EEXCL='End exclusion'
          HLO='Additional information';
    datalines;
BENEFIT       LOW    7304         WORDDATE20.   1 40  20  20   1E-12      0.00    0 N N N     LF
BENEFIT      7305    HIGH  ** Not Eligible **   1 40  20  20   1E-12      0.00    0 N N N
DOLLARS       LOW    HIGH             000,000   1 40   7   7   1E-12 $    1.96    0 P N N     LH
NOZEROS       LOW    0.01                 999   1 40   5   5   1E-12 . 1000.00    0 P N Y     L
NOZEROS      0.01     0.1                  99   1 40   5   5   1E-12 .  100.00    0 P N Y
NOZEROS       0.1       1               0.000   1 40   5   5   1E-12 . 1000.00    0 P N Y
NOZEROS         1    HIGH               0.000   1 40   5   5   1E-12   1000.00    0 P N N     H
BRIT          BR1     BR1          Birmingham   1 40  14  14       0      0.00    0 C N N
```

```
BRIT          BR2     BR2            Plymouth   1 40  14  14       0      0.00    0 C N N
BRIT          BR3     BR3               York    1 40  14  14       0      0.00    0 C N N
BRIT     *OTHER****OTHER*      INCORRECT CODE   1 40  14  14       0      0.00    0 C N N     0
SKILL          A      D~             Test A    1 40   6   6       0      0.00    0 C N N
SKILL          E      M~             Test B    1 40   6   6       0      0.00    0 C N N
SKILL          N      Z~             Test C    1 40   6   6       0      0.00    0 C N N
SKILL          a      d~             Test A    1 40   6   6       0      0.00    0 C N N
SKILL          e      m~             Test B    1 40   6   6       0      0.00    0 C N N
SKILL          n      z~             Test C    1 40   6   6       0      0.00    0 C N N
EVAL           0       4             _SAME_    1 40   1   1       0      0.00    0 I N N     I
EVAL           C       C                  1    1 40   1   1       0      0.00    0 I N N
EVAL           E       E                  2    1 40   1   1       0      0.00    0 I N N
EVAL           N       N                  0    1 40   1   1       0      0.00    0 I N N
EVAL           O       O                  4    1 40   1   1       0      0.00    0 I N N
EVAL           S       S                  3    1 40   1   1       0      0.00    0 I N N
;
run;
```

CONTROL.BODYFAT

```
data control.bodyfat;
    input NAME $  AGE $;
    datalines;
jeff    44
;
run;
```

CONTROL.CONFOUND

```
data control.confound;
    input SMOKING $8.  STATUS $8.  CANCER $8.  WT;
    datalines;
Yes     Single  Yes     34
Yes     Single  No      120
Yes     Married Yes     7
Yes     Married No      30
Yes     Single  Yes     2
Yes     Single  No      30
Yes     Married Yes     6
Yes     Married No      145
;
run;
```

CONTROL.CORONARY

```
 ata control.coronary;
    input SEX  ECG AGE CA;
    datalines;
0       0       28      0
0       0       34      0
0       0       38      0
```

```
0       0       41      0
0       0       44      0
0       0       45      1
0       0       46      0
0       0       47      0
0       0       50      0
0       0       51      0
0       0       51      0
0       0       53      0
0       0       55      1
0       0       59      0
0       0       60      1
0       0       32      1
0       0       33      0
0       0       35      0
0       0       39      0
0       0       40      0
0       0       46      0
0       0       48      1
0       0       49      0
0       0       49      0
0       0       52      0
0       0       53      1
0       0       54      1
0       0       55      0
0       0       57      1
0       0       46      1
0       0       48      0
0       0       57      1
0       0       60      1
0       0       30      0
0       0       34      0
0       0       36      1
0       0       38      1
0       0       39      0
0       0       42      0;
run;
```

CONTROL.DRUG1

```
data control.drug1 (label='JAN2005 DATA');
    input CHAR $8.  NUM;
    datalines;
junk    0
junk    0
junk    0
junk    0
junk    0
junk    0
;
run;
```

CONTROL.DRUG2

```
 data control.drug2 (label='MAY2005 DATA');
    input CHAR $8.  NUM;
    datalines;
junk    0
junk    0
junk    0
junk    0
junk    0
junk    0
junk    0
junk    0
junk    0
junk    0
junk    0
junk    0
;
run;
```

CONTROL.DRUG3

```
data control.drug3 (label='JUL2005 DATA');
    input CHAR $8.  NUM;
    datalines;
junk    0
junk    0
junk    0
junk    0
junk    0
junk    0
junk    0
junk    0
junk    0
junk    0
junk    0
;
run;
```

CONTROL.DRUG4

```
data control.drug4 (label='JAN2002 DATA');
    input CHAR $8.  NUM;
    datalines;
junk    0
junk    0
junk    0
junk    0
junk    0
junk    0
junk    0
;
```

```
run;
```

CONTROL.DRUG5

```
data control.drug5 (label='JUL2002 DATA');
    input CHAR $8.  NUM;
    datalines;
junk    0;
run
```

CONTROL.GROUP

```
data control.group;
    input IDNUM $ 1-4 LNAME $ 5-19 FNAME $ 20-34 CITY $ 35-49 STATE $
          50-52 SEX $ 53-54 JOBCODE $ 55-58 SALARY comma8. BIRTH
          HIRED date7. HPHONE $ 85-96;
    format salary comma8.;
    format hired date7.;
    informat hired date7.;
    datalines;
1919 ADAMS          GERALD          STAMFORD        CT M TA2   34,377    -4125 07JUN75 203/781-1255
1653 AHMAD          AZEEM           BRIDGEPORT      CT F ME2   35,109    -2631 12AUG78 203/675-7715
1400 ALVAREZ        GLORIA          NEW YORK        NY M ME1   29,770    -1515 19OCT78 212/586-0808
1350 ARTHUR         BARBARA         NEW YORK        NY F FA3   32,887    -2311 01AUG78 718/383-1549
1401 AVERY          JERRY           PATERSON        NJ M TA3   38,823    -7686 20NOV73 201/732-8787
1499 BAREFOOT       JOSEPH          PRINCETON       NJ M ME3   43,026    -6456 10JUN68 201/812-5665
1101 BASQUEZ        RICHARDO        NEW YORK        NY M SCP   18,724    -3493 04OCT78 212/586-8060
1333 BEAULIEU       ARMANDO         NEW YORK        NY M TA2   32,616    -3268 05DEC78 718/384-2849
1479 BOSTIC         MARIE           NEW YORK        NY F TA3   38,786    -1102 08OCT77 718/384-8816
1403 BOWDEN         EARL            BRIDGEPORT      CT M ME1   28,073    -1065 24DEC79 203/675-3434
1739 BOYCE          JONATHAN        NEW YORK        NY M PT1   66,518    -2560 30JAN79 212/587-1247
1658 BRADLEY        JEREMY          NEW YORK        NY M SCP   17,944    -1726 03MAR80 212/587-3622
1428 BRADY          CHRISTINE       STAMFORD        CT F PT1   68,768     -634 19NOV79 203/781-1212
1782 BROWN          JASON           STAMFORD        CT M ME2   35,346     -390 25FEB80 203/781-0019
1244 BRYANT         LEONARD         NEW YORK        NY M ME2   36,926    -3042 20JAN76 718/383-3334
1383 BURNETTE       THOMAS          NEW YORK        NY M BCK   25,824    -1434 23OCT80 718/384-3569
1574 CAHILL         MARSHALL        NEW YORK        NY M FA2   28,573    -4263 23DEC80 718/383-2338
1789 CANALES        VIVIANA         NEW YORK        NY M SCP   18,327    -5451 14APR66 212/587-9000
1404 CARTER         DONALD          NEW YORK        NY M PT2   91,377    -6882 04JAN68 718/384-2946
1437 CARTER         DOROTHY         BRIDGEPORT      CT F FA3   33,105    -4117 03SEP72 203/675-4117
1639 CARTER         KAREN           STAMFORD        CT F TA3   40,261    -5299 31JAN72 203/781-8839
1269 CASTON         FRANKLIN        STAMFORD        CT M NA1   41,691      126 01DEC80 203/781-3335
1065 CHAPMAN        NEIL            NEW YORK        NY M ME2   35,091   -10199 10JAN75 718/384-5618
1876 CHAN           CHING           NEW YORK        NY M TA3   39,676    -4971 30APR73 212/588-5634
1037 CHOW           JANE            STAMFORD        CT F TA1   28,559    -2819 16SEP80 203/781-8868
1129 COOK           BRENDA          NEW YORK        NY F ME2   34,930    -3673 20AUG79 718/383-2313
1988 COOPER         ANTHONY         NEW YORK        NY M FA3   32,218    -4412 21SEP72 212/587-1228
1405 DAVIDSON       JASON           PATERSON        NJ M SCP   18,057    -2125 29JAN80 201/732-2323
1430 DEAN           SANDRA          BRIDGEPORT      CT F TA2   32,926    -3591 30APR75 203/675-1647
1983 DEAN           SHARON          NEW YORK        NY F FA3   33,420    -3591 30APR75 718/384-1647
1134 DELGADO        MARIA           STAMFORD        CT F TA2   33,463    -1029 24DEC76 203/781-1528
1118 DENNIS         ROGER           NEW YORK        NY M PT3  111,380   -10209 21DEC68 718/383-1122
1438 DESAI          AAKASH          STAMFORD        CT F TA3   39,224    -2480 21NOV75 203/781-2229
```

```
1125 DUNLAP        DONNA          NEW YORK       NY F FA2   28,889    -1146 14DEC75 718/383-2094
1475 EATON         ALICIA         NEW YORK       NY F FA2   27,788    -3666 16JUL78 718/383-2828
1117 EDGERTON      JOSHUA         NEW YORK       NY M TA3   39,772    -3129 16AUG80 212/588-1239
1935 FERNANDEZ     KATRINA        BRIDGEPORT     CT F NA2   51,082    -6485 19OCT69 203/675-2962
1124 FIELDS        DIANA          WHITE PLAINS   NY F FA1   23,178    -4920 04OCT78 914/455-2998
1422 FLETCHER      MARIE          PRINCETON      NJ F FA1   22,455    -2764 09APR79 201/812-0902
1616 FLOWERS       ANNETTE        NEW YORK       NY F TA2   34,138     -668 07JUN81 718/384-3329
1406 FOSTER        GERALD         BRIDGEPORT     CT M ME2   35,186    -3948 20FEB75 203/675-6363
1120 GARCIA        JACK           NEW YORK       NY M ME1   28,620      257 10OCT81 718/384-4930
1094 GOMEZ         ALAN           BRIDGEPORT     CT M FA1   22,269     -636 20APR79 203/675-7181
1389 GORDON        LEVI           NEW YORK       NY M BCK   25,029    -4550 21AUG78 718/384-9326
1905 GRAHAM        ALVIN          NEW YORK       NY M PT1   65,112      109 01JUN80 212/586-8815
1407 GRANT         DANIEL         MT. VERNON     NY M PT1   68,097    -1011 21MAR78 914/468-1616
1114 GREEN         JANICE         NEW YORK       NY F TA2   32,929     -832 30JUN75 212/588-1092
1410 HARRIS        CHARLES        STAMFORD       CT M PT2   84,686    -1701 10NOV74 203/781-0937
1439 HARRISON      FELICIA        BRIDGEPORT     CT F PT1   70,737    -2854 13SEP78 203/675-4987
1409 HARTFORD      RAYMOND        STAMFORD       CT M ME3   41,552    -7924 25OCT69 203/781-9697
1408 HENDERSON     WILLIAM        PRINCETON      NJ M TA2   34,139    -4292 17OCT75 201/812-4789
1121 HERNANDEZ     MICHAEL        NEW YORK       NY M ME1   29,113      -94 10DEC79 718/384-3313
1991 HOLMES        GABRIEL        BRIDGEPORT     CT F TA1   27,646      130 15DEC80 203/675-0007
1102 HOLMES        SHANE          WHITE PLAINS   NY M TA2   34,543    -4472 18APR79 914/455-0976
1356 HOLMES        SHAWN          NEW YORK       NY M ME2   36,870    -5207 25FEB71 212/586-8411
1545 HUNTER        CLYDE          STAMFORD       CT M PT1   66,131    -4522 01JUN78 203/781-1119
1292 HUNTER        HELEN          BRIDGEPORT     CT F ME2   36,692    -2618 05JUL77 203/675-4830
1440 JACKSON       LAURA          STAMFORD       CT F ME2   35,758    -3380 12APR79 203/781-0088
1368 JEPSEN        RONALD         STAMFORD       CT M FA2   27,809    -3853 06NOV72 203/781-8413
1369 JOHNSON       ANTHONY        NEW YORK       NY M TA2   33,706    -3653 16MAR75 212/587-5385
1411 JOHNSON       JACKSON        PATERSON       NJ M FA2   27,266    -3868 04DEC77 201/732-3678
1113 JONES         LESLIE         NEW YORK       NY F FA1   22,368    -1444 20OCT79 718/383-3003
1704 JOSHI         ABHAY          NEW YORK       NY M BCK   25,466    -1947 01JUL75 718/384-0049
1900 KING          WILLIAM        NEW YORK       NY M ME2   35,106    -3505 30OCT75 718/383-3698
1126 KOSTECKA      NICHOLAS       NEW YORK       NY F TA3   40,900    -3137 24NOV68 212/586-1229
1677 KRAMER        JACKSON        BRIDGEPORT     CT M BCK   26,008    -2976 30MAR77 203/675-7432
1441 LAWRENCE      KATHY          PRINCETON      NJ F FA2   27,159     -770 26MAR79 201/812-3337
1421 LEE           RUSSELL        MT. VERNON     NY M TA2   33,156    -4738 03MAR78 914/468-9143
1119 LI            JEFF           NEW YORK       NY M TA1   26,925    -3479 09SEP76 212/586-2344
1834 LONG          RUSSELL        NEW YORK       NY M BCK   26,897       41 05JUL80 718/384-0040
1777 LUFKIN        ROY            NEW YORK       NY M PT3  109,631    -7402 24JUN69 718/383-4413
1663 MAHANNAHS     SHANTHA        NEW YORK       NY M BCK   26,453    -1813 14AUG79 212/587-7742
1106 MARSHBURN     JASPER         STAMFORD       CT M PT2   89,633    -5166 19AUG72 203/781-1457
1103 MCDANIEL      RONDA          NEW YORK       NY F FA1   23,739    -1412 26JUL80 212/586-0013
1477 MEYERS        PRESTON        BRIDGEPORT     CT M FA2   28,567    -2839 10MAR76 203/675-8125
1476 MONROE        JOYCE          STAMFORD       CT F TA2   34,804    -2039 20MAR75 203/781-2837
1379 MORGAN        ALFRED         STAMFORD       CT M ME3   42,265    -3795 13JUN72 203/781-2216
1104 MORGAN        CHRISTOPHER    NEW YORK       NY M SCP   17,947    -3170 13JUN79 718/383-9740
1009 MORGAN        GEORGE         NEW YORK       NY M TA1   28,881    -4685 29MAR80 212/586-7753
1412 MURPHEY       JOHN           PRINCETON      NJ M ME1   27,800    -5672 08DEC79 201/812-4414
1115 MURPHY        ALICE          NEW YORK       NY F FA3   32,700    -4146 03MAR68 718/384-1982
1128 NELSON        FELICIA        BRIDGEPORT     CT F TA2   32,778    -2411 23OCT78 203/675-1166
1442 NEWKIRK       SANDRA         PRINCETON      NJ F PT2   84,537    -1941 15APR76 201/812-3331
1417 NEWKIRK       WILLIAM        PATERSON       NJ M NA2   52,271    -2741 10MAR77 201/732-6611
1478 NEWTON        JAMES          NEW YORK       NY M PT2   84,204    -4525 27OCT78 212/587-5549
1673 NICHOLLS      HENRY          STAMFORD       CT M BCK   25,478     -670 18JUL79 203/781-7770
1839 NORRIS        DIANE          NEW YORK       NY F NA1   43,434     -395 06JUL81 718/384-1767
```

1347	O'NEAL	BRYAN	NEW YORK	NY	M	TA3	40,080	-1560	09SEP72 718/384-0230
1423	OSWALD	LESLIE	MT. VERNON	NY	F	ME2	35,774	-1324	22AUG78 914/468-9171
1200	OVERMAN	MICHELLE	STAMFORD	CT	F	ME1	27,817	-353	17AUG80 203/781-1835
1970	PAPI	PAOLO	NEW YORK	NY	F	FA1	22,616	-2651	15MAR79 718/383-3895
1521	PAPIA	ISMAEL	NEW YORK	NY	M	ME3	41,527	-3183	16JUL76 212/587-7603
1354	PAPIA	FRANCISCO	WHITE PLAINS	NY	F	SCP	18,336	-214	19JUN80 914/455-2337
1424	PATTERSON	RENEE	NEW YORK	NY	F	FA2	28,979	-877	14DEC77 212/587-8991
1132	PEARCE	CAROL	NEW YORK	NY	F	FA1	22,414	153	25OCT81 718/384-1986
1845	PEARSON	JAMES	NEW YORK	NY	M	BCK	25,997	-4422	25MAR68 718/384-2311
1556	PENNINGTON	MICHAEL	NEW YORK	NY	M	PT1	71,350	-2746	14DEC79 718/383-5681
1413	PETERS	RANDALL	PRINCETON	NJ	M	FA2	27,436	-2295	05JAN78 201/812-2478
1123	PETERSON	SUZANNE	NEW YORK	NY	F	TA1	28,408	307	08DEC80 718/383-0077
1907	PHELPS	WILLIAM	STAMFORD	CT	M	TA2	33,330	-4061	09JUL75 203/781-1118
1436	PORTER	SUSAN	NEW YORK	NY	F	TA2	34,476	-2757	15MAR75 718/383-5777
1385	RAYNOR	MILTON	BRIDGEPORT	CT	M	ME3	43,901	-3634	04APR74 203/675-2846
1432	REED	MARILYN	MT. VERNON	NY	F	ME2	35,328	-3708	13FEB73 914/468-5454
1111	RHODES	JEREMY	PRINCETON	NJ	M	NA1	40,587	563	03NOV80 201/812-1837
1116	RICHARDS	CASEY	NEW YORK	NY	F	FA1	22,863	-822	24MAR79 212/587-1224
1352	RIVERS	SIMON	NEW YORK	NY	M	NA2	53,799	-4044	19OCT74 718/383-3345
1555	RODRIGUEZ	JULIA	BRIDGEPORT	CT	F	FA2	27,500	-1383	07JUL80 203/675-2401
1038	RODRIGUEZ	MARIA	BRIDGEPORT	CT	F	TA1	26,534	-780	26NOV79 203/675-2048
1420	ROUSE	JEREMY	PATERSON	NJ	M	ME3	43,072	-2504	25JUL75 201/732-9834
1561	SANDERS	RAYMOND	NEW YORK	NY	M	TA2	34,515	-2951	10OCT75 212/588-6615
1434	SANDERSON	EDITH	STAMFORD	CT	F	FA2	28,623	-3458	31OCT78 203/781-1333
1414	SARKAR	ABHEEK	BRIDGEPORT	CT	M	FA1	23,645	86	15APR80 203/675-1715
1112	SAYERS	RANDY	NEW YORK	NY	M	TA1	26,906	-2586	10DEC80 718/384-4895
1390	SMART	JONATHAN	NEW YORK	NY	M	FA2	27,762	-2504	26JUN79 718/383-1141
1332	STEPHENSON	ADAM	BRIDGEPORT	CT	M	NA1	42,179	-468	07JUN79 203/675-1497
1890	STEPHENSON	ROBERT	NEW YORK	NY	M	PT2	85,897	-7467	28NOV67 718/384-9874
1429	THOMPSON	ALICE	STAMFORD	CT	F	TA1	27,940	-4322	10AUG80 203/781-3857
1107	THOMPSON	WAYNE	NEW YORK	NY	M	PT2	89,978	-6412	13FEB67 718/384-3785
1908	TRENTON	MELISSA	NEW YORK	NY	F	TA2	32,996	-749	26APR78 212/586-6262
1830	TRIPP	KATHY	BRIDGEPORT	CT	F	PT2	84,472	-5329	01FEB71 203/675-2479
1882	TUCKER	ALAN	NEW YORK	NY	M	ME3	41,539	-5285	24NOV66 718/384-0216
1050	TUTTLE	THOMAS	WHITE PLAINS	NY	M	ME2	35,168	-3090	27AUG74 914/455-2119
1425	UNDERWOOD	JENNY	STAMFORD	CT	F	FA1	23,980	-1	03MAR81 203/781-0978
1928	UPCHURCH	LARRY	WHITE PLAINS	NY	M	PT2	89,859	-6313	16JUL78 914/455-5009
1480	UPDIKE	THERESA	NEW YORK	NY	F	TA3	39,584	-5230	28MAR69 212/587-8729
1100	VANDEUSEN	RICHARD	NEW YORK	NY	M	BCK	25,005	-4045	10MAY76 212/586-2531
1995	VARNER	ELIZABETH	NEW YORK	NY	F	ME1	28,811	604	22SEP81 718/384-7113
1135	VEGA	ANNA	NEW YORK	NY	F	FA2	27,322	-4117	03APR78 718/384-5913
1415	VEGA	FRANKLIN	NEW YORK	NY	M	FA2	28,279	-5043	15FEB76 718/384-2823
1076	VENTER	RANDALL	NEW YORK	NY	M	PT1	66,559	290	06OCT79 718/383-2321
1426	VICK	THERESA	PRINCETON	NJ	F	TA2	32,992	-1850	28JUN78 201/812-2424
1564	WALTERS	ANNE	NEW YORK	NY	F	SCP	18,834	-3548	04JUL80 212/587-3257
1221	WALTERS	DIANE	NEW YORK	NY	F	FA2	27,897	-1559	07OCT79 718/384-1918
1133	WANG	CHIN	NEW YORK	NY	M	TA1	27,702	-1995	15FEB80 212/587-1956
1435	WARD	ELAINE	NEW YORK	NY	F	TA3	38,809	-4614	11FEB68 718/383-4987
1418	WATSON	BERNARD	NEW YORK	NY	M	ME1	28,006	-5388	09JAN80 718/383-1298
1017	WELCH	DARIUS	NEW YORK	NY	M	TA3	40,859	-5114	19OCT69 212/586-5535
1443	WELLS	AGNES	STAMFORD	CT	F	NA1	42,275	-1137	01SEP79 203/781-5546
1131	WELLS	NADINE	NEW YORK	NY	F	TA2	32,576	-3	22APR79 718/383-1045
1427	WHALEY	CAROLYN	MT. VERNON	NY	F	TA2	34,047	-424	02FEB78 914/468-4528
1036	WONG	LESLIE	NEW YORK	NY	F	TA3	39,393	-2415	26OCT72 212/587-2570

```
1130 WOOD          DEBORAH        NEW YORK       NY F FA1   23,917     -227 08JUN80 212/587-0013
1127 WOOD          SANDRA         NEW YORK       NY F TA2   33,012    -2606 10DEC74 212/587-2881
1433 YANCEY        ROBIN          PRINCETON      NJ F FA3   32,983    -2000 20JAN75 201/812-1874
1431 YOUNG         DEBORAH        STAMFORD       CT F FA3   33,231    -2759 08APR76 203/781-2987
1122 YOUNG         JOANN          NEW YORK       NY F FA2   27,957    -3164 30NOV76 718/384-2021
1105 YOUNG         LAWRENCE       NEW YORK       NY M ME2   34,806    -3590 16AUG78 718/384-0008
;
run;
```

CONTROL.MLSCL

```
data control.mlscl (label='Multiple Sclerosis Data');
    input GROUP OBS1 OBS2 WT;
    datalines;
      1        4       4     10
      1        4       1      3
      1        4       2      7
      1        4       3      3
      1        1       4      1
      1        1       1     38
      1        1       2      5
      1        1       3      0
      1        2       4      0
      1        2       1     33
      1        2       2     11
      1        2       3      3
      1        3       4      6
      1        3       1     10
      1        3       2     14
      1        3       3      5
      2        4       1      1
      2        4       2      2
      2        4       3      4
      2        4       4     14
      2        3       1      2
      2        3       2     13
      2        3       3      3
      2        3       4      4
      2        2       1      3
      2        2       2     11
      2        2       3      4
      2        2       4      0
      2        1       1      5
      2        1       2      3
      2        1       3      0
      2        1       4      0
;
run;
```

CONTROL.NAMES

```
data control.names;
    input LABEL $ 1-16 START $ 17-24 FMTNAME $ 31-35 TYPE $ 41-41;
    datalines;
Capalleti, Jimmy    2355      bonus     C
Chen, Len           5889      bonus     C
Davis, Brad         3878      bonus     C
Leung, Brenda       4409      bonus     C
Patel, Mary         2398      bonus     C
Smith, Robert       5862      bonus     C
Zook, Carla         7385      bonus     C
;
run;
```

CONTROL.OXYGEN

```
data control.oxygen;
    input AGE WEIGHT RUNTIME RSTPULSE RUNPULSE MAXPULSE OXYGEN;
    datalines;
44     89.47     11.37        62          178         182      44.609
40     75.07     10.07        62          185         185      45.313
44     85.84      8.65        45          156         168      54.297
42     68.15      8.17        40          166         172      59.571
38     89.02      9.22        55          178         180      49.874
47     77.45     11.63        58          176         176      44.811
40     75.98     11.95        70          176         180      45.681
43     81.19     10.85        64          162         170      49.091
44     81.42     13.08        63          174         176      39.442
38     81.87      8.63        48          170         186      60.055
44     73.03     10.13        45          168         168      50.541
45     87.66     14.03        56          186         192      37.388
45     66.45     11.12        51          176         176      44.754
47     79.15     10.60        47          162         164      47.273
54     83.12     10.33        50          166         170      51.855
49     81.42      8.95        44          180         185      49.156
51     69.63     10.95        57          168         172      40.836
51     77.91     10.00        48          162         168      46.672
48     91.63     10.25        48          162         164      46.774
49     73.37     10.08        76          168         168      50.388
57     73.37     12.63        58          174         176      39.407
54     79.38     11.17        62          156         165      46.080
52     76.32      9.63        48          164         166      45.441
50     70.87      8.92        48          146         155      54.625
51     67.25     11.08        48          172         172      45.118
54     91.63     12.88        44          168         172      39.203
51     73.71     10.47        59          186         188      45.790
57     59.08      9.93        49          148         155      50.545
49     76.32      9.40        56          186         188      48.673
48     61.24     11.50        52          170         176      47.920
52     82.78     10.50        53          170         172      47.467
;
run;
```

CONTROL.PERSONL

```
data control.personl;
    input IDNUM $ 1-4 LNAME $ 5-19 FNAME $ 20-34 CITY $ 35-49 STATE $ 50-51
          SEX $ 53-53 JOBCODE $ 55-57 SALARY BIRTH date. @66
          HIRED date7. @74 HPHONE $ 84-95;
    format birth date7.;
    informat birth date.;
    format hired date7.;
    informat hired date.;
    datalines;
1919 ADAMS          GERALD         STAMFORD       CT M TA2  34376 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
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  25824 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 FA3  33104 23SEP48 03SEP72   203/675-4117
1639 CARTER         KAREN          STAMFORD       CT F TA3  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
1410 HARRIS        CHARLES        STAMFORD       CT M PT2  84685 06MAY55 10NOV74   203/781-0937
1439 HARRISON      FELICIA        BRIDGEPORT     CT F PT1  70736 09MAR52 13SEP78   203/675-4987
1409 HARTFORD      RAYMOND        STAMFORD       CT M ME3  41551 22APR38 25OCT69   203/781-9697
1408 HENDERSON     WILLIAM        PRINCETON      NJ M TA2  34138 01APR48 17OCT75   201/812-4789
1121 HERNANDEZ     MICHAEL        NEW YORK       NY M ME1  29112 29SEP59 10DEC79   718/384-3313
1991 HOWARD        GRETCHEN       BRIDGEPORT     CT F TA1  27645 10MAY60 15DEC80   203/675-0007
1102 HOWARD        LEONARD        WHITE PLAINS   NY M TA2  34542 04OCT47 18APR79   914/455-0976
1356 HOWARD        MICHAEL        NEW YORK       NY M ME2  36869 29SEP45 25FEB71   212/586-8411
1545 HUNTER        CLYDE          STAMFORD       CT M PT1  66130 15AUG47 01JUN78   203/781-1119
1292 HUNTER        HELEN          BRIDGEPORT     CT F ME2  36691 31OCT52 05JUL77   203/675-4830
1440 JACKSON       LAURA          STAMFORD       CT F ME2  35757 30SEP50 12APR79   203/781-0088
1368 JEPSEN        RONALD         STAMFORD       CT M FA2  27808 14JUN49 06NOV72   203/781-8413
1369 JOHNSON       ANTHONY        NEW YORK       NY M TA2  33705 31DEC49 16MAR75   212/587-5385
1411 JOHNSON       JACKSON        PATERSON       NJ M FA2  27265 30MAY49 04DEC77   201/732-3678
1113 JONES         LESLIE         NEW YORK       NY F FA1  22367 18JAN56 20OCT79   718/383-3003
1704 JONES         NATHAN         NEW YORK       NY M BCK  25465 02SEP54 01JUL75   718/384-0049
1900 KING          WILLIAM        NEW YORK       NY M ME2  35105 28MAY50 30OCT75   718/383-3698
1126 KIRBY         ANNE           NEW YORK       NY F TA3  40899 31MAY51 24NOV68   212/586-1229
1677 KRAMER        JACKSON        BRIDGEPORT     CT M BCK  26007 08NOV51 30MAR77   203/675-7432
1441 LAWRENCE      KATHY          PRINCETON      NJ F FA2  27158 22NOV57 26MAR79   201/812-3337
1421 LEE           RUSSELL        MT. VERNON     NY M TA2  33155 11JAN47 03MAR78   914/468-9143
1119 LI            JEFF           NEW YORK       NY M TA1  26924 23JUN50 09SEP76   212/586-2344
1834 LONG          RUSSELL        NEW YORK       NY M BCK  26896 11FEB60 05JUL80   718/384-0040
1777 LUFKIN        ROY            NEW YORK       NY M PT3 109630 26SEP39 24JUN69   718/383-4413
1663 MARKS         JOHN           NEW YORK       NY M BCK  26452 14JAN55 14AUG79   212/587-7742
1106 MARSHBURN     JASPER         STAMFORD       CT M PT2  89632 09NOV45 19AUG72   203/781-1457
1103 MCDANIEL      RONDA          NEW YORK       NY F FA1  23738 19FEB56 26JUL80   212/586-0013
1477 MEYERS        PRESTON        BRIDGEPORT     CT M FA2  28566 24MAR52 10MAR76   203/675-8125
1476 MONROE        JOYCE          STAMFORD       CT F TA2  34803 02JUN54 20MAR75   203/781-2837
1379 MORGAN        ALFRED         STAMFORD       CT M ME3  42264 11AUG49 13JUN72   203/781-2216
1104 MORGAN        CHRISTOPHER    NEW YORK       NY M SCP  17946 28APR51 13JUN79   718/383-9740
1009 MORGAN        GEORGE         NEW YORK       NY M TA1  28880 05MAR47 29MAR80   212/586-7753
1412 MURPHEY       JOHN           PRINCETON      NJ M ME1  27799 21JUN44 08DEC79   201/812-4414
1115 MURPHY        ALICE          NEW YORK       NY F FA3  32699 25AUG48 03MAR68   718/384-1982
1128 NELSON        FELICIA        BRIDGEPORT     CT F TA2  32777 26MAY53 23OCT78   203/675-1166
1442 NEWKIRK       SANDRA         PRINCETON      NJ F PT2  84536 08SEP54 15APR76   201/812-3331
1417 NEWKIRK       WILLIAM        PATERSON       NJ M NA2  52270 30JUN52 10MAR77   201/732-6611
1478 NEWTON        JAMES          NEW YORK       NY M PT2  84203 12AUG47 27OCT78   212/587-5549
1673 NICHOLLS      HENRY          STAMFORD       CT M BCK  25477 02MAR58 18JUL79   203/781-7770
1839 NORRIS        DIANE          NEW YORK       NY F NA1  44433 02DEC58 06JUL81   718/384-1767
1347 O'NEAL        BRYAN          NEW YORK       NY M TA3  40079 24SEP55 09SEP72   718/384-0230
1423 OSWALD        LESLIE         MT. VERNON     NY F ME2  35773 17MAY56 22AUG78   914/468-9171
1200 OVERMAN       MICHELLE       STAMFORD       CT F ME1  27816 13JAN59 17AUG80   203/781-1835
1970 PARKER        ANNE           NEW YORK       NY F FA1  22615 28SEP52 15MAR79   718/383-3895
1521 PARKER        JAY            NEW YORK       NY M ME3  41526 15APR51 16JUL76   212/587-7603
1354 PARKER        MARY           WHITE PLAINS   NY F SCP  18335 01JUN59 19JUN80   914/455-2337
1424 PATTERSON     RENEE          NEW YORK       NY F FA2  28978 07AUG57 14DEC77   212/587-8991
```

```
1132 PEARCE        CAROL          NEW YORK       NY F FA1  22413 02JUN60 25OCT81   718/384-1986
1845 PEARSON       JAMES          NEW YORK       NY M BCK  25996 23NOV47 25MAR68   718/384-2311
1556 PENNINGTON    MICHAEL        NEW YORK       NY M PT1  71349 25JUN52 14DEC79   718/383-5681
1413 PETERS        RANDALL        PRINCETON      NJ M FA2  27435 19SEP53 05JAN78   201/812-2478
1907 PHELPS        WILLIAM        STAMFORD       CT M TA2  33329 18NOV48 09JUL75   203/781-1118
1436 PORTER        SUSAN          NEW YORK       NY F TA2  34475 14JUN52 15MAR75   718/383-5777
1385 RAYNOR        MILTON         BRIDGEPORT     CT M ME3  43900 19JAN50 04APR74   203/675-2846
1432 REED          MARILYN        MT. VERNON     NY F ME2  35327 06NOV49 13FEB73   914/468-5454
1111 RHODES        JEREMY         PRINCETON      NJ M NA1  40586 17JUL61 03NOV80   201/812-1837
1116 RICHARDS      CASEY          NEW YORK       NY F FA1  22862 01OCT57 24MAR79   212/587-1224
1352 RIVERS        SIMON          NEW YORK       NY M NA2  53798 05DEC48 19OCT74   718/383-3345
1555 RODRIGUEZ     JULIA          BRIDGEPORT     CT F FA2  27499 19MAR56 07JUL80   203/675-2401
1038 RODRIGUEZ     MARIA          BRIDGEPORT     CT F TA1  26533 12NOV57 26NOV79   203/675-2048
1420 ROUSE         JEREMY         PATERSON       NJ M ME3  43071 22FEB53 25JUL75   201/732-9834
1561 SANDERS       RAYMOND        NEW YORK       NY M TA2  34514 03DEC51 10OCT75   212/588-6615
1434 SANDERSON     EDITH          STAMFORD       CT F FA2  28622 14JUL50 31OCT78   203/781-1333
1414 SANDERSON     NATHAN         BRIDGEPORT     CT M FA1  23644 27MAR60 15APR80   203/675-1715
1112 SAYERS        RANDY          NEW YORK       NY M TA1  26905 02DEC52 10DEC80   718/384-4895
1390 SMART         JONATHAN       NEW YORK       NY M FA2  27761 22FEB53 26JUN79   718/383-1141
1332 STEPHENSON    ADAM           BRIDGEPORT     CT M NA1  42178 20SEP58 07JUN79   203/675-1497
1890 STEPHENSON    ROBERT         NEW YORK       NY M PT2  85896 23JUL39 28NOV67   718/384-9874
1429 THOMPSON      ALICE          STAMFORD       CT F TA1  27939 02MAR48 10AUG80   203/781-3857
1107 THOMPSON      WAYNE          NEW YORK       NY M PT2  89977 12JUN42 13FEB67   718/384-3785
1908 TRENTON       MELISSA        NEW YORK       NY F TA2  32995 13DEC57 26APR78   212/586-6262
1830 TRIPP         KATHY          BRIDGEPORT     CT F PT2  84471 30MAY45 01FEB71   203/675-2479
1882 TUCKER        ALAN           NEW YORK       NY M ME3  41538 13JUL45 24NOV66   718/384-0216
1050 TUTTLE        THOMAS         WHITE PLAINS   NY M ME2  35167 17JUL51 27AUG74   914/455-2119
1425 UNDERWOOD     JENNY          STAMFORD       CT F FA1  23979 31DEC59 03MAR81   203/781-0978
1928 UPCHURCH      LARRY          WHITE PLAINS   NY M PT2  89858 19SEP42 16JUL78   914/455-5009
1480 UPDIKE        THERESA        NEW YORK       NY F TA3  39583 06SEP45 28MAR69   212/587-8729
1100 VANDEUSEN     RICHARD        NEW YORK       NY M BCK  25004 04DEC48 10MAY76   212/586-2531
1995 VARNER        ELIZABETH      NEW YORK       NY F ME1  28810 27AUG61 22SEP81   718/384-7113
1135 VEGA          ANNA           NEW YORK       NY F FA2  27321 23SEP48 03APR78   718/384-5913
1415 VEGA          FRANKLIN       NEW YORK       NY M FA2  28278 12MAR46 15FEB76   718/384-2823
1076 VENTER        RANDALL        NEW YORK       NY M PT1  66558 17OCT60 06OCT79   718/383-2321
1426 VICK          THERESA        PRINCETON      NJ F TA2  32991 08DEC54 28JUN78   201/812-2424
1564 WALTERS       ANNE           NEW YORK       NY F SCP  18833 15APR50 04JUL80   212/587-3257
1221 WALTERS       DIANE          NEW YORK       NY F FA2  27896 25SEP55 07OCT79   718/384-1918
1133 WANG          CHIN           NEW YORK       NY M TA1  27701 16JUL54 15FEB80   212/587-1956
1435 WARD          ELAINE         NEW YORK       NY F TA3  38808 15MAY47 11FEB68   718/383-4987
1418 WATSON        BERNARD        NEW YORK       NY M ME1  28005 01APR45 09JAN80   718/383-1298
1017 WELCH         DARIUS         NEW YORK       NY M TA3  40858 31DEC45 19OCT69   212/586-5535
1443 WELLS         AGNES          STAMFORD       CT F NA1  42274 20NOV56 01SEP79   203/781-5546
1131 WELLS         NADINE         NEW YORK       NY F TA2  32575 29DEC59 22APR79   718/383-1045
1427 WHALEY        CAROLYN        MT. VERNON     NY F TA2  34046 03NOV58 02FEB78   914/468-4528
1036 WONG          LESLIE         NEW YORK       NY F TA3  39392 22MAY53 26OCT72   212/587-2570
1130 WOOD          DEBORAH        NEW YORK       NY F FA1  23916 19MAY59 08JUN80   212/587-0013
1127 WOOD          SANDRA         NEW YORK       NY F TA2  33011 12NOV52 10DEC74   212/587-2881
1433 YANCEY        ROBIN          PRINCETON      NJ F FA3  32982 11JUL54 20JAN75   201/812-1874
1431 YOUNG         DEBORAH        STAMFORD       CT F FA3  33230 12JUN52 08APR76   203/781-2987
1122 YOUNG         JOANN          NEW YORK       NY F FA2  27956 04MAY51 30NOV76   718/384-2021
1105 YOUNG         LAWRENCE       NEW YORK       NY M ME2  34805 04MAR50 16AUG78   718/384-0008
;
run;
```

CONTROL.PHARM

```
data control.pharm (label='Sugar Study');
    input DRUG $8. RESPONSE $8. WT ;
    datalines;
      A cured   14
      A uncured 22
      B cured   24
      B uncured 19
      C cured   17
      C uncured 13
;
run;
```

CONTROL.POINTS

```
data control.points;
    input EMPID $8. Q1 Q2 Q3 Q4 TOTPTS;
    datalines;
2355            3       4       4       3       14
5889            2       2       2       2       8
3878            1       2       2       2       7
4409            0       1       1       1       3
2398            2       2       1       1       6
5862            1       1       1       2       5
;
run;
```

CONTROL.PRENAT

```
data control.prenat;
    input IDNUM $ 1-4 LNAME $ 6-20  FNAME $ 22-36 CITY $ 39-53
          STATE $ 55-56 HPHONE $ 58-69;
    datalines;
1919 ADAMS           GERALD          STAMFORD        CT 203/781-1255
1653 ALIBRANDI       MARIA           BRIDGEPORT      CT 203/675-7715
1400 ALHERTANI       ABDULLAH        NEW YORK        NY 212/586-0808
1350 ALVAREZ         MERCEDES        NEW YORK        NY 718/383-1549
1401 ALVAREZ         CARLOS          PATERSON        NJ 201/732-8787
1499 BAREFOOT        JOSEPH          PRINCETON       NJ 201/812-5665
1101 BAUCOM          WALTER          NEW YORK        NY 212/586-8060
1333 BANADYGA        JUSTIN          STAMFORD        CT 203/781-1777
1402 BLALOCK         RALPH           NEW YORK        NY 718/384-2849
1479 BALLETTI        MARIE           NEW YORK        NY 718/384-8816
1403 BOWDEN          EARL            BRIDGEPORT      CT 203/675-3434
1739 BRANCACCIO      JOSEPH          NEW YORK        NY 212/587-1247
1658 BREUHAUS        JEREMY          NEW YORK        NY 212/587-3622
1428 BRADY           CHRISTINE       STAMFORD        CT 203/781-1212
1782 BREWCZAK        JAKOB           STAMFORD        CT 203/781-0019
1244 BUCCI           ANTHONY         NEW YORK        NY 718/383-3334
1383 BURNETTE        THOMAS          NEW YORK        NY 718/384-3569
1574 CAHILL          MARSHALL        NEW YORK        NY 718/383-2338
```

```
1789 CARAWAY         DAVIS           NEW YORK        NY 212/587-9000
1404 COHEN           LEE             NEW YORK        NY 718/384-2946
1437 CARTER          DOROTHY         BRIDGEPORT      CT 203/675-4117
1639 CARTER-COHEN    KAREN           STAMFORD        CT 203/781-8839
1269 CASTON          FRANKLIN        STAMFORD        CT 203/781-3335
1065 COPAS           FREDERICO       NEW YORK        NY 718/384-5618
1876 CHIN            JACK            NEW YORK        NY 212/588-5634
1037 CHOW            JANE            STAMFORD        CT 203/781-8868
1129 COUNIHAN        BRENDA          NEW YORK        NY 718/383-2313
1988 COOPER          ANTHONY         NEW YORK        NY 212/587-1228
1405 DACKO           JASON           PATERSON        NJ 201/732-2323
1430 DABROWSKI       SANDRA          BRIDGEPORT      CT 203/675-1647
1983 DEAN            SHARON          NEW YORK        NY 718/384-1647
1134 DELGADO         MARIA           STAMFORD        CT 203/781-1528
1118 DENNIS          ROGER           NEW YORK        NY 718/383-1122
1438 DABBOUSSI       KAMILLA         STAMFORD        CT 203/781-2229
1125 DUNLAP          DONNA           NEW YORK        NY 718/383-2094
1475 ELGES           MARGARETE       NEW YORK        NY 718/383-2828
1117 EDGERTON        JOSHUA          NEW YORK        NY 212/588-1239
1935 FERNANDEZ       KATRINA         BRIDGEPORT      CT 203/675-2962
1124 FIELDS          DIANA           WHITE PLAINS    NY 914/455-2998
1422 FUJIHARA        KYOKO           PRINCETON       NJ 201/812-0902
1616 FUENTAS         CARLA           NEW YORK        NY 718/384-3329
1406 FOSTER          GERALD          BRIDGEPORT      CT 203/675-6363
1120 GARCIA          JACK            NEW YORK        NY 718/384-4930
1094 GOMEZ           ALAN            BRIDGEPORT      CT 203/675-7181
1389 GOLDSTEIN       LEVI            NEW YORK        NY 718/384-9326
1905 GRAHAM          ALVIN           NEW YORK        NY 212/586-8815
1407 GREGORSKI       DANIEL          MT. VERNON      NY 914/468-1616
1114 GREENWALD       JANICE          NEW YORK        NY 212/588-1092
1410 HARRIS          CHARLES         STAMFORD        CT 203/781-0937
1439 HASENHAUER      CHRISTINA       BRIDGEPORT      CT 203/675-4987
1409 HAVELKA         RAYMOND         STAMFORD        CT 203/781-9697
1408 HENDERSON       WILLIAM         PRINCETON       NJ 201/812-4789
1121 HERNANDEZ       ROBERTO         NEW YORK        NY 718/384-3313
1991 HOWARD          GRETCHEN        BRIDGEPORT      CT 203/675-0007
1102 HERMANN         JOACHIM         WHITE PLAINS    NY 914/455-0976
1356 HOWARD          MICHAEL         NEW YORK        NY 212/586-8411
1545 HERRERO         CLYDE           STAMFORD        CT 203/781-1119
1292 HUNTER          HELEN           BRIDGEPORT      CT 203/675-4830
1440 JACKSON         LAURA           STAMFORD        CT 203/781-0088
1368 JEPSEN          RONALD          STAMFORD        CT 203/781-8413
1369 JONSON          ANTHONY         NEW YORK        NY 212/587-5385
1411 JOHNSEN         JACK            PATERSON        NJ 201/732-3678
1113 JOHNSON         LESLIE          NEW YORK        NY 718/383-3003
1704 JONES           NATHAN          NEW YORK        NY 718/384-0049
1900 KING            WILLIAM         NEW YORK        NY 718/383-3698
1126 KIMANI          ANNE            NEW YORK        NY 212/586-1229
1677 KRAMER          JACKSON         BRIDGEPORT      CT 203/675-7432
1441 LAWRENCE        KATHY           PRINCETON       NJ 201/812-3337
1421 LEE             RUSSELL         MT. VERNON      NY 914/468-9143
1119 LI              JEFF            NEW YORK        NY 212/586-2344
1834 LEBLANC         RUSSELL         NEW YORK        NY 718/384-0040
1777 LUFKIN          ROY             NEW YORK        NY 718/383-4413
```

1663	MARKS	JOHN	NEW YORK	NY 212/587-7742
1106	MARSHBURN	JASPER	STAMFORD	CT 203/781-1457
1103	MCDANIEL	RONDA	NEW YORK	NY 212/586-0013
1477	MEYERS	PRESTON	BRIDGEPORT	CT 203/675-8125
1476	MONROE	JOYCE	STAMFORD	CT 203/781-2837
1379	MORGAN	ALFRED	STAMFORD	CT 203/781-2216
1104	MORGAN	CHRISTOPHER	NEW YORK	NY 718/383-9740
1009	MORGAN	GEORGE	NEW YORK	NY 212/586-7753
1412	MURPHEY	JOHN	PRINCETON	NJ 201/812-4414
1115	MURPHY	ALICE	NEW YORK	NY 718/384-1982
1128	NELSON	FELICIA	BRIDGEPORT	CT 203/675-1166
1442	NEWKIRK	SANDRA	PRINCETON	NJ 201/812-3331
1417	NEWKIRK	WILLIAM	PATERSON	NJ 201/732-6611
1478	NEWTON	JAMES	NEW YORK	NY 212/587-5549
1673	NICHOLLS	HENRY	STAMFORD	CT 203/781-7770
1839	NORRIS	DIANE	NEW YORK	NY 718/384-1767
1347	O'NEAL	BRYAN	NEW YORK	NY 718/384-0230
1423	OSWALD	LESLIE	MT. VERNON	NY 914/468-9171
1200	OVERMAN	MICHELLE	STAMFORD	CT 203/781-1835
1970	PARKER	ANNE	NEW YORK	NY 718/383-3895
1521	PARKER	JAY	NEW YORK	NY 212/587-7603
1354	PARKER	MARY	WHITE PLAINS	NY 914/455-2337
1424	PATTERSON	RENEE	NEW YORK	NY 212/587-8991
1132	PEARCE	CAROL	NEW YORK	NY 718/384-1986
1845	PEARSON	JAMES	NEW YORK	NY 718/384-2311
1556	PENNINGTON	MICHAEL	NEW YORK	NY 718/383-5681
1413	PETERS	RANDALL	PRINCETON	NJ 201/812-2478
1123	PETERSON	SUZANNE	NEW YORK	NY 718/383-0077
1907	PHELPS	WILLIAM	STAMFORD	CT 203/781-1118
1436	PORTER	SUSAN	NEW YORK	NY 718/383-5777
1385	RAYNOR	MILTON	BRIDGEPORT	CT 203/675-2846
1432	REED	MARILYN	MT. VERNON	NY 914/468-5454
1111	RHODES	JEREMY	PRINCETON	NJ 201/812-1837
1116	RICHARDS	CASEY	NEW YORK	NY 212/587-1224
1352	RIVERS	SIMON	NEW YORK	NY 718/383-3345
1555	RODRIGUEZ	JULIA	BRIDGEPORT	CT 203/675-2401
1038	RODRIGUEZ	MARIA	BRIDGEPORT	CT 203/675-2048
1420	ROUSE	JEREMY	PATERSON	NJ 201/732-9834
1561	SANDERS	RAYMOND	NEW YORK	NY 212/588-6615
1434	SANDERSON	EDITH	STAMFORD	CT 203/781-1333
1414	SANDERSON	NATHAN	BRIDGEPORT	CT 203/675-1715
1112	SANYERS	RANDY	NEW YORK	NY 718/384-4895
1390	SMART	JONATHAN	NEW YORK	NY 718/383-1141
1332	STEPHENSON	ADAM	BRIDGEPORT	CT 203/675-1497
1890	STEPHENSON	ROBERT	NEW YORK	NY 718/384-9874
1429	THOMPSON	ALICE	STAMFORD	CT 203/781-3857
1107	THOMPSON	WAYNE	NEW YORK	NY 718/384-3785
1908	TRENTON	MELISSA	NEW YORK	NY 212/586-6262
1830	TRIPP	KATHY	BRIDGEPORT	CT 203/675-2479
1882	TUCKER	ALAN	NEW YORK	NY 718/384-0216
1050	TUTTLE	THOMAS	WHITE PLAINS	NY 914/455-2119
1425	UNDERWOOD	JENNY	STAMFORD	CT 203/781-0978
1928	UPCHURCH	LARRY	WHITE PLAINS	NY 914/455-5009
1480	UPDIKE	THERESA	NEW YORK	NY 212/587-8729

```
1100 VANDEUSEN       RICHARD          NEW YORK        NY 212/586-2531
1995 VARNER          ELIZABETH        NEW YORK        NY 718/384-7113
1135 VEGA            ANNA             NEW YORK        NY 718/384-5913
1415 VEGA            FRANKLIN         NEW YORK        NY 718/384-2823
1076 VENTER          RANDALL          NEW YORK        NY 718/383-2321
1426 VICK            THERESA          PRINCETON       NJ 201/812-2424
1564 WALTERS         ANNE             NEW YORK        NY 212/587-3257
1221 WALTERS         DIANE            NEW YORK        NY 718/384-1918
1133 WANG            CHIN             NEW YORK        NY 212/587-1956
1435 WARD            ELAINE           NEW YORK        NY 718/383-4987
1418 WATSON          BERNARD          NEW YORK        NY 718/383-1298
1017 WELCH           DARIUS           NEW YORK        NY 212/586-5535
1443 WELLS           AGNES            STAMFORD        CT 203/781-5546
1131 WELLS           NADINE           NEW YORK        NY 718/383-1045
1427 WHALEY          CAROLYN          MT. VERNON      NY 914/468-4528
1036 WONG            LESLIE           NEW YORK        NY 212/587-2570
1130 WOOD            DEBORAH          NEW YORK        NY 212/587-0013
1127 WOOD            SANDRA           NEW YORK        NY 212/587-2881
1433 YANCEY          ROBIN            PRINCETON       NJ 201/812-1874
1431 YOUNG           DEBORAH          STAMFORD        CT 203/781-2987
1122 YOUNG           JOANN            NEW YORK        NY 718/384-2021
1105 YOUNG           LAWRENCE         NEW YORK        NY 718/384-0008

;
run;
```

CONTROL.RESULTS

```
data control.results;
    input ID    TREAT $8.   INITWT    WT3MOS    AGE;
    datalines;
       1   Other  166.28  146.98     35
       2   Other  214.42  210.22     54
       3   Other  172.46  159.42     33
       5   Other  175.41  160.66     37
       6   Other  173.13  169.40     20
       7   Other  181.25  170.94     30
      10   Other  239.83  214.48     48
      11   Other  175.32  162.66     51
      12   Other  227.01  211.06     29
      13   Other  274.82  251.82     31
;
run;
```

CONTROL.SLEEP

```
data control.sleep;
    input  GROUP TIME  SOL  WASO FNA TST;
    datalines;
    1.00    1.00   38.69   48.43     0      0
    2.36  424.50    0.00    0.00     0      0
    1.00    2.00   15.83    9.67     0      0
```

```
2.16  500.30    0.00    0.00     0       0
1.00    1.00   93.04   87.10     0       0
1.86  302.40    0.00    0.00     0       0
1.00    2.00   65.00   16.67     0       0
1.50  305.00    0.00    0.00     0       0
1.00    1.00   19.82   74.38     0       0
2.00  359.20    0.00    0.00     0       0
1.00    2.00   13.75   13.90     0       0
1.40  378.13    0.00    0.00     0       0
1.00    1.00   47.35   60.52     0       0
2.89  248.10    0.00    0.00     0       0
1.00    2.00   72.14   86.07     0       0
4.14  308.50    0.00    0.00     0       0
1.00    1.00   99.65  151.79     0       0
3.86  263.93    0.00    0.00     0       0
1.00    2.00   65.35  118.33     0       0
2.71  374.17    0.00    0.00     0       0
1.00    1.00   47.15  105.36     0       0
2.50  254.60    0.00    0.00     0       0
1.00    2.00   66.00   92.60     0       0
2.20  297.40    0.00    0.00     0       0
1.00    1.00   30.84   84.97     0       0
3.24  242.90    0.00    0.00     0       0
1.00    2.00    7.86   23.86     0       0
1.28  340.70    0.00    0.00     0       0
1.00    1.00  117.41   36.93     0       0
1.00  204.20    0.00    0.00     0       0
1.00    2.00   50.42   24.86     0       0
1.85  231.00    0.00    0.00     0       0
1.00    1.00    6.50   41.15     0       0
2.67  308.00    0.00    0.00     0       0
1.00    2.00    2.00    0.00     0       0
0.00  454.40    0.00    0.00     0       0
2.00    1.00   54.29   84.93     0       0
3.43  394.70    0.00    0.00     0       0
2.00    2.00   13.57   48.00     0       0
1.00  405.00    0.00    0.00     0       0
2.00    1.00    4.43   58.43     0       0
2.89  375.70    0.00    0.00     0       0
2.00    2.00    5.00   49.28     0       0
3.57  423.60    0.00    0.00     0       0
2.00    1.00   32.50   38.43     0       0
4.57  357.20    0.00    0.00     0       0
2.00    2.00   17.14   33.14     0       0
3.57  315.70    0.00    0.00     0       0
2.00    1.00   33.57   83.43     0       0
4.36  282.50    0.00    0.00     0       0
2.00    2.00   22.85   37.86     0       0
2.42  288.57    0.00    0.00     0       0
2.00    1.00   53.21  120.00     0       0
3.50  366.80    0.00    0.00     0       0
2.00    2.00   29.00   97.40     0       0
2.80  388.00    0.00    0.00     0       0
2.00    1.00   56.79   67.73     0       0
```

```
    3.19  326.00    0.00    0.00     0       0
    2.00    2.00   52.50   64.16     0       0
    4.00   45.80    0.00    0.00     0       0
    2.00    1.00   23.50   35.19     0       0
    6.60  416.80    0.00    0.00     0       0
    2.00    2.00   25.71   38.43     0       0
    8.50  446.30    0.00    0.00     0       0
    2.00    1.00   43.00   95.59     0       0
    1.75  288.90    0.00    0.00     0       0
    2.00    2.00   45.00   85.71     0       0
    1.43  272.10    0.00    0.00     0       0
    3.00    1.00   24.70   67.17     0       0
    3.90  399.90    0.00    0.00     0       0
    3.00    2.00    3.86   22.50     0       0
    3.00  425.14    0.00    0.00     0       0
    3.00    1.00   70.72   80.43     0       0
    3.00  286.30    0.00    0.00     0       0
    3.00    2.00   22.50  139.33     0       0
    3.50  283.50    0.00    0.00     0       0
    3.00    1.00   48.23   40.57     0       0
    1.65  407.20    0.00    0.00     0       0
    3.00    2.00   30.00   38.00     0       0
    1.57  372.90    0.00    0.00     0       0
    3.00    1.00   43.93   34.57     0       0
    1.29  393.40    0.00    0.00     0       0
    3.00    2.00   30.71   58.43     0       0
    2.28  348.60    0.00    0.00     0       0
    3.00    1.00   95.00   35.22     0       0
    2.06  409.90    0.00    0.00     0       0
    3.00    2.00   66.43   68.57     0       0
    2.14  345.60    0.00    0.00     0       0
    3.00    1.00   18.93   86.43     0       0
    5.36  349.50    0.00    0.00     0       0
    3.00    2.00   17.50  116.50     0       0
    5.00  337.70    0.00    0.00     0       0
    3.00    1.00  125.00   69.15     0       0
    2.00  242.50    0.00    0.00     0       0
    3.00    2.00   60.00   81.43     0       0
    1.57  304.30    0.00    0.00     0       0
    3.00    1.00   38.57   68.61     0       0
    3.64  394.50    0.00    0.00     0       0
    3.00    2.00   18.33   19.83     0       0
    2.00  448.83    0.00    0.00     0       0
    3.00    1.00   43.00  121.72     0       0
    2.63  247.80    0.00    0.00     0       0
    3.00    2.00   40.00   98.83     0       0
    1.86  199.67    0.00    0.00     0       0
    3.00    1.00   11.61   70.36     0       0
    2.86  348.40    0.00    0.00     0       0
    3.00    2.00   20.43   70.57     0       0
    3.14  335.40    0.00    0.00     0       0
 ;
run;
```

CONTROL.SYNDROME

```
data control.syndrome;
    input FLIGHT $ 1-3 @10 DATE DATE7.  @22 DEPART TIME5. ORIG $ 31-33
          DEST $ 39-41 MILES  BOARDED  CAPACITY;
    format date DATE7.;
    format depart TIME5.;
    informat date DATE7.;
    informat depart TIME5.;
    datalines;
114      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
;
run;
```

CONTROL.TENSION

```
data control.tension;
    input TENSION $8.  CHD $8.  COUNT ;
    datalines;
yes      yes       97
yes      no       307
no       yes      200
no       no      1409
;
run;
```

CONTROL.TEST2

```
data control.test2;
    input STD1 $ TEST1 $  STD2 $   TEST2 $    WT;
    datalines ;
neg      neg     neg      neg     509
neg      neg     neg      pos       4
neg      neg     pos      neg      17
neg      neg     pos      pos       3
neg      pos     neg      neg      13
neg      pos     neg      pos       8
neg      pos     pos      pos       8
pos      neg     neg      neg      14
pos      neg     neg      pos       1
pos      neg     pos      neg      17
pos      neg     pos      pos       9
pos      pos     neg      neg       7
pos      pos     neg      pos       4
pos      pos     pos      neg       9
pos      pos     pos      pos     170
;
run;
```

CONTROL.TRAIN

```
data control.train;
    input NAME $ 1-16 IDNUM $ 17-24;
    datalines;
Capalleti, Jimmy    2355
Chen, Len           5889
Davis, Brad         3878
```

```
Leung, Brenda       4409
Patel, Mary         2398
Smith, Robert       5862
Zook, Carla         7385
;
run;
```

CONTROL.VISION

```
data control.vision;
    input RIGHT LEFT COUNT;
    datalines;
      1        1    1520
      1        2     266
      1        3     124
      1        4      66
      2        1     234
      2        2    1512
      2        3     432
      2        4      78
      3        1     117
      3        2     362
      3        3    1772
      3        4     205
      4        1      36
      4        2      82
      4        3     179
      4        4     492
;
run;
```

CONTROL.WEIGHT

```
data control.weight (label='California Results');
    input ID TREAT $8.  IBW INITWT WT3MOS WT6MOS WT9MOS AGE MMPI1 MMPI2
          MMPI3 MMPI4 MMPI5;
    datalines;
  1  Other    149  166.28  146.98  138.26     .       35    62     68     67     55     67
  2  Other    137  214.42  210.22     .    213.87     54    57     56     59     57     47
  3  Other    138  172.46  159.42  146.01  143.84     33    54     69     63     87     34
  5  Other    122  175.41  160.66  154.30     .       37    56     67     64     71     32
  6  Other    134  173.13  169.40  176.12     .       20    42     51     63     71     45
  7  Other    160  181.25  170.94     .       .       30    72     58     70     71     80
  9  Other    152  212.83  179.93  169.74  164.47  49100    65     87     71     74      4
 10  Other    145  239.83  214.48  208.28     .       48    56     51     56     53     53
 11  Other    158  175.32  162.66  161.39     .       51    66     60     71     62     84
 12  Other    174  227.01  211.06  202.87  205.17     29    77     70     69     65     64
 13  Other    137  274.82  251.82  248.18     .       31    66     82     66     69     55
 15  Other    152  168.75  156.58  154.61  156.58     42    52     58     65     67     51
 16  Other    162  187.81  172.07     .       .       40    57     68     67     67     74
 17  Other    123  226.63     .    219.72     .       21    58     49     59     74     70
 18  Other    146  176.03  160.27  160.27     .       41    48     55     49     68     43
 19  Other    166  190.96  159.04     .       .       32    53     52     51     62     71
```

21	Other	148	165.54	.	166.22	.	48	57	60	65	74	55
22	Other	125	193.60	184.00	.	.	28	64	67	70	69	54
24	Other	152	267.43	230.26	206.09	.	30	48	45	55	50	41
25	Other	151	193.38	185.43	.	.	33	54	99	67	75	74
26	Other	134	252.61	227.61	217.72	223.88	31	62	51	70	57	39
27	Other	140	193.93	191.43	196.43	.	25	54	65	66	55	55
29	Other	119	182.77	.	.	.	38	66	63	64	60	41
30	Other	134	189.93	172.39	175.37	.	35	70	92	73	98	39
31	Other	138	190.22	181.88	178.99	181.16	36	56	69	61	62	34
32	Other	134	182.09	169.40	163.81	163.81	34	42	65	54	55	41
34	Other	133	200.00	189.47	.	.	24	52	61	64	62	39
35	Other	149	221.81	216.11	.	.	46	66	52	69	71	61
36	Other	133	241.35	247.37	.	.	34	50	65	57	74	47
37	Other	134	223.13	217.91	.	.	33	64	63	63	60	41
38	Other	140	235.36	228.57	210.71	.	50	56	61	70	74	39
39	Other	125	178.60	178.40	.	.	39	50	73	58	58	32
40	Other	143	243.01	226.57	210.49	.	38	64	66	70	54	46
41	Other	134	282.65	239.55	.	.	26	66	65	75	74	53
44	Other	139	282.37	258.99	238.13	241.01	43	66	69	68	65	39
45	Other	134	216.04	182.09	.	.	39	50	55	59	67	43
46	Other	115	190.00	171.30	.	.	36	64	59	58	58	44
47	Other	134	175.19	167.16	.	.	43	57	48	52	71	47
48	Other	118	179.87	.	.	.	37	52	57	45	50	30
50	Other	137	173.54	166.97	164.60	.	32	54	76	63	69	25
51	Other	125	180.60	162.40	152.00	157.60	32	50	56	64	53	53
52	Other	155	235.32	225.16	210.32	208.39	35	62	64	55	60	51
53	Other	143	183.39	169.23	.	.	38	64	57	72	81	61
54	Other	131	212.60	208.40	211.45	.	27	50	67	53	74	47
63	Surgery	146	219.18	167.12	139.73	119.18	35	58	53	67	48	55
65	Surgery	123	192.68	155.28	127.64	115.45	31	52	74	54	64	32
66	Surgery	134	199.25	173.88	161.19	144.03	38	68	64	70	77	30
71	Surgery	139	209.35	172.66	156.83	138.13	39	66	56	66	71	42
77	Surgery	137	179.56	150.36	132.12	123.36	29	46	49	42	47	49
80	Surgery	170	138.24	121.76	100.00	.	31	56	57	64	64	39
81	Surgery	129	206.98	173.64	132.56	121.71	28	42	78	52	62	41
84	Surgery	143	220.28	178.32	.	.	29	58	67	54	46	59
85	Surgery	152	189.47	156.58	140.79	140.79	34	75	73	62	74	51
86	Surgery	210	157.14	138.57	121.90	109.05	30	88	75	73	69	65
92	Surgery	139	203.60	169.78	143.88	.	38	62	59	68	60	49
93	Surgery	151	171.52	150.33	123.18	109.27	42	72	69	59	53	47
95	Surgery	134	207.46	155.22	.	.	41	51	52	56	63	57
96	Surgery	138	201.45	172.46	172.46	155.07	55	57	49	57	67	37
97	Surgery	128	209.38	182.81	162.50	153.91	34	68	88	73	74	.
101	Surgery	166	185.54	146.39	139.76	132.53	42	82	80	89	79	51
102	Surgery	127	218.11	173.23	152.76	137.80	39	76	80	70	74	45
107	Surgery	128	227.34	192.97	184.38	170.31	49	50	51	59	50	55
110	Surgery	143	251.75	207.69	183.22	165.03	42	48	84	52	76	38
111	Surgery	152	197.37	164.47	148.68	132.89	56	90	70	86	76	70
112	Surgery	140	202.14	156.43	137.86	.	31	48	51	50	41	41
116	Surgery	139	273.38	235.25	194.24	.	31	58	55	66	81	45
117	Surgery	125	192.00	156.80	140.00	127.20	42	58	44	63	53	51
120	Surgery	119	277.31	231.93	208.40	192.44	31	60	69	59	95	34
122	Surgery	138	165.22	130.43	119.57	.	44	66	67	70	62	34
125	Surgery	152	257.89	200.00	182.89	134.21	39	47	84	53	88	74

```
126  Surgery  138  170.29  142.75     .       .       39    64     64     66     65     46
132  Surgery  149  208.05  173.83     .    126.85     34    82     57     75     65     45
133  Surgery  136  230.15  205.15  190.44  180.88     39    52     71     52     65     30
134  Surgery  168  177.98  140.48  119.64     .       45    70     77     62     67     69
137  Surgery  138  188.41  164.49  152.90  135.51     39    58     61     57     60     46
138  Surgery  141  268.09  220.57  185.82     .       32    52     59     66     74     53
158  Surgery  130  220.00  190.77  173.85     .       27    68     73     66     58     43
223  Surgery  157  220.38  185.35  152.23  138.85     43    78     76     70     57     53
266  Surgery  135  184.44  148.15  123.70     .       41    56     55     59     64     49
269  Surgery  155  201.29  151.61  140.65     .       57    82     84     86     77     73
293  Surgery  160  163.75  130.63     .       .       47    52     67     47     64     76
295  Surgery  128  211.72  172.66  151.56  137.50     45    93     65     80     60     41
298  Surgery  140  243.57  195.00     .    166.43     40    63     62     62     76     52
;
run;
```

CONTROL.WGHT

```
data control.wght (label='California Results');
    input ID TREAT $8.  IBW INITWT WT3MOS WT6MOS WT9MOS AGE MMPI1 MMPI2
          MMPI3 MMPI4 MMPI5;
    datalines;
  1  Other    149  166.28  146.98  138.26     .       35    62     68     67     55     67
  2  Other    137  214.42  210.22     .    213.87     54    57     56     59     57     47
  3  Other    138  172.46  159.42  146.01  143.84     33    54     69     63     87     34
  5  Other    122  175.41  160.66  154.30     .       37    56     67     64     71     32
  6  Other    134  173.13  169.40  176.12     .       20    42     51     63     71     45
  7  Other    160  181.25  170.94     .       .       30    72     58     70     71     80
  9  Other    152  212.83  179.93  169.74  164.47  49100    65     87     71     74      4
 10  Other    145  239.83  214.48  208.28     .       48    56     51     56     53     53
 11  Other    158  175.32  162.66  161.39     .       51    66     60     71     62     84
 12  Other    174  227.01  211.06  202.87  205.17     29    77     70     69     65     64
 13  Other    137  274.82  251.82  248.18     .       31    66     82     66     69     55
 15  Other    152  168.75  156.58  154.61  156.58     42    52     58     65     67     51
 16  Other    162  187.81  172.07     .       .       40    57     68     67     67     74
 17  Other    123  226.63     .    219.72     .       21    58     49     59     74     70
 18  Other    146  176.03  160.27  160.27     .       41    48     55     49     68     43
 19  Other    166  190.96  159.04     .       .       32    53     52     51     62     71
 21  Other    148  165.54     .    166.22     .       48    57     60     65     74     55
 22  Other    125  193.60  184.00     .       .       28    64     67     70     69     54
 24  Other    152  267.43  230.26  206.09     .       30    48     45     55     50     41
 25  Other    151  193.38  185.43     .       .       33    54     99     67     75     74
 26  Other    134  252.61  227.61  217.72  223.88     31    62     51     70     57     39
 27  Other    140  193.93  191.43  196.43     .       25    54     65     66     55     55
 29  Other    119  182.77     .       .       .       38    66     63     64     60     41
 30  Other    134  189.93  172.39  175.37     .       35    70     92     73     98     39
 31  Other    138  190.22  181.88  178.99  181.16     36    56     69     61     62     34
 32  Other    134  182.09  169.40  163.81  163.81     34    42     65     54     55     41
 34  Other    133  200.00  189.47     .       .       24    52     61     64     62     39
 35  Other    149  221.81  216.11     .       .       46    66     52     69     71     61
 36  Other    133  241.35  247.37     .       .       34    50     65     57     74     47
 37  Other    134  223.13  217.91     .       .       33    64     63     63     60     41
 38  Other    140  235.36  228.57  210.71     .       50    56     61     70     74     39
 39  Other    125  178.60  178.40     .       .       39    50     73     58     58     32
```

```
 40  Other    143  243.01  226.57  210.49     .       38    64     66     70     54     46
 41  Other    134  282.65  239.55     .       .       26    66     65     75     74     53
 44  Other    139  282.37  258.99  238.13  241.01     43    66     69     68     65     39
 45  Other    134  216.04  182.09     .       .       39    50     55     59     67     43
 46  Other    115  190.00  171.30     .       .       36    64     59     58     58     44
 47  Other    134  175.19  167.16     .       .       43    57     48     52     71     47
 48  Other    118  179.87     .       .       .       37    52     57     45     50     30
 50  Other    137  173.54  166.97  164.60     .       32    54     76     63     69     25
 51  Other    125  180.60  162.40  152.00  157.60     32    50     56     64     53     53
 52  Other    155  235.32  225.16  210.32  208.39     35    62     64     55     60     51
 53  Other    143  183.39  169.23     .       .       38    64     57     72     81     61
 54  Other    131  212.60  208.40  211.45     .       27    50     67     53     74     47
 63  Surgery  146  219.18  167.12  139.73  119.18     35    58     53     67     48     55
 65  Surgery  123  192.68  155.28  127.64  115.45     31    52     74     54     64     32
 66  Surgery  134  199.25  173.88  161.19  144.03     38    68     64     70     77     30
 71  Surgery  139  209.35  172.66  156.83  138.13     39    66     56     66     71     42
 77  Surgery  137  179.56  150.36  132.12  123.36     29    46     49     42     47     49
 80  Surgery  170  138.24  121.76  100.00     .       31    56     57     64     64     39
 81  Surgery  129  206.98  173.64  132.56  121.71     28    42     78     52     62     41
 84  Surgery  143  220.28  178.32     .       .       29    58     67     54     46     59
 85  Surgery  152  189.47  156.58  140.79  140.79     34    75     73     62     74     51
 86  Surgery  210  157.14  138.57  121.90  109.05     30    88     75     73     69     65
 92  Surgery  139  203.60  169.78  143.88     .       38    62     59     68     60     49
 93  Surgery  151  171.52  150.33  123.18  109.27     42    72     69     59     53     47
 95  Surgery  134  207.46  155.22     .       .       41    51     52     56     63     57
 96  Surgery  138  201.45  172.46  172.46  155.07     55    57     49     57     67     37
 97  Surgery  128  209.38  182.81  162.50  153.91     34    68     88     73     74      .
101  Surgery  166  185.54  146.39  139.76  132.53     42    82     80     89     79     51
102  Surgery  127  218.11  173.23  152.76  137.80     39    76     80     70     74     45
107  Surgery  128  227.34  192.97  184.38  170.31     49    50     51     59     50     55
110  Surgery  143  251.75  207.69  183.22  165.03     42    48     84     52     76     38
111  Surgery  152  197.37  164.47  148.68  132.89     56    90     70     86     76     70
112  Surgery  140  202.14  156.43  137.86     .       31    48     51     50     41     41
116  Surgery  139  273.38  235.25  194.24     .       31    58     55     66     81     45
117  Surgery  125  192.00  156.80  140.00  127.20     42    58     44     63     53     51
120  Surgery  119  277.31  231.93  208.40  192.44     31    60     69     59     95     34
122  Surgery  138  165.22  130.43  119.57     .       44    66     67     70     62     34
125  Surgery  152  257.89  200.00  182.89  134.21     39    47     84     53     88     74
126  Surgery  138  170.29  142.75     .       .       39    64     64     66     65     46
132  Surgery  149  208.05  173.83     .    126.85     34    82     57     75     65     45
133  Surgery  136  230.15  205.15  190.44  180.88     39    52     71     52     65     30
134  Surgery  168  177.98  140.48  119.64     .       45    70     77     62     67     69
137  Surgery  138  188.41  164.49  152.90  135.51     39    58     61     57     60     46
138  Surgery  141  268.09  220.57  185.82     .       32    52     59     66     74     53
158  Surgery  130  220.00  190.77  173.85     .       27    68     73     66     58     43
223  Surgery  157  220.38  185.35  152.23  138.85     43    78     76     70     57     53
266  Surgery  135  184.44  148.15  123.70     .       41    56     55     59     64     49
269  Surgery  155  201.29  151.61  140.65     .       57    82     84     86     77     73
293  Surgery  160  163.75  130.63     .       .       47    52     67     47     64     76
295  Surgery  128  211.72  172.66  151.56  137.50     45    93     65     80     60     41
298  Surgery  140  243.57  195.00     .    166.43     40    63     62     62     76     52
;
run;
```

CUSTOMER_RESPONSE

```
data customer_response;
   input Customer Factor1-Factor4 Source1-Source3
        Quality1-Quality3;
   datalines;
1 . . 1 1 1 1 . 1 . .
2 1 1 . 1 1 1 . 1 1 .
3 . . 1 1 1 1 . . . .
4 1 1 . 1 . 1 . . . 1
5 . 1 . 1 1 . . . . 1
6 . 1 . 1 1 . . . . .
7 . 1 . 1 1 . . 1 . .
8 1 . . 1 1 1 . 1 1 .
9 1 1 . 1 1 . . . . 1
10 1 . . 1 1 1 . 1 1 .
11 1 1 1 1 . 1 . 1 1 1
12 1 1 . 1 1 1 . . . .
13 1 1 . 1 . 1 . 1 1 .
14 1 1 . 1 1 1 . . . .
15 1 1 . 1 . 1 . 1 1 1
16 1 . . 1 1 . . 1 . .
17 1 1 . 1 1 1 . . 1 .
18 1 1 . 1 1 1 1 . . 1
19 . 1 . 1 1 1 1 . 1 .
20 1 . . 1 1 1 . 1 1 1
21 . . . 1 1 1 . 1 . .
22 . . . 1 1 1 . 1 1 .
23 1 . . 1 . . . . . 1
24 . 1 . 1 1 . . 1 . 1
25 1 1 . 1 1 . . . 1 1
26 1 1 . 1 1 . . 1 . .
27 1 . . 1 1 . . . 1 .
28 1 1 . 1 . . . 1 1 1
29 1 . . 1 1 1 . 1 . 1
30 1 . 1 1 1 . . 1 1 .
31 . . . 1 1 . . 1 1 .
32 1 1 1 1 1 . . 1 1 1
33 1 . . 1 1 . . 1 . 1
34 . . 1 1 . . . 1 1 .
35 1 1 1 1 1 . 1 1 . .
36 1 1 1 1 . 1 . 1 . .
37 1 1 . 1 . . . 1 . .
38 . . . 1 1 1 . 1 . .
39 1 1 . 1 1 . . 1 . 1
40 1 . . 1 . . 1 1 . 1
41 1 . . 1 1 1 1 1 . 1
42 1 1 1 1 . . 1 1 . .
43 1 . . 1 1 1 . 1 . .
44 1 . 1 1 . 1 . 1 . 1
45 . . . 1 . . 1 . . 1
46 . . . 1 1 . . . 1 .
```

```
47 1 1 . 1 . . 1 1 . .
48 1 . 1 1 1 . 1 1 . .
49 . . 1 1 1 1 . 1 . 1
50 . 1 . 1 1 . . 1 1 .
51 1 . 1 1 1 1 . . . .
52 1 1 1 1 1 1 . 1 . .
53 . 1 1 1 . 1 . 1 1 1
54 1 . . 1 1 . . 1 1 .
55 1 1 . 1 1 1 . 1 . .
56 1 . . 1 1 . . 1 1 .
57 1 1 . 1 1 . 1 . . 1
58 . 1 . 1 . 1 . . 1 1
59 1 1 1 1 . . 1 1 1 .
60 . 1 1 1 1 1 . . 1 1
61 1 1 1 1 1 1 . 1 . .
62 1 1 . 1 1 . . 1 1 .
63 . . . 1 . . . 1 1 1
64 1 . . 1 1 1 . 1 . .
65 1 . . 1 1 1 . 1 . .
66 1 . . 1 1 1 1 1 1 .
67 1 1 . 1 1 1 . 1 1 .
68 1 1 . 1 1 1 . 1 1 .
69 1 1 . 1 1 . 1 . . .
70 . . . 1 1 1 . 1 . .
71 1 . . 1 1 . 1 . . 1
72 1 . 1 1 1 1 . . 1 .
73 1 1 . 1 . 1 . 1 1 .
74 1 1 1 1 1 1 . 1 . .
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     15SEP70      07JUN05    203/781-1255
1653    Alexander  Susan     Bridgeport   CT
F       ME2        35108     18OCT72      12AUG98    203/675-7715
1400    Apple      Troy      New York     NY
M       ME1        29769     08NOV85      19OCT06    212/586-0808
1350    Arthur     Barbara   New York     NY
F       FA3        32886     03SEP63      01AUG00    718/383-1549
1401    Avery      Jerry     Paterson     NJ
M       TA3        38822     16DEC68      20NOV93    201/732-8787
1499    Barefoot   Joseph    Princeton    NJ
M       ME3        43025     29APR62      10JUN95    201/812-5665
1101    Baucom     Walter    New York     NY
M       SCP        18723     09JUN80      04OCT98    212/586-8060
1333    Blair      Justin    Stamford     CT
M       PT2        88606     02APR79      13FEB03    203/781-1777
1402    Blalock    Ralph     New York     NY
M       TA2        32615     20JAN71      05DEC98    718/384-2849
1479    Bostic     Marie     New York     NY
F       TA3        38785     25DEC66      08OCT03    718/384-8816
1403    Bowden     Earl      Bridgeport   CT
M       ME1        28072     31JAN79      24DEC99    203/675-3434
1739    Boyce      Jonathan  New York     NY
M       PT1        66517     28DEC82      30JAN00    212/587-1247
1658    Bradley    Jeremy    New York     NY
M       SCP        17943     11APR65      03MAR00    212/587-3622
1428    Brady      Christine Stamford     CT
F       PT1        68767     07APR80      19NOV02    203/781-1212
1782    Brown      Jason     Stamford     CT
M       ME2        35345     07DEC73      25FEB00    203/781-0019
1244    Bryant     Leonard   New York     NY
```

```
M       ME2         36925     03SEP71      20JAN96    718/383-3334
1383    Burnette    Thomas    New York     NY
M       BCK         25823     28JAN76      23OCT00    718/384-3569
1574    Cahill      Marshall  New York     NY
M       FA2         28572     30APR74      23DEC97    718/383-2338
1789    Caraway     Davis     New York     NY
M       SCP         18326     28JAN85      14APR04    212/587-9000
1404    Carter      Donald    New York     NY
M       PT2         91376     27FEB71      04JAN98    718/384-2946
1437    Carter      Dorothy   Bridgeport   CT
F       A3          33104     23SEP68      03SEP92    203/675-4117
1639    Carter      Karen     Stamford     CT
F       A3          40260     29JUN65      31JAN92    203/781-8839
1269    Caston      Franklin  Stamford     CT
M       NA1         41690     06MAY80      01DEC00    203/781-3335
1065    Chapman     Neil      New York     NY
M       ME2         35090     29JAN72      10JAN95    718/384-5618
1876    Chin        Jack      New York     NY
M       TA3         39675     23MAY66      30APR96    212/588-5634
1037    Chow        Jane      Stamford     CT
F       TA1         28558     13APR82      16SEP04    203/781-8868
1129    Cook        Brenda    New York     NY
F       ME2         34929     11DEC79      20AUG03    718/383-2313
1988    Cooper      Anthony   New York     NY
M       FA3         32217     03DEC57      21SEP92    212/587-1228
1405    Davidson    Jason     Paterson     NJ
M       SCP         18056     08MAR54      29JAN00    201/732-2323
1430    Dean        Sandra    Bridgeport   CT
F       TA2         32925     03MAR70      30APR05    203/675-1647
1983    Dean        Sharon    New York     NY
F       FA3         33419     03MAR50      30APR85    718/384-1647
1134    Delgado     Maria     Stamford     CT
F       TA2         33462     08MAR77      24DEC04    203/781-1528
1118    Dennis      Roger     New York     NY
M       PT3         111379    19JAN57      21DEC88    718/383-1122
1438    Donaldson   Karen     Stamford     CT
F       TA3         39223     18MAR63      21NOV03    203/781-2229
1125    Dunlap      Donna     New York     NY
F       FA2         28888     11NOV76      14DEC95    718/383-2094
1475    Eaton       Alicia    New York     NY
F       FA2         27787     18DEC71      16JUL98    718/383-2828
1117    Edgerton    Joshua    New York     NY
M       TA3         39771     08JUN56      16AUG00    212/588-1239
1935    Fernandez   Katrina   Bridgeport   CT
F       NA2         51081     31MAR72      19OCT01    203/675-2962
1124    Fields      Diana     White Plains NY
F       FA1         23177     13JUL82      04OCT01    914/455-2998
1422    Fletcher    Marie     Princeton    NJ
F       FA1         22454     07JUN79      09APR99    201/812-0902
1616    Flowers     Annette   New York     NY
F       TA2         34137     04MAR68      07JUN01    718/384-3329
1406    Foster      Gerald    Bridgeport   CT
M       ME2         35185     11MAR69      20FEB95    203/675-6363
1120    Garcia      Jack      New York     NY
```

```
M       ME1        28619     14SEP80      10OCT01    718/384-4930
1094    Gomez      Alan      Bridgeport   CT
M       FA1        22268     05APR78      20APR99    203/675-7181
1389    Gordon     Levi      New York     NY
M       BCK        25028     18JUL67      21AUG03    718/384-9326
1905    Graham     Alvin     New York     NY
M       PT1        65111     19APR80      01JUN00    212/586-8815
1407    Grant      Daniel    Mt. Vernon   NY
M       PT1        68096     26MAR77      21MAR98    914/468-1616
1114    Green      Janice    New York     NY
F       TA2        32928     21SEP77      30JUN06    212/588-1092
;
```

ENERGY

```
data energy;
   length State $2;
   input Region Division state $ Type Expenditures;
   datalines;
1 1 ME 1 708
1 1 ME 2 379
1 1 NH 1 597
1 1 NH 2 301
1 1 VT 1 353
1 1 VT 2 188
1 1 MA 1 3264
1 1 MA 2 2498
1 1 RI 1 531
1 1 RI 2 358
1 1 CT 1 2024
1 1 CT 2 1405
1 2 NY 1 8786
1 2 NY 2 7825
1 2 NJ 1 4115
1 2 NJ 2 3558
1 2 PA 1 6478
1 2 PA 2 3695
4 3 MT 1 322
4 3 MT 2 232
4 3 ID 1 392
4 3 ID 2 298
4 3 WY 1 194
4 3 WY 2 184
4 3 CO 1 1215
4 3 CO 2 1173
4 3 NM 1 545
4 3 NM 2 578
4 3 AZ 1 1694
4 3 AZ 2 1448
4 3 UT 1 621
4 3 UT 2 438
4 3 NV 1 493
```

```
4 3 NV 2 378
4 4 WA 1 1680
4 4 WA 2 1122
4 4 OR 1 1014
4 4 OR 2 756
4 4 CA 1 10643
4 4 CA 2 10114
4 4 AK 1 349
4 4 AK 2 329
4 4 HI 1 273
4 4 HI 2 298
;
```

EXP Library

The following is a printout of the contents in the EXP library in the DATASETS procedure section.

The following sections are the raw data and DATA steps for the EXP library.

EXP.RESULTS

```
proc datasets library=exp;

data exp.results;
    input id  treat $  initwt  wt3mos age;
    datalines;
 1    Other      166.28    146.98     35
 2    Other      214.42    210.22     54
 3    Other      172.46    159.42     33
 5    Other      175.41    160.66     37
 6    Other      173.13    169.40     20
 7    Other      181.25    170.94     30
10    Other      239.83    214.48     48
11    Other      175.32    162.66     51
12    Other      227.01    211.06     29
13    Other      274.82    251.82     31
14    surgery    203.60    169.78     38
17    surgery    171.52    150.33     42
18    surgery    207.46    155.22     41
;
run;
```

EXP.SUR

```
data exp.sur;
    input id  treat $  initwt  wt3mos  wt6mos  age;
    datalines;
14    surgery    203.60    169.78    143.88     38
17    surgery    171.52    150.33    123.18     42
18    surgery    207.46    155.22         .     41
;
```

```
run;
```

EXPREV

```
ods html close;
data exprev;
input Country $ 1-24 Emp_ID $ 25-32 Order_Date $  Ship_Date $  Sale_Type $ & Quantity Price Cost;

datalines;
```

Antarctica	99999999	1/1/08	1/7/08	Internet	2	92.60	20.70
Puerto Rico	99999999	1/1/08	1/5/08	Catalog	14	51.20	12.10
Virgin Islands (U.S.)	99999999	1/1/08	1/4/08	In Store	25	31.10	15.65
Aruba	99999999	1/1/08	1/4/08	Catalog	30	123.70	59.00
Bahamas	99999999	1/1/08	1/4/08	Catalog	8	113.40	28.45
Bermuda	99999999	1/1/08	1/4/08	Catalog	7	41.00	9.25
Belize	120458	1/2/08	1/2/08	In Store	2	146.40	36.70
British Virgin Islands	99999999	1/2/08	1/5/08	Catalog	11	40.20	20.20
Canada	99999999	1/2/08	1/5/08	Catalog	100	11.80	5.00
Cayman Islands	120454	1/2/08	1/2/08	In Store	20	71.00	32.30
Costa Rica	99999999	1/2/08	1/6/08	Internet	31	53.00	26.60
Cuba	121044	1/2/08	1/2/08	Internet	12	42.40	19.35
Dominican Republic	121040	1/2/08	1/2/08	Internet	13	48.00	23.95
El Salvador	99999999	1/2/08	1/6/08	Catalog	21	266.40	66.70
Guatemala	120931	1/2/08	1/2/08	In Store	13	144.40	65.70
Haiti	121059	1/2/08	1/2/08	Internet	5	47.90	23.45
Honduras	120455	1/2/08	1/2/08	Internet	20	66.40	30.25
Jamaica	99999999	1/2/08	1/4/08	In Store	23	169.80	38.70
Mexico	120127	1/2/08	1/2/08	In Store	30	211.80	33.65
Montserrat	120127	1/2/08	1/2/08	In Store	19	184.20	36.90
Nicaragua	120932	1/2/08	1/2/08	Internet	16	122.00	28.75
Panama	99999999	1/2/08	1/6/08	Internet	20	88.20	38.40
Saint Kitts/Nevis	99999999	1/2/08	1/6/08	Internet	20	41.40	18.00
St. Helena	120360	1/2/08	1/2/08	Internet	19	94.70	47.45
St. Pierre/Miquelon	120842	1/2/08	1/16/08	Internet	16	103.80	47.25
Turks/Caicos Islands	120372	1/2/08	1/2/08	Internet	10	57.70	28.95
United States	120372	1/2/08	1/2/08	Internet	20	88.20	38.40
Anguilla	99999999	1/2/08	1/6/08	In Store	15	233.50	22.25
Antigua/Barbuda	120458	1/2/08	1/2/08	In Store	31	99.60	45.35
Argentina	99999999	1/2/08	1/6/08	In Store	42	408.80	87.15
Barbados	99999999	1/2/08	1/6/08	In Store	26	94.80	42.60
Bolivia	120127	1/2/08	1/2/08	In Store	26	66.00	16.60
Brazil	120127	1/2/08	1/2/08	Catalog	12	73.40	18.45
Chile	120447	1/2/08	1/2/08	In Store	20	19.10	8.75
Colombia	121059	1/2/08	1/2/08	Internet	28	361.40	90.45
Dominica	121043	1/2/08	1/2/08	Internet	35	121.30	57.80
Ecuador	121042	1/2/08	1/2/08	In Store	11	100.90	50.55
Falkland Islands	120932	1/2/08	1/2/08	In Store	15	61.40	30.80
French Guiana	120935	1/2/08	1/2/08	Catalog	15	96.40	43.85
Grenada	120931	1/2/08	1/2/08	Catalog	19	56.30	25.05
Guadeloupe	120445	1/2/08	1/2/08	Internet	21	231.60	48.70
Guyana	120455	1/2/08	1/2/08	In Store	25	132.80	30.25
Martinique	120841	1/2/08	1/3/08	In Store	16	56.30	31.05

```
Netherlands Antilles    99999999     1/2/08      1/6/08          In Store    31    41.80      19.45
Paraguay                120603       1/2/08      1/2/08          Catalog     17    117.60     58.90
Peru                    120845       1/2/08      1/2/08          Catalog     12    93.80      41.75
St. Lucia               120845       1/2/08      1/2/08          Internet    19    64.30      28.65
Suriname                120538       1/3/08      1/3/08          Internet    22    110.80     29.35
;
run;
```

GROC

```
data groc;
   input Region $9. Manager $ Department $ Sales;
   datalines;
Southeast    Hayes       Paper       250
Southeast    Hayes       Produce     100
Southeast    Hayes       Canned      120
Southeast    Hayes       Meat         80
Southeast    Michaels    Paper        40
Southeast    Michaels    Produce     300
Southeast    Michaels    Canned      220
Southeast    Michaels    Meat         70
Northwest    Jeffreys    Paper        60
Northwest    Jeffreys    Produce     600
Northwest    Jeffreys    Canned      420
Northwest    Jeffreys    Meat         30
Northwest    Duncan      Paper        45
Northwest    Duncan      Produce     250
Northwest    Duncan      Canned      230
Northwest    Duncan      Meat         73
Northwest    Aikmann     Paper        45
Northwest    Aikmann     Produce     205
Northwest    Aikmann     Canned      420
Northwest    Aikmann     Meat         76
Southwest    Royster     Paper        53
Southwest    Royster     Produce     130
Southwest    Royster     Canned      120
Southwest    Royster     Meat         50
Southwest    Patel       Paper        40
Southwest    Patel       Produce     350
Southwest    Patel       Canned      225
Southwest    Patel       Meat         80
Northeast    Rice        Paper        90
Northeast    Rice        Produce      90
Northeast    Rice        Canned      420
Northeast    Rice        Meat         86
Northeast    Fuller      Paper       200
Northeast    Fuller      Produce     300
Northeast    Fuller      Canned      420
Northeast    Fuller      Meat        125
;
```

MATCH_11

```
data match_11;
   input Pair Low Age Lwt Race Smoke Ptd Ht UI @@;
   select(race);
      when (1) do;
         race1=0;
         race2=0;
      end;
      when (2) do;
         race1=1;
         race2=0;
      end;
      when (3) do;
         race1=0;
         race2=1;
      end;
   end;
   datalines;
1  0 14 135 1 0 0 0 0     1  1 14 101 3 1 1 0 0
2  0 15  98 2 0 0 0 0     2  1 15 115 3 0 0 0 1
3  0 16  95 3 0 0 0 0     3  1 16 130 3 0 0 0 0
4  0 17 103 3 0 0 0 0     4  1 17 130 3 1 1 0 1
5  0 17 122 1 1 0 0 0     5  1 17 110 1 1 0 0 0
6  0 17 113 2 0 0 0 0     6  1 17 120 1 1 0 0 0
7  0 17 113 2 0 0 0 0     7  1 17 120 2 0 0 0 0
8  0 17 119 3 0 0 0 0     8  1 17 142 2 0 0 1 0
9  0 18 100 1 1 0 0 0     9  1 18 148 3 0 0 0 0
10 0 18  90 1 1 0 0 1     10 1 18 110 2 1 1 0 0
11 0 19 150 3 0 0 0 0     11 1 19  91 1 1 1 0 1
12 0 19 115 3 0 0 0 0     12 1 19 102 1 0 0 0 0
13 0 19 235 1 1 0 1 0     13 1 19 112 1 1 0 0 1
14 0 20 120 3 0 0 0 1     14 1 20 150 1 1 0 0 0
15 0 20 103 3 0 0 0 0     15 1 20 125 3 0 0 0 1
16 0 20 169 3 0 1 0 1     16 1 20 120 2 1 0 0 0
17 0 20 141 1 0 1 0 1     17 1 20  80 3 1 0 0 1
18 0 20 121 2 1 0 0 0     18 1 20 109 3 0 0 0 0
19 0 20 127 3 0 0 0 0     19 1 20 121 1 1 1 0 1
20 0 20 120 3 0 0 0 0     20 1 20 122 2 1 0 0 0
21 0 20 158 1 0 0 0 0     21 1 20 105 3 0 0 0 0
22 0 21 108 1 1 0 0 1     22 1 21 165 1 1 0 1 0
23 0 21 124 3 0 0 0 0     23 1 21 200 2 0 0 0 0
24 0 21 185 2 1 0 0 0     24 1 21 103 3 0 0 0 0
25 0 21 160 1 0 0 0 0     25 1 21 100 3 0 1 0 0
26 0 21 115 1 0 0 0 0     26 1 21 130 1 1 0 1 0
27 0 22  95 3 0 0 1 0     27 1 22 130 1 1 0 0 0
28 0 22 158 2 0 1 0 0     28 1 22 130 1 1 1 0 1
29 0 23 130 2 0 0 0 0     29 1 23  97 3 0 0 0 1
30 0 23 128 3 0 0 0 0     30 1 23 187 2 1 0 0 0
31 0 23 119 3 0 0 0 0     31 1 23 120 3 0 0 0 0
32 0 23 115 3 1 0 0 0     32 1 23 110 1 1 1 0 0
33 0 23 190 1 0 0 0 0     33 1 23  94 3 1 0 0 0
```

```
34 0 24  90 1 1 1 0 0     34 1 24 128 2 0 1 0 0
35 0 24 115 1 0 0 0 0     35 1 24 132 3 0 0 1 0
36 0 24 110 3 0 0 0 0     36 1 24 155 1 1 1 0 0
37 0 24 115 3 0 0 0 0     37 1 24 138 1 0 0 0 0
38 0 24 110 3 0 1 0 0     38 1 24 105 2 1 0 0 0
39 0 25 118 1 1 0 0 0     39 1 25 105 3 0 1 1 0
40 0 25 120 3 0 0 0 1     40 1 25  85 3 0 0 0 1
41 0 25 155 1 0 0 0 0     41 1 25 115 3 0 0 0 0
42 0 25 125 2 0 0 0 0     42 1 25  92 1 1 0 0 0
43 0 25 140 1 0 0 0 0     43 1 25  89 3 0 1 0 0
44 0 25 241 2 0 0 1 0     44 1 25 105 3 0 1 0 0
45 0 26 113 1 1 0 0 0     45 1 26 117 1 1 1 0 0
46 0 26 168 2 1 0 0 0     46 1 26  96 3 0 0 0 0
47 0 26 133 3 1 1 0 0     47 1 26 154 3 0 1 1 0
48 0 26 160 3 0 0 0 0     48 1 26 190 1 1 0 0 0
49 0 27 124 1 1 0 0 0     49 1 27 130 2 0 0 0 1
50 0 28 120 3 0 0 0 0     50 1 28 120 3 1 1 0 1
51 0 28 130 3 0 0 0 0     51 1 28  95 1 1 0 0 0
52 0 29 135 1 0 0 0 0     52 1 29 130 1 0 0 0 1
53 0 30  95 1 1 0 0 0     53 1 30 142 1 1 1 0 0
54 0 31 215 1 1 0 0 0     54 1 31 102 1 1 1 0 0
55 0 32 121 3 0 0 0 0     55 1 32 105 1 1 0 0 0
56 0 34 170 1 0 1 0 0     56 1 34 187 2 1 0 1 0
;
```

PROCLIB.DELAY

```
data proclib.delay;
   input flight $3. +5 date date7. +2 orig $3. +3 dest $3. +3
         delaycat $15. +2 destype $15. +8 delay;
   informat date date7.;
   format date date7.;
   datalines;
114     01MAR08  LGA   LAX   1-10 Minutes     Domestic               8
202     01MAR08  LGA   ORD   No Delay         Domestic              -5
219     01MAR08  LGA   LON   11+ Minutes      International         18
622     01MAR08  LGA   FRA   No Delay         International         -5
132     01MAR08  LGA   YYZ   11+ Minutes      International         14
271     01MAR08  LGA   PAR   1-10 Minutes     International          5
302     01MAR08  LGA   WAS   No Delay         Domestic              -2
114     02MAR08  LGA   LAX   No Delay         Domestic               0
202     02MAR08  LGA   ORD   1-10 Minutes     Domestic               5
219     02MAR08  LGA   LON   11+ Minutes      International         18
622     02MAR08  LGA   FRA   No Delay         International          0
132     02MAR08  LGA   YYZ   1-10 Minutes     International          5
271     02MAR08  LGA   PAR   1-10 Minutes     International          4
302     02MAR08  LGA   WAS   No Delay         Domestic               0
114     03MAR08  LGA   LAX   No Delay         Domestic              -1
202     03MAR08  LGA   ORD   No Delay         Domestic              -1
219     03MAR08  LGA   LON   1-10 Minutes     International          4
622     03MAR08  LGA   FRA   No Delay         International         -2
132     03MAR08  LGA   YYZ   1-10 Minutes     International          6
```

```
271     03MAR08  LGA   PAR   1-10 Minutes     International          2
302     03MAR08  LGA   WAS   1-10 Minutes     Domestic               5
114     04MAR08  LGA   LAX   11+ Minutes      Domestic              15
202     04MAR08  LGA   ORD   No Delay         Domestic              -5
219     04MAR08  LGA   LON   1-10 Minutes     International          3
622     04MAR08  LGA   FRA   11+ Minutes      International         30
132     04MAR08  LGA   YYZ   No Delay         International         -5
271     04MAR08  LGA   PAR   1-10 Minutes     International          5
302     04MAR08  LGA   WAS   1-10 Minutes     Domestic               7
114     05MAR08  LGA   LAX   No Delay         Domestic              -2
202     05MAR08  LGA   ORD   1-10 Minutes     Domestic               2
219     05MAR08  LGA   LON   1-10 Minutes     International          3
622     05MAR08  LGA   FRA   No Delay         International         -6
132     05MAR08  LGA   YYZ   1-10 Minutes     International          3
271     05MAR08  LGA   PAR   1-10 Minutes     International          5
114     06MAR08  LGA   LAX   No Delay         Domestic              -1
202     06MAR08  LGA   ORD   No Delay         Domestic              -3
219     06MAR08  LGA   LON   11+ Minutes      International         27
132     06MAR08  LGA   YYZ   1-10 Minutes     International          7
302     06MAR08  LGA   WAS   1-10 Minutes     Domestic               1
114     07MAR08  LGA   LAX   No Delay         Domestic              -1
202     07MAR08  LGA   ORD   No Delay         Domestic              -2
219     07MAR08  LGA   LON   11+ Minutes      International         15
622     07MAR08  LGA   FRA   11+ Minutes      International         21
132     07MAR08  LGA   YYZ   No Delay         International         -2
271     07MAR08  LGA   PAR   1-10 Minutes     International          4
302     07MAR08  LGA   WAS   No Delay         Domestic               0
;
```

PROCLIB.EMP95

```
data proclib.emp95;
   input #1 idnum $4. @6 name $15.
         #2 address $42.
         #3 salary 6.;
   datalines;
2388 James Schmidt
100 Apt. C Blount St. SW Raleigh NC 27693
92100
2457 Fred Williams
99 West Lane  Garner NC 27509
33190
2776 Robert Jones
12988 Wellington Farms Ave. Cary NC 27512
29025
8699 Jerry Capalleti
222 West L St. Oxford NC 27587
39985
2100 Lanny Engles
293 Manning Pl. Raleigh NC 27606
30998
9857 Kathy Krupski
```

```
1000 Taft Ave. Morrisville NC 27508
38756
0987 Dolly Lunford
2344 Persimmons Branch  Apex NC 27505
44010
3286 Hoa Nguyen
2818 Long St. Cary NC 27513
87734
6579 Bryan Samosky
3887 Charles Ave. Garner NC 27508
50234
3888 Kim Siu
5662 Magnolia Blvd Southeast Cary NC 27513
77558
;
```

PROCLIB.EMP96

```
data proclib.emp96;
   input #1 idnum $4. @6 name $15.
         #2 address $42.
         #3 salary 6.;
   datalines;
2388 James Schmidt
100 Apt. C Blount St. SW Raleigh NC 27693
92100
2457 Fred Williams
99 West Lane  Garner NC 27509
33190
2776 Robert Jones
12988 Wellington Farms Ave. Cary NC 27511
29025
8699 Jerry Capalleti
222 West L St. Oxford NC 27587
39985
3278 Mary Cravens
211 N. Cypress St. Cary NC 27512
35362
2100 Lanny Engles
293 Manning Pl. Raleigh NC 27606
30998
9857 Kathy Krupski
100 Taft Ave. Morrisville NC 27508
40456
0987 Dolly Lunford
2344 Persimmons Branch Trail Apex NC 27505
45110
3286 Hoa Nguyen
2818 Long St. Cary NC 27513
89834
6579 Bryan Samosky
```

```
3887 Charles Ave. Garner NC 27508
50234
3888 Kim Siu
5662 Magnolia Blvd Southwest Cary NC 27513
79958
6544 Roger Monday
3004 Crepe Myrtle Court Raleigh NC 27604
47007
;
```

PROCLIB.INTERNAT

```
data proclib.internat;
   input flight $3.  +5 date date7. +2 dest $3. +8 boarded;
   informat date date7.;
   format date date7.;
   datalines;
219     01MAR08  LON        198
622     01MAR08  FRA        207
132     01MAR08  YYZ        115
271     01MAR08  PAR        138
219     02MAR08  LON        147
622     02MAR08  FRA        176
132     02MAR08  YYZ        106
271     02MAR08  PAR        172
219     03MAR08  LON        197
622     03MAR08  FRA        180
132     03MAR08  YYZ         75
271     03MAR08  PAR        147
219     04MAR08  LON        232
622     04MAR08  FRA        137
132     04MAR08  YYZ        117
271     04MAR08  PAR        146
219     05MAR08  LON        160
622     05MAR08  FRA        185
132     05MAR08  YYZ        157
271     05MAR08  PAR        177
219     06MAR08  LON        163
132     06MAR08  YYZ        150
219     07MAR08  LON        241
622     07MAR08  FRA        210
132     07MAR08  YYZ        164
271     07MAR08  PAR        155
;
```

PROCLIB.LAKES

```
data proclib.lakes;
    input region $ 1-2 lake $ 5-13 pol_a1 pol_a2 pol_b1-pol_b4;
    datalines;
```

```
NE  Carr       0.24     0.99     0.95     0.36     0.44     0.67
NE  Duraleigh  0.34     0.01     0.48     0.58     0.12     0.56
NE  Charlie    0.40     0.48     0.29     0.56     0.52     0.95
NE  Farmer     0.60     0.65     0.25     0.20     0.30     0.64
NW  Canyon     0.63     0.44     0.20     0.98     0.19     0.01
NW  Morris     0.85     0.95     0.80     0.67     0.32     0.81
NW  Golf       0.69     0.37     0.08     0.72     0.71     0.32
NW  Falls      0.01     0.02     0.59     0.58     0.67     0.02
SE  Pleasant   0.16     0.96     0.71     0.35     0.35     0.48
SE  Juliette   0.82     0.35     0.09     0.03     0.59     0.90
SE  Massey     1.01     0.77     0.45     0.32     0.55     0.66
SE  Delta      0.84     1.05     0.90     0.09     0.64     0.03
SW  Alumni     0.45     0.32     0.45     0.44     0.55     0.12
SW  New Dam    0.80     0.70     0.31     0.98     1.00     0.22
SW  Border     0.51     0.04     0.55     0.35     0.45     0.78
SW  Red        0.22     0.09     0.02     0.10     0.32     0.01
;
```

PROCLIB.MARCH

```
data proclib.march;
   input flight $3. +5 date date7. +3 depart time5. +2 orig $3.
         +3 dest $3.  +7 miles +6 boarded +6 capacity;
   format date date7. depart time5.;
   informat date date7. depart time5.;
   datalines;
114     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
;
```

h

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 1300 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 0 5 5 0 0 0 0 0 5
493 35 m 4 4 7
7 5 0 5 7 0 0 7 7 7 7 0 0 0
677 27 m 15 11 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
848 27 f 3 5 1
1 1 0 0 1 1 0 0 1 1 1 1 0 0
362 30 f 1 0 1
1 0 0 0 7 5 0 0 0 0 0 0 0 0
783 29 f 1 1 4
4 0 0 0 4 0 0 0 4 0 0 0 4 0
467 39 f .5 2 4
7 0 4 0 4 4 0 0 4 4 4 4 4 4
677 27 m 2 2 7
7 0 0 0 7 0 0 7 7 0 0 7 0 0
221 23 f 2.5 1 1
1 0 0 0 1 0 0 0 0 0 0 0 0 0
677 29 f 1 1 7
0 0 0 0 7 0 0 0 7 0 0 0 0 0
783 32 m 1 2 5
```

```
4 5 5 5 4 2 0 0 0 0 3 2 2 0
833 25 f 1 0 1
1 1 0 0 0 0 0 0 0 0 0 0 0 0
859 24 f 7 3 7
1 0 0 0 1 0 0 0 0 1 0 0 1 0
677 29 m 2 2 8
0 8 8 0 8 0 0 0 8 8 8 0 0 0
688 31 m 8 2 5
7 5 5 5 5 7 0 0 7 7 0 0 0 0
856 31 m 9 4 1
1 1 1 1 1 0 0 0 0 0 0 0 1 0
856 44 f 1 0 6
6 0 0 0 6 0 0 0 0 0 0 0 0 0
677 37 f 3 3 1
0 0 1 0 0 0 0 0 4 4 0 0 0 0
859 27 m 2 .5 2
2 2 2 2 2 2 2 2 0 0 0 0 0 2
781 30 f 10 4 2
2 0 0 0 2 0 2 0 0 0 0 0 0 2
362 27 m 12 4 3
3 1 1 1 1 3 3 3 0 0 0 0 3 0
362 33 f 2 4 1
1 0 0 0 7 0 0 7 1 1 1 1 1 0
222 26 f 8 1 1
1 1 1 1 0 0 0 1 0 0 0 0 0 0
779 37 f 6 3 1
1 1 1 1 1 0 0 1 1 0 0 0 1 0
467 32 f 1 1 2
2 0 0 0 0 0 0 0 2 0 0 2 0 0
859 23 m 1 1 1
1 0 0 0 1 1 0 1 0 0 0 0 1 1
781 33 f 1 .5 6
6 0 0 0 6 0 0 0 0 0 0 0 0 0
779 28 m 5 2 1
1 1 1 1 1 0 0 0 0 7 7 1 1 0
677 28 m 3 1 5
7 5 5 5 5 6 0 0 6 6 6 6 6 0
677 25 f 9 2 5
1 5 5 5 5 1 1 0 1 1 1 1 1 1
848 30 f 6 2 8
8 0 0 0 2 7 0 0 0 0 2 0 2 0
546 36 f 4 6 4
7 0 0 0 4 4 0 5 5 5 5 2 4 4
222 30 f 2 3 2
2 2 0 0 2 0 0 0 2 0 2 2 0 0
383 32 m 4 1 2
2 0 0 0 2 0 0 2 0 0 0 0 0 0
851 43 f 8 1 6
4 6 0 6 4 0 0 0 0 0 0 0 0 0
222 27 f 1 3 1
1 1 0 1 1 1 0 0 1 0 0 0 4 0
833 22 f 1.5 2 1
1 0 0 0 1 1 0 0 1 1 1 0 0 0
467 29 f 2 1 8
```

```
8 0 8 0 8 0 0 0 0 0 8 0 0 0
856 28 f 2 3 1
1 0 0 0 1 0 0 0 1 0 0 1 0 0
580 31 f 2.5 2.5 6
6 6 6 6 6 6 6 6 1 1 1 1 6 6
688 39 f 8 8 3
3 3 3 3 3 3 3 3 3 3 3 3 3 3
677 37 f 1.5 .5 1
6 1 1 1 6 6 0 0 1 1 6 6 6 0
859 38 m 3 6 3
7 0 0 0 7 3 0 0 3 0 3 0 0 0
677 25 f 7 1 1
0 1 1 1 2 0 0 0 1 2 1 1 1 0
848 36 f 7 1 1
0 1 0 1 1 0 0 0 0 0 0 1 1 0
781 31 f 2 4 1
1 0 0 0 1 1 0 1 1 1 1 1 0 0
781 40 f 2 2 8
8 0 0 8 8 0 0 0 0 0 8 8 0 0
677 25 f 3 5 1
1 6 1 6 6 3 0 0 2 2 1 1 1 1
779 33 f 3 2 1
1 0 1 0 0 0 1 0 1 0 0 0 1 0
677 25 m 7 1.5 1
1 1 0 1 1 0 0 0 0 0 1 0 0 0
362 35 f .5 0 1
1 0 0 0 1 0 0 0 0 0 0 0 0 0
677 41 f 6 2 7
7 7 0 7 7 0 0 0 0 0 8 0 0 0
677 24 m 5 1 5
1 5 0 5 0 0 0 0 1 0 0 0 0 0
833 29 f .5 0 6
6 0 0 0 6 0 0 0 0 0 0 0 0 0
362 30 f 1 1 1
1 0 0 0 1 0 0 0 1 0 0 0 0 0
850 26 f 6 12 6
6 0 0 0 2 2 2 6 6 6 0 0 6 6
467 25 f 2 3 1
1 0 0 6 1 1 0 0 0 0 1 1 1 1
967 29 f 1 2 7
7 0 0 0 7 0 0 7 7 0 0 0 0 0
833 31 f 1 1 7
7 0 7 0 7 3 0 0 3 3 0 0 0 0
859 40 f 7 1 5
1 5 0 5 5 1 0 0 1 0 0 0 0 0
848 31 m 1 2 1
1 0 0 0 1 1 0 0 4 4 1 4 0 0
222 32 f 2 3 3
3 0 0 0 0 7 0 0 3 0 8 0 0 0
783 33 f 2 0 4
7 0 0 0 7 0 0 0 4 0 4 0 0 0
856 28 f 8 4 2
0 2 0 2 2 0 0 0 2 0 2 0 4 0
781 30 f 3 5 1
```

```
1 1 1 1 1 1 0 0 1 1 1 1 1 0
850 25 f 6 3 1
7 5 0 5 7 1 0 0 7 0 1 0 1 0
580 33 f 2.5 4 2
2 0 0 0 2 0 0 0 0 0 8 8 0 0
677 38 f 3 3 1
1 0 0 0 1 0 1 1 1 0 1 0 0 4
677 26 f 2 2 1
1 0 1 0 1 0 0 0 1 1 1 0 0 0
467 52 f 3 2 2
2 6 6 6 6 2 0 0 2 2 2 2 0 0
542 31 f 1 3 1
1 0 1 0 1 0 0 0 1 1 1 1 1 0
859 50 f 9 3 6
6 6 6 6 6 6 6 6 6 3 3 3 6 6
779 26 f 1 2 1
7 0 1 0 1 1 4 1 4 1 1 1 4 4
779 36 m 1.5 2 4
1 4 0 4 4 0 0 4 4 4 4 4 0 0 0
222 31 f 0 3 7
1 0 0 0 7 0 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 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 0 8 8 0 0 0

467 38 m 3 5 1

1 0 0 0 1 0 0 1 1 0 0 0 0 0

781 29 f 2 3 8

8 0 0 0 8 8 0 0 8 8 0 8 8 0

781 30 f 1 0 5

5 0 0 0 0 5 0 0 0 0 0 0 0 0

783 40 f 1.5 3 1

1 0 0 0 1 4 0 0 1 1 1 0 0 0

851 30 f 1 1 6

6 0 0 0 6 0 0 0 6 0 0 6 0 0

851 40 f 1 1 5

5 0 0 0 5 0 0 0 0 1 0 0 0 0

779 40 f 1 0 2

```
2 0 0 0 2 0 0 0 0 0 0 0 0 0
467 37 f 4 8 1
1 0 0 0 1 0 3 0 3 1 1 1 0 0
859 37 f 4 3 3
0 3 7 0 0 7 0 0 0 7 8 3 7 0
781 26 f 4 1 2
2 2 0 2 1 0 0 0 2 0 0 0 0 0
859 23 f 8 3 3
3 2 0 2 3 0 0 0 1 0 0 3 0 0
967 31 f .5 0 1
1 0 0 0 0 0 0 0 0 0 0 0 0 0
851 38 m 4 2 5
7 5 0 5 4 0 4 7 7 0 4 0 8 0
467 30 m 2 1 2
2 2 0 2 0 0 0 0 2 0 2 0 0 0
848 33 f 2 2 7
7 0 0 0 0 7 0 7 7 0 0 0 7 0
688 35 f 5 8 3
2 2 2 2 2 0 0 3 3 3 3 3 0 0
467 27 f 2 3 1
1 0 1 0 0 1 0 0 1 1 1 0 0 0
783 42 f 3 1 1
1 0 0 0 1 0 0 0 1 0 1 1 0 0
687 40 m 1.5 2 1
7 0 0 0 1 1 0 0 1 0 7 0 1 0
779 30 f 4 8 7
7 0 0 0 7 0 6 7 4 2 2 0 0 6
222 34 f 9 0 8
8 2 0 2 8 0 0 0 0 0 0 0 0 0
467 28 m 3 1 2
2 0 0 0 2 2 0 0 0 2 2 0 0 0
222 28 f 8 4 2
1 2 1 2 2 0 0 1 2 2 0 0 2 0
542 35 m 2 3 2
6 0 7 0 7 0 7 0 0 0 2 2 0 0
677 31 m 12 4 3
7 3 0 3 3 4 0 0 4 4 4 0 0 0
783 45 f 1.5 2 6
6 0 0 0 6 0 0 6 6 0 0 0 0 0
942 34 f 1 .5 4
4 0 0 0 1 0 0 0 0 0 2 0 0 0
222 30 f 8 4 1
1 1 1 1 1 0 0 0 1 1 0 0 0 0
967 38 f 1.5 2 7
7 0 0 0 7 0 0 7 1 1 1 1 0 0
783 37 f 2 1 1
6 6 1 1 6 6 0 0 6 1 1 1 6 0
467 31 f 1.5 2 2
2 0 7 0 7 0 0 7 7 0 0 0 7 0
859 48 f 3 0 7
7 0 0 0 0 0 0 0 0 7 0 0 0 0
490 35 f 1 1 7
7 0 0 0 7 0 0 0 0 0 0 0 8 0
222 27 f 3 2 3
```

```
8 0 0 0 3 8 0 3 3 0 0 0 0 0
382 36 m 3 2 4
7 0 5 4 7 4 4 0 7 7 4 7 0 4
859 37 f 1 1 2
7 0 0 0 0 2 0 2 2 0 0 0 0 2
856 29 f 3 1 1
1 0 0 0 1 1 1 1 0 0 1 1 0 1
542 32 m 3 3 7
7 0 0 0 0 7 7 7 0 0 0 0 7 7
783 31 m 1 1 1
1 0 0 0 1 0 0 0 1 1 1 0 0 0
833 35 m 1 1 1
5 4 1 5 1 0 0 1 1 0 0 0 0 0
782 38 m 30 8 5
7 5 5 5 5 0 0 4 4 4 4 4 0 0
222 33 m 3 3 1
1 1 1 1 1 1 1 1 4 1 1 1 1 1
467 24 f 2 4 1
0 0 1 0 1 0 0 0 1 1 1 0 0 0
467 34 f 1 1 1
1 0 0 0 1 0 0 1 1 0 0 0 0 0
781 53 f 2 1 5
5 0 0 0 5 5 0 0 0 0 5 5 5 0
222 30 m 2 5 3
6 3 3 3 6 0 0 0 3 3 3 3 0 0
688 26 f 2 2 1
1 0 0 0 1 0 0 0 1 0 1 1 0 0
222 29 m 8 5 1
1 6 0 6 1 0 0 1 1 1 1 0 0 0
783 33 m 1 2 7
7 0 0 0 7 0 0 0 7 0 0 0 7 0
781 39 m 1.5 2.5 2
2 0 2 0 2 0 0 0 2 2 2 0 0 0
850 22 f 2 1 1
1 0 0 0 1 1 1 0 5 0 0 1 0 0
493 36 f 1 0 5
0 0 0 0 7 0 0 0 0 0 0 0 0 0
967 46 f 2 4 7
7 5 0 5 7 0 0 0 4 7 4 0 0 0
856 41 m 2 2 4
7 4 0 0 7 4 0 4 0 0 0 7 0 0
546 25 m 5 5 8
8 8 0 0 0 0 0 0 0 0 0 0 0 0
222 27 f 4 4 3
2 2 2 3 7 7 0 2 2 2 3 3 3 0
688 23 m 9 3 3
3 3 3 3 3 7 0 0 3 0 0 0 0 0
849 26 m .5 .5 8
8 0 0 0 8 0 0 0 0 8 0 0 0 0
783 29 f 3 3 1
1 0 0 0 4 0 0 4 1 0 1 0 0 0
856 34 f 1.5 2 1
7 0 0 0 7 0 0 7 4 0 0 7 0 0
966 33 m 3 5 4
```

```
7 0 0 0 7 4 5 0 7 0 0 7 4 4
493 34 f 2 5 1
1 0 0 0 1 0 0 0 7 0 1 1 8 0
467 29 m 2 4 2
2 0 0 0 2 0 0 2 2 2 2 2 2 2
677 28 f 1 4 1
1 1 1 1 1 0 0 0 1 0 1 0 0 0
781 27 m 2 2 1
1 0 1 0 4 2 4 0 2 2 1 0 1 4
467 24 m 4 4 1
7 1 0 1 1 1 0 7 1 0 0 0 0 0
859 26 m 5 5 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
848 27 m 7 2 5
7 5 0 5 4 5 0 0 0 7 4 4 0 4
677 25 f 1 2 8
8 0 0 0 0 5 0 0 8 0 0 0 2 0
222 26 f 3.5 0 2
2 0 0 0 2 0 0 0 0 0 0 0 0 0
833 32 m 1 2 1
1 0 0 0 1 0 0 0 5 0 1 0 0 0
781 28 m 2 .5 7
7 0 0 0 7 0 0 0 4 0 0 0 0 0
783 28 f 1 1 1
1 0 0 0 1 0 0 0 0 0 1 1 0 0
222 28 f 5 5 2
2 6 6 2 2 0 0 0 2 2 0 0 2 2
851 33 m 4 5 3
1 0 0 0 7 3 0 3 3 3 3 3 7 5
859 39 m 2 1 1
1 0 0 0 1 0 0 0 0 0 0 1 0 0
848 45 m 2 2 7
7 0 0 0 7 0 0 0 7 0 0 0 0 0
467 37 m 2 2 7
7 0 0 0 0 7 0 0 0 7 0 0 7 0
859 32 m .25 .25 1
1 0 0 0 0 0 0 0 1 0 0 0 0 0
```

SALES

```
data sales;
   input Region $ CitySize $ Population Product $ SaleType $ Units NetSales;
   cards;
NC S 25000 A100 R 150 3750.00
NC M  125000 A100 R 350 8650.00
NC L 837000 A100 R 800 20000.00
NC S 25000 A100 W 150 3000.00
NC M 125000 A100 W 350 7000.00
NC M 625000 A100 W 750 15000.00
TX M 227000 A100 W 350 7250.00
TX L 5000 A100 W 750 5000.00
;
```

APPENDIX

4

ICU License

ICU License - ICU 1.8.1 and later **1643**

ICU License - ICU 1.8.1 and later

APPENDIX
5

Recommended Reading

Recommended Reading **1645**

Recommended Reading

Here is the recommended reading list for this title:

- *The Little SAS Book: A Primer, Second 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 9.2 SQL Procedure User's Guide*
- *Step-by-Step Programming with Base SAS Software*

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

SAS Publishing Sales
SAS Campus Drive
Cary, NC 27513
Telephone: 1-800-727-3228
Fax: 1-919-531-9439
E-mail: **sasbook@sas.com**
Web address: **support.sas.com/bookstore**

Customers outside the United States and Canada, please contact your local SAS office for assistance.

Index

A

ABORT statement
  FCMP procedure 424
ACCELERATE= option
  ITEM statement (PMENU) 784
ACROSS option
  DEFINE statement (REPORT) 1037
across variables 989, 1037
activities data set 66, 87
ADDMATRIX CALL routine 452
AFTER= option
  PROC CPORT statement 272
AGE statement
  DATASETS procedure 300
aging data sets 388
aging files 300
ALL class variable 1427
ALL keyword 1272
ALL option
  PROC JAVAINFO statement 607
ALLOBS option
  PROC COMPARE statement 213
ALLSTATS option
  PROC COMPARE statement 213
ALLVARS option
  PROC COMPARE statement 213
ALPHA= option
  PROC MEANS statement 614
  PROC TABULATE statement 1364
ALTER= option
  AGE statement (DATASETS) 300
  CHANGE statement (DATASETS) 313
  COPY statement (DATASETS) 319
  DELETE statement (DATASETS) 328
  EXCHANGE statement (DATASETS) 332
  MODIFY statement (DATASETS) 343
  PROC DATASETS statement 297
  REBUILD statement (DATASETS) 347
  REPAIR statement (DATASETS) 349
  SELECT statement (DATASETS) 352
ALTER TABLE statement
  SQL procedure 1213
alternative hypotheses 1565
ANALYSIS option
  DEFINE statement (REPORT) 1037
analysis variables 635, 988, 1037
  SUMMARY procedure 1353
  TABULATE procedure 1389, 1391
  weights for 42, 1051, 1391
ANSI Standard
  SQL procedure and 1293
APPEND procedure 55
  syntax 55
APPEND statement
  DATASETS procedure 302
appending data sets 302
  APPEND procedure versus APPEND statement 309
  block I/O method 304
  compressed data sets 306
  indexed data sets 306
  integrity constraints and 308
  password-protected data sets 305
  restricting observations 305
  SET statement versus APPEND statement 305
  system failures 309
  variables with different attributes 307
  with different variables 307
  with generation groups 308
appending observations 55
APPENDVER= option
  APPEND statement (DATASETS) 303
argument lists
  updating 431
arithmetic mean 1539, 1545
arithmetic operators 1295
ARRAY statement
  FCMP procedure 424
arrays
  changing size, within a function 471
  DATA step versus FCMP procedure 436
  FCMP procedure and 438
  passing 438
  reading and writing to a data set 439
  resizing 439
  temporary 472
  variable arguments with 428
ASCENDING option
  CHART procedure 169
  CLASS statement (MEANS) 623
  CLASS statement (TABULATE) 1375
  KEY statement (SORT) 1181
ASCII collating sequence 1169, 1183
ASCII option
  PROC SORT statement 1169
ASIS option
  PROC CPORT statement 272
asterisk (*) notation 1234
ATTR option
  RECORD statement (SCAPROC) 1145

TEXT statement (PMENU) 791
ATTRIB statement
DATASETS procedure 309
FCMP procedure 426
procedures and 35
audit files
creating 312
event logging 311
AUDIT statement
DATASETS procedure 311
audit trails
migrating data files with 689
AUDIT_ALL= option
AUDIT statement (DATASETS) 311
authentication 593
AUTOLABEL option
OUTPUT statement (MEANS) 632
AUTONAME option
OUTPUT statement (MEANS) 632
axes
customizing 1491
AXIS= option
CHART procedure 169
PLOT statement (TIMEPLOT) 1484
AXIS2CONFIGDIR option
PROC SOAP statement 1155
AXIS2CONFIGFILE option
PROC SOAP statement 1155

B

bar charts 156
horizontal 156, 165, 185
maximum number of bars 164
percentage charts 177
side-by-side 182
vertical 156, 167, 179
BASE= argument
APPEND statement (DATASETS) 302
base data set 208
BASE= option
PROC COMPARE statement 214
batch mode
creating printer definitions 921
printing from 1002
BATCH option
PROC DISPLAY statement 402
BETWEEN condition 1247
Black-Scholes implied volatility 470
BLANKLINE=
PROC PRINT statement 819
block charts 157, 163
for BY groups 186
block I/O method
appending data sets 304
copying data sets 323
BLOCK statement
CHART procedure 163
BOX option
PLOT statement (PLOT) 729
PROC REPORT statement 1008
TABLE statement (TABULATE) 1381
BREAK automatic variable 995
break lines 995
BREAK automatic variable 995
creating 995
order of 995, 1026, 1051
BREAK statement
REPORT procedure 1022
BREAK window
REPORT procedure 1052
breaks 995
BRIEFSUMMARY option
PROC COMPARE statement 214
browsing external files 577
BTRIM function (SQL) 1248
BUFFERSIZE= option
PROC SQL statement 1205
BUFSIZE= option
PROC MIGRATE statement 686
buttons 783
BY-group information
titles containing 21
BY-group processing 21, 37
error processing for 25
formats and 31
TABULATE procedure 1394
BY groups
block charts for 186
complex transposition 1503
maintaining order of observations in 1191
plotting 758
retaining first observation of 1193
transposing 1516
transpositions with 1506
BY lines
inserting into titles 24
suppressing the default 21
BY processing
COMPARE procedure 219
BY statement 36
BY-group processing 37
CALENDAR procedure 72
CHART procedure 164
COMPARE procedure 218
example 38
formatting BY-variable values 37
MEANS procedure 621, 650
options 36
PLOT procedure 726
PRINT procedure 828
procedures supporting 37
RANK procedure 949
REPORT procedure 1027
SORT procedure 1179
STANDARD procedure 1341
TABULATE procedure 1373
TIMEPLOT procedure 1481
TRANSPOSE procedure 1506
BY variables
formatting values 37
inserting names into titles 24
inserting values into titles 22

C

C argument types 908
C helper functions and CALL routines 463, 916
C language
structure types 429
C language types 909
C or C++ functions

See PROTO procedure
C return types 908
C source code 914
C structures in SAS 911
declaring and referencing 912
enumerations 913
limitations for 915
calculated columns
SQL 1249
CALCULATED component 1249
CALEDATA= option
PROC CALENDAR statement 66
calendar, defined 84
calendar data set 66, 89
multiple calendars 85, 86
CALENDAR procedure 59, 64
activities data set 87
activity lines 93
advanced scheduling 63
calendar data set 89
calendar types 59, 82
concepts 82
customizing calendar appearance 93
default calendars 83
duration 74
holiday duration 76
holidays data set 88
input data sets 86
missing values 91
multiple calendars 59, 72, 84
ODS portability 93
output, format of 92
output, quantity of 92
project management 63
results 92
schedule calendars 82
scheduling 114
summary calendars 83
syntax 64
task tables 64, 65
workdays data set 90
calendar reports 84
CALID statement
CALENDAR procedure 73
CALL DEFINE statement
REPORT procedure 1028
CALL routines
See also FCMP procedure
C helper 463
declaring 435
matrix 451
special 451
call stacks 444
CAPS option
PROC FSLIST statement 579
Cartesian product 1259, 1260
case-control studies 1328
CASE expression 1249
CATALOG= argument
PROC DISPLAY statement 402
catalog concatenation 145
catalog entries
copying 137, 142, 147
deleting 134, 138, 147, 153
displaying contents of 151
excluding, for copying 140
exporting 280, 283
importing 204
modifying descriptions of 141, 151
moving, from multiple catalogs 147
renaming 135, 151
routing log or output to entries 895
saving from deletion 141
switching names of 139
CATALOG= option
CONTENTS statement (CATALOG) 136
PROC PMENU statement 779
CATALOG procedure 131, 132
catalog concatenation 145
concepts 142
ending a step 143
entry type specification 144
error handling 143
interactive processing with RUN groups 143
results 146
syntax 132
task tables 132, 133, 137
catalogs
concatenating 145
exporting multiple 279
format catalogs 539
listing contents of 135
locking 137
MIGRATE procedure and unsupported catalogs 699
migrating 689
PMENU entries 779, 786, 792
repairing 349
categories 1361
headings for 1407
categories of procedures 3
CC option
FSLIST command 582
PROC FSLIST statement 579
CEDA processing
migration and 694
CENTER option
DEFINE statement (REPORT) 1037
PROC REPORT statement 1008
centiles 338
CENTILES option
CONTENTS statement (DATASETS) 315
CFREQ option
CHART procedure 169
CHANGE statement
CATALOG procedure 135
DATASETS procedure 313
character data
converting to numeric values 554
in FUNCTION statement (FCMP) 428
character strings
converting to lowercase 1270
converting to uppercase 1293
formats for 532
ranges for 566
returning a substring 1284
trimming 1248
character values
formats for 549
character variables
PROTO procedure 909
sorting orders for 1182

CHART procedure 155, 161
  bar charts 156, 182
  block charts 157, 163, 186
  concepts 173
  customizing charts 168
  formatting characters 161
  frequency counts 175
  horizontal bar charts 165, 185
  missing values 171, 174
  ODS output 174
  ODS table names 174
  options 169
  percentage bar charts 177
  pie charts 158, 166
  results 174
  star charts 159, 166
  syntax 161
  task table 168
  variable characteristics 173
  vertical bar charts 167, 179
charts
  bar charts 156, 177, 182
  block charts 157, 163, 186
  customizing 168
  horizontal bar charts 165, 185
  missing values 171
  pie charts 158, 166
  star charts 159, 166
  vertical bar charts 167, 179
CHARTYPE option
  PROC MEANS statement 614
check boxes 780, 782
  active vs. inactive 780
  color of 780
CHECKBOX statement
  PMENU procedure 780
CHOL CALL routine 453
Cholesky decomposition 453
CIMPORT procedure 192
  overview 191
  syntax 192
  task table 193
CLASS statement
  MEANS procedure 622
  TABULATE procedure 1374
  TIMEPLOT procedure 1482
class variables 622
  BY statement (MEANS) with 650
  CLASSDATA= option (MEANS) with 652
  combinations of 634, 636, 1415
  computing descriptive statistics 648
  formatting in TABULATE 1393
  level value headings 1378
  MEANS procedure 638
  missing 1404, 1405, 1406
  missing values 625, 667, 1377
  multilabel value formats with 655
  ordering values 638
  preloaded formats with 659, 1418
  TABULATE procedure 1374
  TIMEPLOT procedure 1482
CLASSDATA= option
  PROC MEANS statement 615, 652
  PROC TABULATE statement 1364
classifying formatted data 28
CLASSLEV statement
  TABULATE procedure 1378
CLASSPATHS option
  PROC JAVAINFO statement 608
CLEARSASUSER option
  PROC REGISTRY statement 965
CLM keyword 1543
CLONE option
  COPY statement (DATASETS) 319
CMPLIB= system option 448
CNTLIN= option
  PROC FORMAT statement 515
CNTLOUT= option
  PROC FORMAT statement 515, 516, 530
COALESCE function (SQL) 1251
Code Analyzer
  See SAS Code Analyzer
code blocks
  declaring for subroutines 430
coefficient of variation 1538, 1551
collating sequence 1169
  alternate 1169, 1294
  ASCII 1169, 1183
  based on National Use Differences 1169
  Danish 1169
  default 1182
  EBCDIC 1169, 1182
  Finnish 1169
  Norwegian 1169
  Polish 1169
  specifying 1170
  specifying for character variables 1183
  Swedish 1169
collision states 741
COLOR= option
  BREAK statement (REPORT) 1022
  CHECKBOX statement (PMENU) 780
  DEFINE statement (REPORT) 1037
  RBREAK statement (REPORT) 1048
  RBUTTON statement (PMENU) 788
  TEXT statement (PMENU) 791
column aliases 1235
column attributes 1216, 1252
  reports 1028
column-definition component 1251
column-header option
  DEFINE statement (REPORT) 1038
column headings
  customizing 1425
  customizing text in 841
  page layout 834
column-modifier component 1252
column modifiers 1294
column-name component 1254
column names
  specifying with WRITE_ARRAY function 442
COLUMN statement
  REPORT procedure 1030
column width 835
columns
  aliases 1235
  altering 1213
  calculated 1249
  combinations of values 1325
  for each variable value 1099
  in reports 1030

indexes on 1216, 1218, 1232
inserting values 1231
length of 1253
modifiers 1294
renaming 1216, 1233
returning values 1251
selecting 1233, 1254
SQL procedure 1199
storing values of 1235
updating values 1245
COLWIDTH= option
PROC REPORT statement 1008
COMMAND option
PROC REPORT statement 1009
COMMIT statement (SQL) 1296
COMPARE= option
PROC COMPARE statement 214
COMPARE procedure 208, 211
BY processing 219
comparing selected variables 222
comparing unsorted data 220
comparing variables 222
comparisons with 222
concepts 222
customizing output 209
differences report 239
duplicate ID values 221
equality criterion 224
ID variables 220, 224, 249
information provided by 208
listing variables for matching observations 220
log and 226
macro return codes 227
ODS table names 236
output 228
output data set 237
output statistics data set 238
position of observations 223
restricting comparisons 221
results 226
syntax 211
task tables 211, 212
variable formats 226
COMPAREREG1 option
PROC REGISTRY statement 965
COMPAREREG2 option
PROC REGISTRY statement 965
COMPARETO= option
PROC REGISTRY statement 965
comparison data set 208
Compatibility Calculator 684
compiled functions and subroutines
location of 448
COMPLETECOLS option
PROC REPORT statement 1009
COMPLETEROWS option
PROC REPORT statement 1009
COMPLETETYPES option
PROC MEANS statement 615
composite indexes 1218
compound names 996
compressed data sets
appending 306
computational code blocks
declaring for subroutines 430
compute blocks 992
contents of 992
processing 994
referencing report items in 993
starting 1033
COMPUTE statement
REPORT procedure 1033
COMPUTE window
REPORT procedure 1056
COMPUTED option
DEFINE statement (REPORT) 1038
COMPUTED VAR window
REPORT procedure 1056
computed variables 989, 1038
storing 1122
concatenating catalogs 145
concatenating data sets 386
CONDENSE option
TABLE statement (TABULATE) 1382
confidence limits 642, 662
keywords and formulas 1543
one-sided, above the mean 1544
one-sided, below the mean 1543
TABULATE procedure 1364
two-sided 1543
CONNECT statement
SQL procedure 1217
CONNECTION TO component 1255
CONSTDATETIME option
PROC SQL statement 1205
CONSTRAINT= option
COPY statement (DATASETS) 322
PROC CPORT statement 273
CONTAINS condition 1255, 1294
CONTENTS= option
PROC PRINT statement 820
PROC REPORT statement 1009, 1023, 1038, 10
PROC TABULATE statement 1365
TABLE statement (TABULATE) 1382
CONTENTS procedure 259, 260
syntax 260
task table 260
versus CONTENTS statement (DATASETS) 318
CONTENTS statement
CATALOG procedure 135
DATASETS procedure 314
contingency tables 1451
continuation messages 1361
CONTOUR= option
PLOT statement (PLOT) 729
contour plots 729, 755
converting files 191, 269
COPY procedure 261, 262
concepts 262
syntax 262
transporting data sets 262
versus COPY statement (DATASETS) 327
COPY statement
CATALOG procedure 137
DATASETS procedure 319
TRANSPOSE procedure 1508
copying data libraries
entire data library 324
copying data sets
between hosts 263
block I/O method 323

long variable names 326
copying files 319
COPY statement versus COPY procedure 327
excluding files 333
member type 324
password-protected files 326
selected files 324, 352
copying views 325
corrected sum of squares 1538
CORRECTENCODING= option
MODIFY statement (DATASETS) 343
correlated subqueries 1283
CORRESPONDING keyword 1272
COUNT(*) function 1286
CPERCENT option
CHART procedure 169
CPM procedure 63, 114
CPORT procedure 270
concepts 278
Data Control Blocks 279
file transport process 192, 270
overview 269
password-protected data sets 278
results 279
syntax 270
task table 271
CREATE INDEX statement
SQL procedure 1218
CREATE TABLE statement
SQL procedure 1219
CREATE VIEW statement
SQL procedure 1224
CRITERION= option
PROC COMPARE statement 214, 224
cross joins 1263
crosstabulation tables 1451
CSS keyword 1538
cumulative distribution function 1545
customizing charts 168
CV keyword 1538

D

DANISH option
PROC SORT statement 1169
DATA= argument
PROC EXPORT statement 410
DATA COLUMNS window
REPORT procedure 1056
Data Control Blocks (DCBs) 279
data encryption 591, 1158
data files
migrating 687, 689
data libraries
copying entire library 324
copying files 319
deleting files 328
exchanging filenames 332
importing 203
printing directories of 259, 314
processing all data sets in 31
renaming files 313
saving files from deletion 351
USER data library 18
DATA= option
APPEND statement (DATASETS) 303
CONTENTS statement (DATASETS) 315
PROC CALENDAR statement 66
PROC CHART statement 161
PROC COMPARE statement 214
PROC MEANS statement 615
PROC OPTLOAD statement 716
PROC PLOT statement 723
PROC PRINT statement 820
PROC PRTDEF statement 922
PROC RANK statement 946
PROC REPORT statement 1010
PROC SORT statement 1173
PROC STANDARD statement 1339
PROC TABULATE statement 1365
PROC TIMEPLOT statement 1480
PROC TRANSPOSE statement 1504
DATA SELECTION window
REPORT procedure 1057
data set labels
changing 345
data set options 19
data sets
aging 388
appending 302
appending compressed data sets 306
appending indexed data sets 306
appending password-protected data sets 305
concatenating 386
content descriptions 314
contents of 259
copying between hosts 263
creating formats from 557
describing 384
exporting 281
input data sets 20
loading system options from 715
long variable names 326
migrating 684
migrating, containing non-English characters 690
migrating, with NODUPKEY sort indicator 689
modifying 382
naming 18
permanent 18
printing all data sets in library 882
printing formatted values for 26
processing all data sets in a library 31
reading and writing arrays to 439
removing all labels and formats 372
renaming variables 349
repairing 349
saving system option settings in 717
sort indicator information 392
sorting 1166
standardizing variables 1335
temporary 18
transporting 262, 328
transporting password-protected 278
USER data library and 18
writing printer attributes to 936
data sets, comparing
base data set 208
comparison data set 208
comparison summary 228
variables in different data sets 244
variables in same data set 222, 247

DATA step
  calling DIR_ENTRIES from 447
  compared with FCMP procedure 435
  terminating 424
DATA step debugger
  DATA step versus FCMP procedure 436
DATA step views
  migrating 688
  SQL procedure 1199
data summaries 1286, 1427
data summarization tools 1351
DATAFILE= argument
  PROC IMPORT statement 596
DATAROW= statement
  IMPORT procedure 598
DATASETS procedure 288, 293
  concepts 353
  directory listings, as output 360
  directory listings, to log 359
  ending 355
  error handling 355
  execution of statements 353
  forcing RUN-group processing 355
  generation data sets 358
  ODS and 364, 389
  output 289
  output data sets 366
  password errors 355
  passwords with 355
  procedure output 360
  restricting member types 356
  results 359
  RUN-group processing 353
  syntax 293
  task tables 293, 297, 314, 342
DATATYPE= option
  PICTURE statement (FORMAT) 521
date formats 552
DATECOPY option
  PROC CPORT statement 273
  PROC SORT statement 1174
  COPY statement (DATASETS) 322
DATETIME option
  PROC CALENDAR statement 66
DAYLENGTH= option
  PROC CALENDAR statement 66
DBMS
  SORT procedure with 1184
DBMS connections
  ending 1228
  sending DBMS statements to 1229
  SQL procedure 1217
  storing in views 1225
DBMS= option
  PROC EXPORT statement 411
  PROC IMPORT statement 597
DBMS queries 1255
DCBs (Data Control Blocks) 279
DDNAME= option
  PROC DATASETS statement 299
debugging
  registry debugging 966
DEBUGOFF option
  PROC REGISTRY statement 966
DEBUGON option
  PROC REGISTRY statement 966
DECSEP= option
  PICTURE statement (FORMAT) 522
DEFAULT= option
  FORMAT procedure 534
  RADIOBOX statement (PMENU) 787
DEFINE option
  PROC OPTIONS statement 708
DEFINE statement
  REPORT procedure 1035
DEFINITION window
  REPORT procedure 1057
DELETE option
  PROC PRTDEF statement 922
DELETE statement
  CATALOG procedure 138
  DATASETS procedure 328
  SQL procedure 1226
delimited files
  exporting 412
  importing 599
DELIMITER= option
  PROC TRANSPOSE statement 1505
DELIMITER= statement
  IMPORT procedure 598
denominator definitions 1451
density function 1545
DESC option
  PROC PMENU statement 779
DESCENDING option
  BY statement 36
  BY statement (CALENDAR) 72
  BY statement (CHART) 164
  BY statement (COMPARE) 219
  BY statement (MEANS) 622
  BY statement (PLOT) 726
  BY statement (PRINT) 828
  BY statement (RANK) 950
  BY statement (REPORT) 1027
  BY statement (SORT) 1179
  BY statement (STANDARD) 1341
  BY statement (TABULATE) 1374
  BY statement (TIMEPLOT) 1481
  BY statement (TRANSPOSE) 1506
  CHART procedure 169
  CLASS statement (MEANS) 623
  CLASS statement (TABULATE) 1375
  DEFINE statement (REPORT) 1039
  ID statement (COMPARE) 220
  KEY statement (SORT) 1181
  PROC RANK statement 947
DESCENDTYPES option
  PROC MEANS statement 615
DESCRIBE statement
  SQL procedure 1227
DESCRIPTION= argument
  MODIFY statement (CATALOG) 141
descriptive statistics 646, 1351
  computing with class variables 648
  keywords and formulas 1538
  table of 32
DET CALL routine 454
detail reports 981
detail rows 981
DETAILS option
  CONTENTS statement (DATASETS) 316
  PROC DATASETS statement 297

determinant of a matrix 454
deviation from the mean 1551
dialog boxes 781
  check boxes in 780
  collecting user input 797
  color for 791
  input fields 791
  radio buttons in 788
  searching multiple values 800
  text for 791
DIALOG statement
  PMENU procedure 781
DICTIONARY tables
  reporting from 1307
difference 226
  report of differences 239
DIG3SEP= option
  PICTURE statement (FORMAT) 522
digit selectors 524
dimension expressions 1386
  elements in 1386
  operators in 1388
  style elements in 1388
directives 525
directories
  calling DIR_ENTRIES from DATA step 447
  gathering filenames 446
  opening and closing 446
DIRECTORY option
  CONTENTS statement (DATASETS) 316
directory transversal 445, 446
DIR_ENTRIES
  calling from DATA step 447
DISCONNECT statement
  SQL procedure 1228
DISCRETE option
  CHART procedure 170
DISPLAY option
  DEFINE statement (REPORT) 1039
DISPLAY PAGE window
  REPORT procedure 1063
DISPLAY procedure 401
  overview 401
  syntax 401
display variables 987, 1039
distribution 1545
DMOPTLOAD command 715
DMOPTSAVE command 717
DO statement
  DATA step versus FCMP procedure 436
DOL option
  BREAK statement (REPORT) 1024
  RBREAK statement (REPORT) 1049
DOUBLE option
  PROC PRINT statement 820
  PROC SQL statement 1206
double overlining 1024, 1049
double underlining 1024, 1049
DQUOTE= option
  PROC SQL statement 1206
DROP statement
  SQL procedure 1228
DTC= option
  MODIFY statement (DATASETS) 343
DUL option
  RBREAK statement (REPORT) 1024, 1049
DUPOUT= option
  PROC SORT statement 1174
DUR statement
  CALENDAR procedure 74
DYNAMIC_ARRAY subroutine 471

E

EBCDIC collating sequence 1169, 1182
EBCDIC option
  PROC SORT statement 1169
EET= option
  PROC CIMPORT statement 194
  PROC CPORT statement 273
efficiency
  statistical procedures 7
elementary statistics procedures 1535
ELEMMULT CALL routine 455
embedded LIBNAME statements 1225
embedded SQL 1296
encoded passwords 937, 939
  encoding methods 938, 942
  in SAS programs 938, 940
  saving to paste buffer 941
encoding
  versus encryption 939
encoding methods 938, 942
encoding values 1171
ENCRYPT option
  PROC FCMP statement 423
encryption 591, 1158
  versus encoding 939
ENDCOMP statement
  REPORT procedure 1045
ENTRYTYPE= option
  CATALOG procedure 144
  CHANGE statement (CATALOG) 135
  COPY statement (CATALOG) 137
  DELETE statement (CATALOG) 139
  EXCHANGE statement (CATALOG) 139
  EXCLUDE statement (CATALOG) 140
  EXCLUDE statement (CIMPORT) 196
  EXCLUDE statement (CPORT) 276
  MODIFY statement (CATALOG) 141
  PROC CATALOG statement 134
  SAVE statement (CATALOG) 141
  SELECT statement (CATALOG) 142
  SELECT statement (CIMPORT) 197
  SELECT statement (CPORT) 277
enumerations 913
ENVELOPE property
  PROC SOAP statement 1157
EQUALS option
  PROC SORT statement 1174
equijoins 1259
error checking
  formats and 31
error handling
  CATALOG procedure 143
ERROR option
  PROC COMPARE statement 214
error processing
  of BY-group specifications 25
ERRORSTOP option
  PROC SQL statement 1206
estimates 1545

ET= option
  PROC CIMPORT statement 194
  PROC CPORT statement 273
ETYPE= option
  SELECT statement (CPORT) 277
event logging 311
Excel
  importing spreadsheet from workbook 602
  importing subset of records from 603
EXCEPT operator 1275
EXCHANGE statement
  CATALOG procedure 139
  DATASETS procedure 332
EXCLNPWGT option
  PROC REPORT statement 1010
  PROC STANDARD statement 1339
EXCLNPWGTS option
  PROC MEANS statement 615
  PROC TABULATE statement 1365
EXCLUDE statement
  CATALOG procedure 140
  CIMPORT procedure 196
  CPORT procedure 276
  DATASETS procedure 333
  FORMAT procedure 516
  PRTEXP procedure 934
EXCLUSIVE option
  CLASS statement (MEANS) 623
  CLASS statement (TABULATE) 1375
  DEFINE statement (REPORT) 1040
  PROC MEANS statement 615
  PROC TABULATE statement 1365
EXEC option
  PROC SQL statement 1206
EXECUTE statement
  SQL procedure 1229
EXISTS condition 1256
EXITCODE option
  PROC SQL statement 1207
expected value 1545
EXPLODE 409
EXPLODE procedure 409
EXPLORE window
  REPORT procedure 1063
EXPMATRIX CALL routine 456
EXPORT= option
  PROC REGISTRY statement 966
EXPORT procedure 410
  DBMS specifications 411
  overview 409
  syntax 410
exporting
  catalog entries 280, 283
  CPORT procedure 269
  excluding files or entries 276
  multiple catalogs 279
  printer definitions 923
  registry contents 966, 972
  selecting files or entries 277
exporting data 409
  delimited files 412
EXTENDSN= option
  PROC CIMPORT statement 194
external C functions 910
external files
  browsing 577
  comparing registry with 973
  routing output or log to 892
extreme values 669, 672

F

FAT file system 691
FCmp Function Editor 438, 477
  closing functions 481
  creating functions 483
  Data Explorer 488
  deleting functions 482
  duplicating functions 481
  exporting functions to a file 482
  Function Browser 486
  Log window 485
  moving functions 480
  opening 477
  opening functions 479
  opening multiple functions 480
  printing functions 483
  renaming functions 482
  using functions in DATA step 488
  working with functions 479
FCMP procedure 420, 421
  additional features 438
  arrays and 438
  C helper functions and CALL routines 463
  calling functions from DATA step 488
  compared with DATA step 435
  computing implicit values of a function 438
  concepts 432
  creating CALL routine and a function 490
  creating functions 488
  creating functions and subroutines 432
  DATA step differences 436
  directory transversal 445, 446
  executing STANDARDIZE on each row of a data set 491
  FCmp Function Editor 438, 477
  functions for calling code from within functions 472
  location of compiled functions and subroutines 448
  macros with routines 442
  Microsoft Excel and 438
  passing arrays 438
  reading and writing arrays to a data set 439
  recursion 443
  REPORT procedure and compute blocks 438
  special functions and CALL routines 451, 467
  syntax 421
  task tables 421, 422
  user-defined functions 433
  user-defined functions with GTL 493
  variable scope 442
FEEDBACK option
  PROC SQL statement 1207
file allocation table (FAT) file system 691
file extensions
  migrating short-extension files 691
FILE= option
  CONTENTS statement (CATALOG) 136
  PROC CIMPORT statement 195
  PROC CPORT statement 273
file transport process 192, 270
filenames
  gathering 446

filerefs
executing SAS code in specified fileref 476
files
aging 300
converting 191, 269
copying 261, 319
deleting 328
exchanging names 332
excluding from copying 333
manipulating 374
modifying attributes 342
moving 324
renaming 313
renaming groups of 300
saving from deletion 351, 380
selecting for copying 352
FILL option
PROC CALENDAR statement 67
PICTURE statement (FORMAT) 522
FILLMATRIX CALL routine 457
FIN statement
CALENDAR procedure 75
Finnish collating sequence 1169
FINNISH option
PROC SORT statement 1169
floating point exception (FPE) recovery 1372
FLOW option
DEFINE statement (REPORT) 1040
PROC FCMP statement 423
PROC SQL statement 1207
FMTLEN option
CONTENTS statement (DATASETS) 316
FMTLIB option
PROC FORMAT statement 515, 516, 530
font files
adding 506, 507
searching directories for 501
specifying 500
TrueType 503, 508
Type 1 504
FONTFILE statement
FONTREG procedure 500
FONTPATH statement
FONTREG procedure 501
FONTREG procedure 498
concepts 504
font naming conventions 504
overview 497
removing fonts from registry 505
supported font types 504
syntax 498
fonts
naming conventions 504
removing from registry 505
FORCE option
APPEND statement (DATASETS) 303
COPY statement (DATASETS) 322
PROC CATALOG statement 134, 153
PROC CIMPORT statement 195
PROC DATASETS statement 298
PROC SORT statement 1174
FOREIGN option
PROC PRTDEF statement 923
format catalogs 539
format-name formats 534
FORMAT= option
ATTRIB statement (FCMP) 426
DEFINE statement (REPORT) 1040
MEAN statement (CALENDAR) 78
PROC TABULATE statement 1365
SUM statement (CALENDAR) 81
FORMAT procedure 512, 514
associating informats and formats with variables 538
concepts 538
excluding entries from processing 516
input control data set 543
options 534
output control data set 541
printing informats and formats 540
procedure output 544
ranges 536
results 541
selecting entries for processing 530
storing informats and formats 539
syntax 514
task tables 514, 517, 520, 532
values 536
FORMAT statement 35
DATASETS procedure 334
FORMAT_PRECEDENCE= option
TABLE statement (TABULATE) 1382
formats 512
See also picture formats
assigning style attribute values 1000
assigning style attributes 1400
associating with variables 512, 538
BY-group processing and 31
comparing unformatted values 226
creating from data sets 557
creating groups with 1126
creating in non-English languages 570
date formats 552
error checking and 31
for character values 532, 549
for columns 1253
format-name formats 534
managing with DATASETS procedure 334
missing 540
multilabel 1423
multilabel value formats 655
permanent 539
picture-name formats 530
preloaded 1042, 1376, 1418
preloaded, with class variables 659
printing 540
printing descriptions of 561
ranges for character strings 566
removing from data sets 372
retrieving permanent formats 563
specifying information for variables 426
storing 539
temporarily associating with variables 29
temporarily dissociating from variables 30
temporary 539
FORMATS window
REPORT procedure 1064
formatted values 26, 37
classifying formatted data 28
grouping formatted data 28
printing 26

FORMCHAR option
  PROC CHART statement 161
  PROC PLOT statement 724
  PROC REPORT statement 1010
  PROC TABULATE statement 1365
  PROC CALENDAR statement 67
forms
  printing reports with 1001
FORMS 575
FORMS procedure 575
formulas
  for statistics 1536
FORTCC option
  FSLIST command 582
  PROC FSLIST statement 579
FPE recovery 1372
FRACTION option
  PROC RANK statement 947
FRAME applications
  associating menus with 812
FreeType fonts 497
FREQ option
  CHART procedure 170
  CHART procedure 170
FREQ statement 39
  example 40
  MEANS procedure 626
  procedures supporting 40
  REPORT procedure 1045
  STANDARD procedure 1341
  TABULATE procedure 1379
frequency counts
  CHART procedure 175
  displaying with denominator definitions 1451
  TABULATE procedure 1451
frequency of observations 39
FROM clause
  SQL procedure 1239
FSEDIT applications
  menu bars for 794
FSEDIT sessions
  associating menu bar with 796
FSLIST command 578, 580
FSLIST procedure 577
  statement descriptions 578
  syntax 577
  task table 578
FSLIST window 582
  commands 582
  display commands 587
  global commands 582
  scrolling commands 583
  searching commands 585
FULLSTATUS option
  PROC REGISTRY statement 966
Function Compiler (FCMP)
  See FCMP procedure
function prototypes
  registering 907
FUNCTION statement
  FCMP procedure 427
functional categories of procedures 3
functions
  See also FCmp Function Editor
  See also FCMP procedure
  C helper 463
  changing array size within 471
  computing implicit values of 438, 467
  creating with FCMP procedure 432
  declaring 434
  FCMP procedure 1295
  for calling code from within functions 472
  location of compiled 448
  special 451
  sql-expression and 1278
  SQL procedure and 1295
  user-defined 433, 493
functions, C or C++
  See PROTO procedure
FUZZ= option
  FORMAT procedure 535
  PROC COMPARE statement 214
  TABLE statement (TABULATE) 1382
FW= option
  PROC MEANS statement 616

G

G100 option
  CHART procedure 170
Garman-Kohlhagen implied volatility 469
Gaussian distribution 1552
generation data sets
  DATASETS procedure and 358
generation groups
  appending with 308
  changing number of 346
  copying 327
  deleting 329
  removing passwords 347
GENERATION option
  PROC CPORT statement 273
generations
  migrating data files with 689
GENMAX= option
  MODIFY statement (DATASETS) 343
GENNUM= data set option 308
GENNUM= option
  AUDIT statement (DATASETS) 311
  CHANGE statement (DATASETS) 313
  DELETE statement (DATASETS) 328
  MODIFY statement (DATASETS) 344
  PROC DATASETS statement 298
  REBUILD statement (DATASETS) 347
  REPAIR statement (DATASETS) 350
GETNAMES= statement
  IMPORT procedure 598
GETSORT option
  APPEND statement (DATASETS) 303
Ghostview printer definition 928
global statements 20
GRAY option
  ITEM statement (PMENU) 784
grayed items 784
Grid Job Generator 1149
GRID option
  RECORD statement (SCAPROC) 1145
GRID statement 1143
GROUP BY clause
  SQL procedure 1241
GROUP option
  DEFINE statement (REPORT) 1040

CHART procedure 170
PROC OPTIONS statement 708
group variables 988, 1040
grouping formatted data 28
GROUPINTERNAL option
CLASS statement (MEANS) 623
CLASS statement (TABULATE) 1375
groups
creating with formats 1126
GROUPS= option
PROC RANK statement 947
GSPACE= option
CHART procedure 170
GTL
user-defined functions with 493
GUESSING ROWS= statement
IMPORT procedure 599

H

HAVING clause
SQL procedure 1242
HAXIS= option
PLOT statement (PLOT) 729
HBAR statement
CHART procedure 165
HEADER= option
PROC CALENDAR statement 69
headers
response headers 594
HEADING= option
PROC PRINT statement 820
HEADLINE option
PROC REPORT statement 1012
HEADSKIP option
PROC REPORT statement 1012
HELP option
PROC JAVAINFO statement 608
ITEM statement (PMENU) 784
PROC REPORT statement 1012
HEXPAND option
PLOT statement (PLOT) 731
HEXVALUE option
PROC OPTIONS statement 708
hidden label characters 742
hidden observations 744
HIDE option
PROC FCMP statement 423
HILOC option
PLOT statement (TIMEPLOT) 1486
HOLIDATA= option
PROC CALENDAR statement 69
holidays data set 69, 88
multiple calendars 85, 86
HOLIDUR statement
CALENDAR procedure 75
HOLIFIN statement
CALENDAR procedure 76
HOLISTART statement
CALENDAR procedure 77
HOLIVAR statement
CALENDAR procedure 77
horizontal bar charts 156, 165
for subset of data 185
horizontal separators 1432
HOST option
PROC OPTIONS statement 708
host-specific procedures 1571
HPERCENT= option
PROC PLOT statement 724
HPOS= option
PLOT statement (PLOT) 731
HREF= option
PLOT statement (PLOT) 732
HREFCHAR= option
PLOT statement (PLOT) 732
HREVERSE option
PLOT statement (PLOT) 732
HSCROLL= option
PROC FSLIST statement 580
HSPACE= option
PLOT statement (PLOT) 732
HTML files
style elements 1129
TABULATE procedure 1465, 1470
HTML reports 838
HTTP procedure 589, 590
capturing response headers 594
POST request through proxy 593
POST request through proxy and authentication 593
simple POST request 592
syntax 590
HTTP requests 589
HTTPS protocol 591
making PROC HTTP calls with 592
making SOAP procedure calls with 1158
Hypertext Transfer Protocol Secure (HTTPS) 591
hypotheses
keywords and formulas 1543
testing 1565
HZERO option
PLOT statement (PLOT) 732

I

IC CREATE statement
DATASETS procedure 334
IC DELETE statement
DATASETS procedure 337
IC REACTIVATE statement
DATASETS procedure 337
ID option
DEFINE statement (REPORT) 1040
ITEM statement (PMENU) 785
ID statement
COMPARE procedure 220
MEANS procedure 627
PRINT procedure 829
TIMEPLOT procedure 1482
TRANSPOSE procedure 1508
ID variables 1040
COMPARE procedure 220
IDENTIFY CALL routine 457
IDLABEL statement
TRANSPOSE procedure 1509
IDMIN option
PROC MEANS statement 616
IF expressions
DATA step versus FCMP procedure 436
implicit values
of functions 467

IMPORT= option
  PROC REGISTRY statement 966
IMPORT procedure 596
  data source statements 597
  overview 595
  syntax 596
importing
  catalog entries 204
  CIMPORT procedure 191
  data libraries 203
  excluding files or entries 196
  indexed data sets 205
  selecting files or entries 197
  to registry 966, 971
importing data 595
  delimited files 599
  Microsoft Access 604
  spreadsheet from Excel workbook 602
  subset of records from Excel 603
IN= argument
  PROC MIGRATE statement 686
IN condition 1257
in-line views 1240, 1294
  querying 1319
IN= option
  COPY statement (CATALOG) 137
  COPY statement (DATASETS) 319
  PROC SOAP statement 1156
INDENT= option
  TABLE statement (TABULATE) 1382
indenting row headings 1432
INDEX CENTILES statement
  DATASETS procedure 338
INDEX CREATE statement
  DATASETS procedure 339
INDEX DELETE statement
  DATASETS procedure 340
INDEX= option
  COPY statement (DATASETS) 322
  PROC CPORT statement 274
indexed data sets
  importing 205
indexes
  appending indexed data sets 306
  centiles for indexed variables 338
  composite indexes 1218
  creating 339
  deleting 340, 1228
  managing 1219
  migrating data files with 689
  on altered columns 1216
  on columns 1218, 1232
  restoring or deleting when disabled 347
  simple indexes 1218
  SQL procedure 1218
  UNIQUE keyword 1218
INFILE= option
  PROC CIMPORT statement 195
INFORMAT statement
  DATASETS procedure 341
informats 512
  associating with variables 512, 538
  converting raw character data to numeric values 554
  for columns 1252
  managing with DATASETS procedure 341
  missing 540
  permanent 539
  printing 540
  printing descriptions of 561
  raw data values 517
  storing 539
  temporary 539
INITIATE argument
  AUDIT statement (DATASETS) 311
INLIB= option
  PROC FCMP statement 423
inner joins 1260
INOBS= option
  PROC SQL statement 1207
input data sets 20
  CALENDAR procedure 86
  presorted 1184
input fields 791
input files
  procedure output as 898
INSERT statement
  SQL procedure 1231
INT= argument
  MAPMISS statement (PROTO) 906
integrity constraints
  appending data sets and 308
  copying data sets and 322
  creating 334
  deleting 337
  migrating data files with 689
  names for 336
  PROC SQL tables 1217, 1223
  reactivating 337
  restoring or deleting when disabled 347
  SORT procedure 1185
interactive line mode
  printing from 1002
interquartile range 1551
INTERSECT operator 1277
INTERVAL= option
  PROC CALENDAR statement 70
INTO clause
  SQL procedure 1235
INTYPE= option
  PROC CPORT statement 274
INV CALL routine 458
INVALUE statement
  FORMAT procedure 517
IPASSTHRU option
  PROC SQL statement 1207
IS condition 1257
ISNULL C helper function 463, 916
ITEM statement
  PMENU procedure 783
item stores
  migrating 689
ITEMHELP= option
  DEFINE statement (REPORT) 1040

J

Java environment 607
JAVAINFO procedure 607
jobs
  terminating 424
joined-table component 1258

JOINREF option
  PLOT statement (TIMEPLOT) 1486
joins 1259
  cross joins 1263
  equijoins 1259
  inner joins 1260
  joining a table with itself 1259
  joining more than two tables 1266
  joining three tables 1316
  joining two tables 1303
  natural joins 1265
  outer joins 1262, 1294, 1309
  reflexive joins 1259
  rows to be returned 1259
  subqueries compared with 1268
  table limit 1259
  types of 1259
  union joins 1264
JREOPTIONS option
  PROC JAVAINFO statement 608
JUST option
  INVALUE statement (FORMAT) 518

K

KEEPLEN option
  OUTPUT statement (MEANS) 632
KEEPNODUPKEY option
  PROC MIGRATE statement 687, 689
KEY= option
  PROC OPTLOAD statement 716
  PROC OPTSAVE statement 718
key sequences 784
KEY statement
  SORT procedure 1180
KEYLABEL statement
  TABULATE procedure 1379
keyword headings
  style elements for 1380
KEYWORD statement
  TABULATE procedure 1380
keywords
  for statistics 1536
KILL option
  PROC CATALOG statement 134, 153
  PROC DATASETS statement 298
kurtosis 1552
KURTOSIS keyword 1538

L

LABEL option
  PROC PRINT statement 820, 826
  ATTRIB statement (FCMP) 426
  MODIFY statement (DATASETS) 344
  PROC PRINTTO statement 889
  PROC TRANSPOSE statement 1505
LABEL statement 35
  DATASETS procedure 341
  FCMP procedure 429
labels
  for columns 1253
  hidden label characters 742
  on plots 761, 766, 770
  removing from data sets 372
  specifying, up to 256 characters 429
  specifying information for variables 426
language concepts 18
  data set options 19
  global statements 20
  system options 18
  temporary and permanent data sets 18
LANGUAGE option
  PICTURE statement (FORMAT) 522
LCLM keyword 1543
LEFT option
  DEFINE statement (REPORT) 1041
LEGEND option
  PROC CALENDAR statement 70
length
  specifying information for variables 426
LENGTH= option
  ATTRIB statement (FCMP) 426
LET option
  PROC TRANSPOSE statement 1505
LEVELS option
  OUTPUT statement (MEANS) 633
  CHART procedure 170
LIBNAME statement
  embedding in views 1225
libraries
  migrating members 683
  migrating SAS 6 libraries 690
  printing all data sets 882
  validation tools for migrating 684, 693
LIBRARY= option
  PROC DATASETS statement 299
  PROC FCMP statement 423
  PROC FORMAT statement 515
librefs
  stored views and 1224
LIKE condition 1268
  patterns for searching 1269
  searching for literals 1269
  searching for mixed-case strings 1270
line-drawing characters 1008
LINE statement
  REPORT procedure 1046
LINK statement
  PROTO procedure 905
LIST option
  PLOT statement (PLOT) 732
  PROC FCMP statement 423
  PROC PRTDEF statement 923
  PROC REGISTRY statement 967
  PROC REPORT statement 1012
LISTALL option
  PROC COMPARE statement 214
  PROC FCMP statement 423
LISTBASE option
  PROC COMPARE statement 215
LISTBASEOBS option
  PROC COMPARE statement 215
LISTBASEVAR option
  PROC COMPARE statement 215
LISTCODE option
  PROC FCMP statement 423
LISTCOMP option
  PROC COMPARE statement 215
LISTCOMPOBS option
  PROC COMPARE statement 215

LISTCOMPVAR option
  PROC COMPARE statement 215
LISTEQUALVAR option
  PROC COMPARE statement 215
LISTHELP= option
  PROC REGISTRY statement 967
listing reports 816, 841
LISTOBS option
  PROC COMPARE statement 215
LISTPROG option
  PROC FCMP statement 423
LISTREG= option
  PROC REGISTRY statement 967
LISTSOURCE option
  PROC FCMP statement 424
LISTUSER option
  PROC REGISTRY statement 967
LISTVAR option
  PROC COMPARE statement 215
load modules
  name and path of 905
LOAD REPORT window
  REPORT procedure 1065
LOCALE option
  PROC CALENDAR statement 70
LOCKCAT= option
  COPY statement (CATALOG) 137
log
  COMPARE procedure results 226
  default destinations 887
  destinations for 887
  displaying SQL definitions 1227
  listing registry contents in 967
  routing to catalog entries 895
  routing to external files 892
  routing to printer 892, 901
  writing printer attributes to 935
  writing registry contents to 967
LOG option
  AUDIT statement (DATASETS) 311
  PROC PRINTTO statement 889
logarithmic scale for plots 752
LONG option
  PROC OPTIONS statement 709
long variable names
  copying data sets with 326
LOOPS= option
  PROC SQL statement 1207
LOWER function (SQL) 1270
LPI= option
  PROC CHART statement 163
LS= option
  PROC REPORT statement 1013

M

macro return codes
  COMPARE procedure 227
macros
  adjusting plot labels 770
  counting missing values 1331
  executing predefined SAS macros 472
  FCMP procedure routines with 442
MAPMISS statement
  PROTO procedure 906
markers 1017, 1369
matching observations 208
matching patterns 1268, 1328
matching variables 208
matrices
  adding matrix and scalar 452
  adding two 452
  Cholesky decomposition for symmetric matrices 453
  converting input matrix to identity matrix 457
  determinant of 454
  inverse of 458
  multiplicative product of 459
  multiplying 455
  raising scalar value 460
  replacing element values with 0 462
  subtraction of 461
  transpose of 462
matrix CALL routines 451
MAX keyword 1539
MAX= option
  FORMAT procedure 535
MAXDEC= option
  PROC MEANS statement 616
  PROC TIMEPLOT statement 1480
maximum value 1539
MAXLABLEN= option
  PROC FORMAT statement 516
MAXPRINT= option
  PROC COMPARE statement 215
MAXSELEN= option
  PROC FORMAT statement 516
MDDBs
  migrating 689
mean 1545, 1546
MEAN keyword 1539
MEAN option
  CHART procedure 170
  PROC STANDARD statement 1339
MEAN statement
  CALENDAR procedure 78
MEANS procedure 610, 612
  class variables 638
  column width for output 644
  computational resources 639
  computer resources 626
  concepts 637
  descriptive statistics 646, 648
  missing values 625, 644, 667
  N Obs statistic 644
  output 610
  output data set 645
  output statistics 663, 665, 667, 669,
  results 644
  statistic keywords 619, 628
  statistical computations 641
  syntax 612
  task tables 612, 613
MEANTYPE= option
  PROC CALENDAR statement 70
measures of location 1546
measures of shape 1551
measures of variability 1550
median 1546
MEDIAN keyword 1541
member types
  migration and 687

MEMTYPE= option
AGE statement (DATASETS) 300
CHANGE statement (DATASETS) 313
CONTENTS statement (DATASETS) 316
COPY statement (DATASETS) 322, 324
DELETE statement (DATASETS) 329
EXCHANGE statement (DATASETS) 332
EXCLUDE statement (CIMPORT) 197
EXCLUDE statement (CPORT) 276
EXCLUDE statement (DATASETS) 333
MODIFY statement (DATASETS) 344
PROC CIMPORT statement 195
PROC CPORT statement 274
PROC DATASETS statement 299
REBUILD statement (DATASETS) 347
REPAIR statement (DATASETS) 350
SAVE statement (DATASETS) 351
SELECT statement (CIMPORT) 198
SELECT statement (CPORT) 277
SELECT statement (DATASETS) 352
menu bars 777
associating with FSEDIT sessions 796, 803
associating with FSEDIT window 799
defining items 785
for FSEDIT applications 794
items in 783
key sequences for 784
menu items 783
MENU statement
PMENU procedure 786
merging data
SQL procedure 1288
message characters 524
MESSAGE= option
IC CREATE statement (DATASETS) 336
MESSAGES window
REPORT procedure 1065
METHOD= option
PROC COMPARE statement 215
PROC PWENCODE statement 938
Microsoft Access
importing tables 604
Microsoft Excel
FCMP procedure and 438
MIDPOINTS= option
CHART procedure 171
MIGRATE procedure 683, 685
across computers, with SLIBREF= option 696
across computers, without SLIBREF= option 695
alternatives to 700
best practices 684
catalogs 689
concepts 687
data files 687, 689
data sets 684
data sets, containing non-English characters 690
data sets, with NODUPKEY sort indicator 689
item stores 689
MDDBs 689
member types 687
on same computer, with SLIBREF= option 698
on same computer, without SLIBREF= option 698
program files 689
SAS 6 libraries 690
short-extension files 691
syntax 685
unsupported catalogs 699
validation tools 684, 693
views 687
MIN keyword 1539
MIN= option
FORMAT procedure 535
minimum value 1539
missing informats and formats 540
MISSING option
CHART procedure 171
CLASS statement (MEANS) 623
CLASS statement (TABULATE) 1375
DEFINE statement (REPORT) 1041
PROC CALENDAR statement 71
PROC MEANS statement 616
PROC PLOT statement 725
PROC REPORT statement 1013
PROC TABULATE statement 1367
missing values
CALENDAR procedure 91
charts 171, 174
counting with a macro 1331
MEANS procedure 625, 644, 667
NMISS keyword 1539
PLOT procedure 744, 764
PROTO procedure 906, 910
RANK procedure 954
REPORT procedure 994, 1117
SQL procedure 1257, 1331
STANDARD procedure 1343
TABULATE procedure 1377, 1401
TIMEPLOT procedure 1488
TRANSPOSE procedure 1509
MISSTEXT= option
TABLE statement (TABULATE) 1382
MLF option
CLASS statement (MEANS) 624
CLASS statement (TABULATE) 1375
MNEMONIC= option
ITEM statement (PMENU) 785
mode 1546
MODE keyword 1539
MODE= option
PROC FONTREG statement 499
MODIFY statement
CATALOG procedure 141
DATASETS procedure 342
moment statistics 641
MOVE option
COPY statement (CATALOG) 137
COPY statement (DATASETS) 323
PROC MIGRATE statement 686, 693
moving files 324
MSGLEVEL= option
PROC FONTREG statement 499
MT= option
PROC CPORT statement 274
MTYPE= option
EXCLUDE statement (CPORT) 276
SELECT statement (CPORT) 277
MULT CALL routine 459
multi-threaded sorting 1181
multilabel formats 1423
MULTILABEL option
PICTURE statement (FORMAT) 522
VALUE statement (FORMAT) 532

multilabel value formats 655
multipage tables 1434
multiple-choice survey data 1441
multiple-response survey data 1436
MULTIPLIER= option
  PICTURE statement (FORMAT) 523
MUSTUNDERSTAND option
  PROC SOAP statement 1156

N

N keyword 1539
N Obs statistic 644
N option
  PROC PRINT statement 821
NAME= option
  PROC TRANSPOSE statement 1505
NAMED option
  PROC REPORT statement 1014
naming data sets 18
NATIONAL option
  PROC SORT statement 1169
National Use Differences 1169
natural joins 1265
NEDIT option
  PROC CPORT statement 275
nested variables 1362
NEW option
  COPY statement (CATALOG) 137
  PROC CIMPORT statement 195
  PROC PRINTTO statement 890
  APPEND statement (DATASETS) 303
NMISS keyword 1539
NOBORDER option
  PROC FSLIST statement 580
NOBS keyword 1540
NOBYLINE system option
  BY statement (MEANS) with 622
  BY statement (PRINT) with 828
NOCC option
  FSLIST command 582
  PROC FSLIST statement 579
NOCOMPRESS option
  PROC CPORT statement 275
NOCONTINUED option
  TABLE statement (TABULATE) 1383
NODATE option
  PROC COMPARE statement 216
NODS option
  CONTENTS statement (DATASETS) 317
NODUPKEY option
  PROC SORT statement 1175
NODUPRECS option
  PROC SORT statement 1175
NOEDIT option
  COPY statement (CATALOG) 138
  PICTURE statement (FORMAT) 523
  PROC CIMPORT statement 195
  PROC CPORT statement 275
NOEXEC option
  PROC REPORT statement 1014
NOHEADER option
  CHART procedure 171
  PROC REPORT statement 1014
NOINDEX option
  REBUILD statement (DATASETS) 347
NOINHERIT option
  OUTPUT statement (MEANS) 633
NOLEGEND option
  CHART procedure 171
  PROC PLOT statement 725
NOLIST option
  PROC DATASETS statement 299
NOMISS option
  INDEX CREATE statement (DATASETS) 339
  PROC PLOT statement 725
NOMISSBASE option
  PROC COMPARE statement 216
NOMISSCOMP option
  PROC COMPARE statement 216
NOMISSING option
  PROC COMPARE statement 216
NONE option
  RBUTTON statement (PMENU) 788
noninteractive mode
  printing from 1002
NONOBS option
  PROC MEANS statement 616
NOOBS option
  PROC PRINT statement 821
NOPRINT option
  CONTENTS statement (DATASETS) 317
  DEFINE statement (REPORT) 1041
  PROC COMPARE statement 216
  PROC SUMMARY statement 1352
NOREPLACE option
  PROC FORMAT statement 516
normal distribution 1545, 1552
NORMAL= option
  PROC RANK statement 947
NORWEGIAN option
  PROC SORT statement 1169
NOSEPS option
  PROC TABULATE procedure 1367
NOSOURCE option
  COPY statement (CATALOG) 138
NOSRC option
  PROC CIMPORT statement 196
  PROC CPORT statement 275
NOSTATS option
  CHART procedure 171
NOSUMMARY option
  PROC COMPARE statement 216
NOSYMBOL option
  CHART procedure 171
/NOSYMBOLS option
  ARRAY statement (FCMP) 425, 471
NOSYMNAME option
  PLOT statement (TIMEPLOT) 1486
NOTE option
  PROC COMPARE statement 216
NOTRAP option
  PROC MEANS statement 617
NOTSORTED option
  BY statement 36
  BY statement (CALENDAR) 72
  BY statement (CHART) 165
  BY statement (COMPARE) 219
  BY statement (MEANS) 622
  BY statement (PLOT) 726
  BY statement (PRINT) 828
  BY statement (RANK) 950

BY statement (REPORT) 1027
BY statement (STANDARD) 1341
BY statement (TABULATE) 1374
BY statement (TIMEPLOT) 1481
BY statement (TRANSPOSE) 1506
FORMAT procedure 535
ID statement (COMPARE) 220
NOUPDATE option
PROC FONTREG statement 499
NOVALUES option
PROC COMPARE statement 216
NOWARN option
APPEND statement (DATASETS) 304
PROC DATASETS statement 299
NOZERO option
DEFINE statement (REPORT) 1041
NOZEROS option
CHART procedure 171
NPLUS1 option
PROC RANK statement 948
NPP option
PLOT statement (TIMEPLOT) 1486
NSRC option
PROC CPORT statement 275
null hypothesis 1565
NUM option
PROC FSLIST statement 580
NUMBER option
PROC SQL statement 1208
numbers
template for printing 520
numeric data
in FUNCTION statement (FCMP) 428
numeric values
converting raw character data to 554
summing 856
numeric variables
PROTO procedure 909
sorting orders for 1182
summing 852
NWAY option
PROC MEANS statement 617

O

OBS= option
PROC PRINT statement 821
observations
appending 55
consolidating in reports 1097
frequency of 39
grouping for reports 845
hidden 744
maintaining order of, in BY groups 1191
page layout 833
partitioning based on ranks 959
retaining first observation of each BY group 1193
SQL procedure 1199
statistics for groups of 7
total number of 1540
transposing variables into 1501
weighting 637
observations, comparing
comparison summary 230, 235
matching observations 208
with ID variable 249
with output data set 253
ODS output
CALENDAR procedure 93
CHART procedure 174
style elements for 1129, 1134, 1465, 14
TABULATE procedure 1412
ODS (Output Delivery System)
DATASETS procedure and 364, 389
PLOT procedure and 743
printing reports 1001
TABULATE procedure and 1358, 1465, 1470
ODS table names
CHART procedure 174
COMPARE procedure 236
DATASETS procedure 365
PLOT procedure 743
TIMEPLOT procedure 1488
OL option
BREAK statement (REPORT) 1024
RBREAK statement (REPORT) 1049
ON option
CHECKBOX statement (PMENU) 780
one-tailed tests 1566
OPENTIMES option
RECORD statement (SCAPROC) 1145
operands
values from 1277
operating environment-specific procedures 31, 1571
operators
arithmetic 1295
in dimension expressions 1388
order of evaluation 1278
set operators 1271, 1295
truncated string comparison operators 1280
values from 1277
OPTION= option
PROC OPTIONS statement 709
OPTIONS procedure 707
display settings for a group of options 703
output 702
overview 701
results 710
syntax 707
task table 707
OPTLOAD procedure 716
overview 715
syntax 716
task table 716
OPTSAVE procedure 718
overview 717
syntax 718
task table 718
ORDER BY clause
SQL procedure 1243, 1294
ORDER option
DEFINE statement (REPORT) 1041
CLASS statement (MEANS) 624
CLASS statement (TABULATE) 1376
CONTENTS statement (DATASETS) 317, 394
DEFINE statement (REPORT) 1042
PROC MEANS statement 617
PROC TABULATE statement 1367, 1411
order variables 987, 1041
orthogonal expressions 1295
OS option
PROC JAVAINFO statement 608

OTHERWISE statement
  DATA step versus FCMP procedure 437
OUT= argument
  APPEND statement (DATASETS) 302
  COPY statement (CATALOG) 137
  COPY statement (DATASETS) 319
  PROC IMPORT statement 597
  PROC MIGRATE statement 686
OUT= option
  CONTENTS statement (CATALOG) 136
  CONTENTS statement (DATASETS) 317
  OUTPUT statement (MEANS) 627
  PROC COMPARE statement 217, 237, 253
  PROC OPTSAVE statement 718
  PROC PRTEXP statement 934
  PROC PWENCODE statement 938
  PROC RANK statement 948
  PROC REPORT statement 1014
  PROC SOAP statement 1156
  PROC SORT statement 1176
  PROC STANDARD statement 1339
  PROC TABULATE statement 1368
  PROC TRANSPOSE statement 1505
OUT2= option
  CONTENTS statement (DATASETS) 317
OUTALL option
  PROC COMPARE statement 217
OUTARGS statement
  FCMP procedure 431
OUTBASE option
  PROC COMPARE statement 217
OUTCOMP option
  PROC COMPARE statement 217
OUTDIF option
  PROC COMPARE statement 217
OUTDUR statement
  CALENDAR procedure 79
outer joins 1262, 1294, 1309
OUTER UNION set operator 1272
OUTFILE= argument
  PROC EXPORT statement 410
OUTFIN statement
  CALENDAR procedure 79
OUTLIB= option
  PROC CPORT statement 275
  PROC FCMP statement 424
OUTNOEQUAL option
  PROC COMPARE statement 217
OUTOBS= option
  PROC SQL statement 1208
OUTPERCENT option
  PROC COMPARE statement 217
output data sets
  comparing observations 253
  summary statistics in 256
OUTPUT= option
  CALID statement (CALENDAR) 73
OUTPUT statement
  MEANS procedure 627
output statistics 663
  extreme values with 669, 672
  for several variables 665
  with missing class variable values 667
Output window
  printing from 1002
OUTREPT= option
  PROC REPORT statement 1015
OUTSTART statement
  CALENDAR procedure 80
OUTSTATS= option
  PROC COMPARE statement 218, 238, 256
OUTTABLE= argument
  PROC EXPORT statement 411
OUTTYPE= option
  PROC CPORT statement 275
OUTWARD= option
  PLOT statement (PLOT) 732
OVERLAY option
  PLOT statement (PLOT) 732
  PLOT statement (TIMEPLOT) 1486
overlaying plots 742, 748
overlining 1024, 1049
OVERWRITE option
  PROC SORT statement 1176
OVP option
  FSLIST command 582
  PROC FSLIST statement 580
OVPCHAR= option
  PLOT statement (TIMEPLOT) 1486

P

P keywords 1541
p-values 1568
page dimension 1394
page dimension text 1362
page ejects 830
page layout 833
  column headings 834
  column width 835
  customizing 876
  observations 833
  plots 1487
  with many variables 871
page numbering 891
PAGE option
  BREAK statement (REPORT) 1024
  DEFINE statement (REPORT) 1042
  PROC FORMAT statement 516
  RBREAK statement (REPORT) 1050
PAGEBY statement
  PRINT procedure 830
panels
  in reports 1108
PANELS= option
  PROC REPORT statement 1016
parameters 1545
partitioned data sets
  multi-threaded sorting 1181
password-protected data sets
  appending 305
  copying files 326
  transporting 278
passwords 346
  assigning 346
  changing 346
  DATASETS procedure with 355
  encoding 937, 939
  encoding methods 942
  removing 346

paste buffer
saving encoded passwords to 941
pattern matching 1268, 1328
PC environments
migrating short-extension files 691
PCTLDEF= option
PROC MEANS statement 619
PROC REPORT statement 1018
PROC TABULATE statement 1369
PCTN statistic 1395
denominator for 1396
PCTSUM statistic 1395
denominator for 1397
PDF files
style elements 1129
TABULATE procedure 1465
PDF reports 842
peakedness 1552
penalties 740
changing 741, 772
index values for 740
PENALTIES= option
PLOT statement (PLOT) 733
percent coefficient of variation 1538
percent difference 226
PERCENT option
CHART procedure 171
PROC RANK statement 948
percentage bar charts 177
percentages
displaying with denominator definitions 1451
in reports 1114
TABULATE procedure 1395, 1448, 1451
percentiles 1546
keywords and formulas 1541
permanent data sets 18
permanent informats and formats 539
accessing 539
retrieving 563
picture formats 520
building 526
creating 547
digit selectors 524
directives 525
filling 568
message characters 524
picture-name formats 530
PICTURE statement
FORMAT procedure 520
pie charts 158, 166
PIE statement
CHART procedure 166
PLACEMENT= option
PLOT statement (PLOT) 733
PLOT procedure 723
combinations of variables 729
computational resources 742
concepts 738
generating data with program statements 738
hidden observations 744
labeling plot points 739
missing values 744, 764
ODS table names 743
overview 720
portability of ODS output 743
printed output 743
results 743
RUN groups 738
scale of axes 743
syntax 723
task tables 723, 727
variable lists in plot requests 728
PLOT statement
PLOT procedure 727
TIMEPLOT procedure 1483
plots
collision states 741
contour plots 729, 755
customizing axes 1491
customizing plotting symbols 1491
data on logarithmic scale 752
data values on axis 753
hidden label characters 742
horizontal axis 746
labels 761, 766, 770
multiple observations, on one line 1497
multiple plots per page 749
overlaying 742, 748
page layout 1487
penalties 740
plotting a single variable 1489
plotting BY groups 758
pointer symbols 739
reference lines 742, 746
specifying in TIMEPLOT 1483
superimposing 1495
plotting symbols 744
customizing 1491
variables for 1493
PMENU catalog entries
naming 786
steps for building and using 792
storing 779
PMENU command 777
PMENU procedure 779
concepts 792
ending 792
execution of 792
initiating 792
overview 777
PMENU catalog entries 792
syntax 779
task tables 779, 783
templates for 793
pointer symbols 739
Polish collating sequence 1169
POLISH option
PROC SORT statement 1169
populations 1544
POS= option
PLOT statement (TIMEPLOT) 1486
PostScript files 868
PostScript output
previewing 928
POWER CALL routine 460
power of statistical tests 1567
PREFIX= option
PICTURE statement (FORMAT) 523
PROC TRANSPOSE statement 1505
preloaded formats 1042, 1376
class variables with 659, 1418

PRELOADFMT option
  CLASS statement (MEANS) 625
  CLASS statement (TABULATE) 1376
  DEFINE statement (REPORT) 1042
presorted input data sets 1184
PRESORTED option
  PROC SORT statement 1176
PRINT option
  PROC FCMP statement 424
  PROC MEANS statement 618
  PROC SQL statement 1209
  PROC STANDARD statement 1339
  PROC PRINTTO statement 890
PRINT procedure 818
  HTML reports 838
  listing reports 836, 841
  overview 815
  page layout 833, 871, 876
  PDF reports 842
  PostScript files 868
  procedure output 832
  results 832
  RTF reports 848
  style elements 823
  syntax 818
  task tables 818
  XML files 854
PRINTALL option
  PROC COMPARE statement 218
PRINTALLTYPES option
  PROC MEANS statement 618
printer attributes
  extracting from registry 933
  writing to data sets 936
  writing to log 935
printer definitions 921
  adding 931
  available to all users 930
  creating 935
  deleting 922, 931, 932
  exporting 923
  for Ghostview printer 928
  in SASHELP library 923
  modifying 931, 935
  multiple 928
  replicating 935
printers
  list of 923
  routing log or output to 892, 901
PRINTIDVARS option
  PROC MEANS statement 618
printing
  See also printing reports
  all data sets in library 882
  data set contents 259
  description of informats and formats 561
  formatted values 26
  grouping observations 845
  page ejects 830
  page layout 833, 871, 876
  selecting variables for 832, 835
  template for printing numbers 520
printing reports 1001
  batch mode 1002
  from Output window 1002
  from REPORT window 1001
  interactive line mode 1002
  noninteractive mode 1002
  PRINTTO procedure 1002
  with forms 1001
  with ODS 1001
PRINTMISS option
  TABLE statement (TABULATE) 1383
PRINTTO procedure 888
  concepts 891
  overview 887
  printing reports 1002
  syntax 888
  task table 888
probability function 1545
probability values 1568
PROBT keyword 1543
PROC CALENDAR statement 65
PROC CATALOG statement 133
  options 134
PROC CHART statement 161
PROC CIMPORT statement 193
PROC COMPARE statement 212
PROC CONTENTS statement 260
PROC CPORT statement 271
PROC DATASETS statement 297
PROC DISPLAY statement 402
PROC EXPORT statement 410
PROC FCMP statement 422
PROC FONTREG statement 499
PROC FORMAT statement 514
PROC FSLIST statement 578
PROC HTTP calls 592
PROC HTTP statement 590
PROC IMPORT statement 596
PROC JAVAINFO statement 607
PROC MEANS statement 613
PROC MIGRATE Calculator 684
PROC MIGRATE statement 686
PROC OPTIONS statement 707
PROC OPTLOAD statement 716
PROC OPTSAVE statement 718
PROC PLOT statement 723
PROC PMENU statement 779
PROC PRINT statement 818
PROC PRINTTO statement 888
PROC PROTO statement 904
PROC PRTDEF statement 922
PROC PRTEXP statement 934
PROC PWENCODE statement 938
PROC RANK statement 946
PROC REGISTRY statement 964
PROC REPORT statement 1005
PROC SCAPROC statement 1144
PROC SOAP statement 1154
PROC SORT statement 1167
PROC SQL statement 1204
PROC SQL tables 1199
  adding rows 1231
  aliases 1240, 1259
  altering columns 1213
  altering integrity constraints 1213
  changing column attributes 1216
  combining 1305
  counting rows 1286
  creating 1219, 1296
  creating, from query expressions 1222

creating, from query results 1299
deleting 1228
deleting rows 1226
indexes on columns 1216
initial values of columns 1216
inserting data 1296
inserting values 1231
integrity constraints 1217, 1223
joining 1258, 1303, 1322
joining a table with itself 1258, 1259
joining more than two tables 1266
joining three tables 1316
ordering rows 1243
recursive table references 1223
renaming columns 1216
retrieving data from 1271
selecting columns 1233
selecting rows 1233
source tables 1239
table definitions 1227
table expressions 1271, 1292
updating 1245, 1246, 1301
without rows 1222
PROC SQL views
adding rows 1231
creating, from query expressions 1224
creating, from query results 1314
deleting 1228
deleting rows 1226
embedding LIBNAME statements in 1225
inserting rows 1232
librefs and stored views 1224
migrating 688
selecting columns 1233
selecting rows 1233
sorting data retrieved by 1224
source views 1239
SQL procedure 1199
storing DBMS connection information 1225
updating 1225
updating column values 1245
updating tables through 1246
view definitions 1227, 1294
PROC STANDARD statement 1338
PROC SUMMARY statement 1352
PROC TABULATE statement 1363
PROC TIMEPLOT statement 1480
PROC TRANSPOSE statement 1504
procedure concepts 20
formatted values 26
input data sets 20
operating environment-specific procedures 31
processing all data sets in a library 31
RUN-group processing 21
shortcut notations for variable names 25
statistics, computational requirements 33
statistics, descriptions of 32
titles containing BY-group information 21
procedure output
as input file 898
default destinations 887
destinations for 887
page numbering 891
routing to catalog entries 895
routing to external files 892
routing to printer 892, 901
procedures
descriptions of 10
ending 41
functional categories 3
host-specific 1571
raw data for examples 1574
report-writing procedures 3, 5
statistical procedures 3, 6
utility procedures 4, 8
PROFILE= option
PROC REPORT statement 1016
PROFILE window
REPORT procedure 1066
program files
migrating 689
project management 63
PROMPT option
PROC REPORT statement 1017
PROC SQL statement 1209
PROMPTER window
REPORT procedure 1066
PROTO procedure 903, 904
basic C language types 909
C argument types 908
C helper functions and CALL routines 916
C return types 908
C structures in SAS 911
character variables 909
concepts 907
interfacing with external C functions 910
missing values 910
numeric variables 909
registering function prototypes 907
results 918
splitter function 919
syntax 904
task tables 904
proxy
calling Web services with 1161
proxy servers 593
PROXYDOMAIN option
PROC SOAP statement 1156
PROXYHOST option
PROC SOAP statement 1156
PROXYPASSWORD option
PROC SOAP statement 1156
PROXYPORT option
PROC SOAP statement 1156
PROXYUSERNAME option
PROC SOAP statement 1156
PRT 1543
PRTDEF procedure 922
input data set 923
optional variables 925
overview 921
required variables 924
syntax 922
task table 922
valid variables 923
PRTEXP procedure 934
concepts 935
overview 933
syntax 934
PS= option
PROC REPORT statement 1017

PSPACE= option
  PROC REPORT statement 1017
pull-down menus 777
  activating 777
  associating FRAME applications with 812
  defining 786
  for DATA step window applications 806
  grayed items 784
  items in 783
  key sequences for 784
  separator lines 790
  submenus 790
PUT statement
  compared with LINE statement (REPORT) 1047
  DATA step versus FCMP procedure 437
PW= option
  MODIFY statement (DATASETS) 344
  PROC DATASETS statement 299
PWENCODE procedure 937
  concepts 938
  encoded passwords in SAS programs 940
  encoding methods 942
  encoding passwords 939
  encoding versus encryption 939
  saving encoded passwords to paste buffer 941
  syntax 937

Q

Q keywords 1541
QMARKERS= option
  PROC MEANS statement 618
  PROC REPORT statement 1017
  PROC TABULATE statement 1369
QMETHOD= option
  PROC MEANS statement 618
  PROC REPORT statement 1018
  PROC TABULATE statement 1369
QNTLDEF= option
  PROC MEANS statement 619
  PROC REPORT statement 1018
  PROC TABULATE statement 1369
QRANGE keyword 1542
quantiles 1018, 1369
  efficiency issues 7
  MEANS procedure 643
queries
  creating tables from results 1299
  creating views from results 1314
  DBMS queries 1255
  in-line view queries 1319
query-expression component 1271
query expressions 1271
  ALL keyword and 1272
  CORRESPONDING keyword and 1272
  creating PROC SQL tables from 1222
  creating PROC SQL views from 1224
  EXCEPT and 1275
  INTERSECT and 1277
  OUTER UNION and 1272
  set operators and 1271
  subqueries 1280
  UNION and 1274
  validating syntax 1246
QUIT statement 41
  procedures supporting 41

R

radio boxes 782, 787
radio buttons 782, 788
  color of 788
  default 787
RADIOBOX statement
  PMENU procedure 787
range 1550
RANGE keyword 1540
ranges
  for character strings 566
  FORMAT procedure 536
RANK procedure 943, 945
  computer resources 951
  concepts 951
  input variables 951
  missing values 954
  output data set 954
  partitioning observations 959
  ranking data 943
  results 954
  statistical applications 951
  syntax 945
  task tables 946
  tied values 952
  values of multiple variables 955
  values within BY groups 956
  variables for rank values 950
ranking data 943
RANKS statement
  RANK procedure 950
raw data
  character data to numeric values 554
  informats for 517
  procedures examples 1574
RBREAK statement
  REPORT procedure 1047
RBUTTON statement
  PMENU procedure 788
READ= option
  MODIFY statement (DATASETS) 344
  PROC DATASETS statement 300
READ_ARRAY function 439
REBUILD statement
  DATASETS procedure 347
RECORD statement
  SCAPROC procedure 1144
recursion 443
REDUCEPUT option
  PROC SQL statement 1209
REDUCEPUTOBS option
  PROC SQL statement 1209
REDUCEPUTVALUES option
  PROC SQL statement 1210
REF= option
  CHART procedure 172
  PLOT statement (TIMEPLOT) 1486
REFCHAR= option
  PLOT statement (TIMEPLOT) 1487
reference lines 742, 746
reflexive joins 1259
REFRESH option
  INDEX CENTILES statement (DATASETS) 338
registering function prototypes 907
registry 963
  clearing SASUSER 965

comparing file contents with 965, 973
comparing registries 965, 974
debugging 966
exporting contents of 966
extracting printer attributes from 933
importing to 966, 971
keys, subkeys, and values 966, 967
listing 972
listing contents in log 967
loading system options from 715
removing fonts from 505
sample entries 970
SASHELP specification 968
saving system option settings in 717
system fonts in 497
uppercasing key names 968
uppercasing keys, names, and values 968
writing contents to log 967
writing SASHELP to log 967
writing SASUSER to log 967
registry files
creating 968
key names 968
sample registry entries 970
structure of 968
values for keys 969
REGISTRY procedure 964
creating registry files 968
overview 963
syntax 964
task table 964
REMERGE option
PROC SQL statement 1211
remerging data
SQL procedure 1288
REMOVE statement
FONTREG procedure 502
RENAME statement
DATASETS procedure 349
renaming files 313
REPAIR statement
DATASETS procedure 349
REPLACE option
PROC EXPORT statement 412
PROC IMPORT statement 597
PROC PRTDEF statement 923
PROC STANDARD statement 1339
report definitions
specifying 1018
storing and reusing 1002, 1105
report items 1035
report layout 986
across variables 989, 1037
analysis variables 988, 1037, 1051
computed variables 989, 1038
display variables 987, 1039
group variables 988, 1040
order variables 987
planning 986
statistics 991
variables, position and usage 989
variables usage 987
REPORT= option
PROC REPORT statement 1018
REPORT procedure 1004
See also REPORT procedure windows
break lines 995
compound names 996
compute blocks 992
concepts 986
ending program statements 1045
formatting characters 1010
layout of reports 986
missing values 994, 1117
output data set 1120
overview 981
printing reports 1001
report-building 1075
report definitions 1002
report types 981
sample reports 981
statistics 991
style elements 997, 1129, 1134
summary lines 1076
syntax 1004
task tables 1005, 1022, 1035, 10
REPORT procedure windows 1052
BREAK 1052
COMPUTE 1056
COMPUTED VAR 1056
DATA COLUMNS 1056
DATA SELECTION 1057
DEFINITION 1057
DISPLAY PAGE 1063
EXPLORE 1063
FORMATS 1064
LOAD REPORT 1065
MESSAGES 1065
PROFILE 1066
PROMPTER 1066
REPORT 1067
ROPTIONS 1068
SAVE DATA SET 1072
SAVE DEFINITION 1073
SOURCE 1073
STATISTICS 1073
WHERE 1074
WHERE ALSO 1075
report variables 1075
REPORT window
printing from 1001
REPORT procedure 1067
report-writing procedures 3, 5
reports 981
See also report layout
building 1075
code for 1012
colors for 1037, 1048
column attributes 1028
column for each variable value 1099
columns 1030
computed variables 1122
consolidating observations 1097
customized 816
customized summaries 1046, 1110
default summaries 1022, 1047
detail reports 981
from DICTONARY tables 1307
grouping observations 845
groups 1126
header arrangement 1030

help for 1012
ID variables 1040
limiting sums in 865
listing reports 816, 841
multiple-choice survey data 1441
multiple-response survey data 1436
order variables 1041
ordering rows in 1090
panels 1108
PDF 842
percentages in 1114
printing 1001
RTF 848
samples of 981
selecting variables for 832, 1087
shrinking 1001
statistics in 1093, 1103
stub-and-banner reports 1451
summary reports 981
suppressing 1014
RESET statement
SQL procedure 1232
response headers 594
restoring transport files
identifying file content 199
RESUME option
AUDIT statement (DATASETS) 312
REVERSE option
PLOT statement (TIMEPLOT) 1487
PROC SORT statement 1177
RIGHT option
DEFINE statement (REPORT) 1043
ROLLBACK statement (SQL) 1296
ROPTIONS window
REPORT procedure 1068
ROUND option
PICTURE statement (FORMAT) 524
PROC PRINT statement 821
routines
local variables in different routines with same name 443
subroutine declarations 427
row headings
customizing 1425
eliminating 1429
indenting 1432
ROW= option
TABLE statement (TABULATE) 1383
row spacing 1001
rows
adding to tables or views 1231
consolidating observations 1097
counting 1286
deleting 1226
inserting 1232
joins and 1259
ordering 1243
ordering in reports 1090
returned by subqueries 1256
selecting 1233, 1247
SQL procedure 1199
ROWS= option
PROC PRINT statement 822
RTF files
style elements 1129
TABULATE procedure 1465
RTF reports 848
RTSPACE= option
TABLE statement (TABULATE) 1384
RUN-group processing 21
CATALOG procedure 143
DATASETS procedure 353, 355
RUN groups
PLOT procedure 738
RUN_MACRO function 472
RUN_SASFILE function 476

S

S= option
PLOT statement (PLOT) 736
samples 1545
sampling distribution 1555
SAS 6
migrating libraries 690
SAS/ACCESS views
migrating 688
SQL procedure 1199
SAS/AF applications
executing 401, 402
SAS code
calling from within functions 472
executing in specified fileref 476
SAS Code Analyzer 1143
filename or fileref for output 1144
Grid Job Generator 1149
output to record file 1145
record file specification 1148
SAS/CONNECT servers
migration and 694
SAS data views
SQL procedure 1199
SAS/OR 114
SAS programs
encoded passwords in 938, 940
SAS sessions
terminating 424
SAS/SHARE servers
migration and 694
SAS Web Services
calling 1159
SASUSER library
Ghostview printer definition in 928
SAVAGE option
PROC RANK statement 948
SAVE DATA SET window
REPORT procedure 1072
SAVE DEFINITION window
REPORT procedure 1073
SAVE statement
CATALOG procedure 141
DATASETS procedure 351
SCAPROC procedure 1143, 1144
results 1145
specifying Grid Job Generator 1149
specifying record files 1148
syntax 1144
task tables 1144
schedule calendars 59, 82
advanced 60
blank or with holidays 110
containing multiple calendars 97
multiple, with atypical work shifts 100, 105

simple 59
with holidays, 5-day default 94
scheduling 63
automating 114
based on completion of predecessor tasks 114
scope 442
searching for patterns 1268, 1269, 1328
SELECT clause
SQL procedure 1233
SELECT statement
CATALOG procedure 142
CIMPORT procedure 197
CPORT procedure 277
DATASETS procedure 352
FORMAT procedure 530
PRTEXP procedure 935
SQL procedure 1233
SELECTION statement
PMENU procedure 789
separator lines 790
SEPARATOR statement
PMENU procedure 790
set membership 1257
set operators 1271, 1295
SET statement
appending data 305
SETNULL C helper CALL routine 465, 917
short-extension files
migrating 691
SHORT option
CONTENTS statement (DATASETS) 317
PROC OPTIONS statement 709
SHOWALL option
PROC REPORT statement 1018
shrinking reports 1001
significance 1566
simple indexes 1218
simple random sample 1545
skewness 1551
SKEWNESS keyword 1540
SKIP option
BREAK statement (REPORT) 1025
RBREAK statement (REPORT) 1050
SLIBREF= option, PROC MIGRATE statement 687, 694
migrating with, across computers 696
migrating with, on same computer 698
migrating without, across computers 695
migrating without, on same computer 698
SLIST= option
PLOT statement (PLOT) 736
SOAP procedure 1153, 1154
calling SAS Web Services 1159
calling Web services with a proxy 1161
concepts 1157
making calls with HTTPS protocol 1158
SOAPEnvelope element 1157, 1160
SOAPHeader element 1157
SSL and 1158
syntax 1154
task tables 1154
without SOAPEnvelope element 1160
WS-Security client configuration 1157
SOAPACTION option
PROC SOAP statement 1156
SOAPEnvelope element 1157, 1160
SOAPHeader element 1157
SOLVE function 467
sort indicators 392
migration and 689
sort order
for character variables 1182
for numeric variables 1182
SORT procedure 1165, 1167
character variable sorting orders 1182
collating sequence 1169, 1182
concepts 1181
DBMS data source 1184
encoding values 1171
integrity constraints 1185
maintaining order of observations in BY groups 1191
multi-threaded sorting 1181
numeric variable sorting orders 1182
output 1186
output data set 1186
presorted input data sets 1184
results 1186
retaining first observation of each BY group 1193
sorting by values of multiple variables 1187
sorting data sets 1166
sorting in descending order 1189
stored sort information 1183
syntax 1167
task tables 1167, 1186
translation tables 1170
SORTEDBY= option
MODIFY statement (DATASETS) 345
sorting, multi-threaded 1181
sorting data retrieved by views 1224
SORTMSG option
PROC SQL statement 1211
SORTSEQ= option
PROC SORT statement 1170
PROC SQL statement 1211
SORTSIZE= option
PROC SORT statement 1177
SOUNDS-LIKE operator 1320
SOURCE window
REPORT procedure 1073
SPACE= option
CHART procedure 172
SPACING= option
DEFINE statement (REPORT) 1043
PROC REPORT statement 1018
special functions and CALL routines 451
C helper functions and CALL routines 463
matrix CALL routines 451
SPLIT= option
PLOT statement (PLOT) 737
PROC PRINT statement 823
PROC REPORT statement 1019
PROC TIMEPLOT statement 1480
splitter function
PROTO procedure 919
spread of values 1550
spreadsheets
importing from Excel workbook 602
importing subset of records from 603
SQL, embedded 1296
SQL components 1247
BETWEEN condition 1247
BTRIM function 1248
CALCULATED 1249

CASE expression 1249
COALESCE function 1251
column-definition 1251
column-modifier 1252
column-name 1254
CONNECTION TO 1255
CONTAINS condition 1255
EXISTS condition 1256
IN condition 1257
IS condition 1257
joined-table 1258
LIKE condition 1268
LOWER function 1270
query-expression 1271
sql-expression 1277
SUBSTRING function 1284
summary-function 1285
table-expression 1292
UPPER function 1293
sql-expression component 1277
correlated subqueries and 1283
functions and 1278
operators and order of evaluation 1278
query expressions and 1280
subqueries and efficiency 1283
truncated string comparison operators and 1280
USER and 1278
SQL procedure 1199, 1201
See also SQL components
ANSI Standard and 1293
coding conventions 1200
collating sequence 1294
column modifiers 1294
combinations of column values 1325
combining two tables 1305
counting missing values with a macro 1331
creating tables and inserting data 1296
creating tables from query results 1299
creating views from query results 1314
data types and dates 1251
functions supported by 1295
identifiers and naming conventions 1296
indexes 1218
joining three tables 1316
joining two tables 1303, 1322
matching case rows and control rows 1328
missing values 1257, 1331
orthogonal expressions 1295
outer joins 1309
PROC SQL tables 1199
querying in-line views 1319
reporting from DICTIONARY tables 1307
reserved words 1293
resetting options 1232
retrieving values 1320
statistical functions 1295
syntax 1201
task tables 1203, 1204
terminology 1199
three-valued logic 1296
updating PROC SQL tables 1301
user privileges 1296
views 1199
square root value 468
SRSURL option
PROC SOAP statement 1156
SSL 591
SOAP procedure and 1158
standard deviation 1540, 1551
standard error of the mean 1540, 1556
STANDARD procedure 1338
missing values 1343
output data set 1343
overview 1335
results 1343
standardizing data 1335
statistical computations 1343
syntax 1338
task tables 1338
STANDARDIZE procedure
executing on each row of a data set 491
standardizing data 1335
order of variables 1342
specifying variables 1342
weights for analysis variables 1342
star charts 159, 166
STAR statement
CHART procedure 166
START statement
CALENDAR procedure 80
STARTAT= option
PROC REGISTRY statement 967
STATE= option
ITEM statement (PMENU) 785
statements with same function in multiple procedures 35
ATTRIB 35
BY 36
FORMAT 35
FREQ 39
LABEL 35
QUIT 41
WEIGHT 42
WHERE 47
STATES option
PLOT statement (PLOT) 737
statistic, defined 1545
statistic option
DEFINE statement (REPORT) 1043
statistical analysis
transposing data for 1520
statistical functions 1295
statistical procedures 3, 6
efficiency issues 7
quantiles 7
statistical summaries 1285
statistically significant 1566
statistics
based on number of arguments 1287
computational requirements for 33
descriptive statistics 1351
for groups of observations 7
formulas for 1536
in reports 1093
keywords for 1536
measures of location 1546
measures of shape 1551
measures of variability 1550
normal distribution 1552
percentiles 1546
populations 1544
REPORT procedure 991
samples 1545

sampling distribution 1555
summarization procedures 1544
table of descriptive statistics 32
TABULATE procedure 1392
testing hypotheses 1565
weighted statistics 42
weights 1544
statistics procedures 1535
STATISTICS window
REPORT procedure 1073
STATS option
PROC COMPARE statement 218
STD keyword 1540
STD= option
PROC STANDARD statement 1340
STDDEV keyword 1540
STDERR keyword 1540
STDMEAN keyword 1540
STIMER option
PROC SQL statement 1211
stored sort information 1183
string comparison operators
truncated 1280
STRUCT statement
FCMP procedure 429
STRUCTINDEX C helper CALL routine 465, 917
structure types 429
stub-and-banner reports 1451
Student's t distribution 1567
Student's t statistic 1543
two-tailed p-value 1543
Student's t test 643
STYLE= attribute
CALL DEFINE statement (REPORT) 1029
style attributes
applying to table cells 1400
assigning with formats 1400
style elements
class variable level value headings 1378
for keyword headings 1380
for ODS output 1129, 1134
in dimension expressions 1388
PRINT procedure 823
REPORT procedure 997, 1129, 1134
TABULATE procedure 1369, 1398, 1465, 14
STYLE= option
BREAK statement (REPORT) 1025
CLASS statement (TABULATE) 1377
CLASSLEV statement (TABULATE) 1378
COMPUTE statement (REPORT) 1034
DEFINE statement (REPORT) 1043
ID statement (PRINT) 829
KEYWORD statement (TABULATE) 1380
PROC PRINT statement 823
PROC REPORT statement 1019
PROC TABULATE statement 1369
RBREAK statement (REPORT) 1050
REPORT procedure 997
SUM statement (PRINT) 831
TABLE statement (TABULATE) 1384
TABULATE procedure 1399
VAR statement (PRINT) 832
VAR statement (TABULATE) 1390
style precedence 1470
STYLE_PRECEDENCE= option
TABLE statement (TABULATE) 1386
SUBGROUP= option
CHART procedure 172
SUBMENU statement
PMENU procedure 790
submenus 790
subqueries 1280
compared with joins 1268
correlated 1283
efficiency and 1283
returning rows 1256
subroutine declarations 427
SUBROUTINE statement
FCMP procedure 430
subroutines
creating with FCMP procedure 432
declaring computational code blocks for 430
location of compiled 448
updating argument lists 431
subsetting data
SQL procedure 1241, 1243
WHERE statement 47
SUBSTITUTE= option
CHECKBOX statement (PMENU) 780
RBUTTON statement (PMENU) 788
SUBSTRING function (SQL) 1284
subtables 1362
SUBTRACTMATRIX CALL routine 461
SUFFIX= option
PROC TRANSPOSE statement 1505
SUM keyword 1541
sum of squares
corrected 1538
uncorrected 1541
sum of the weights 1541
SUM option
CHART procedure 172
SUM statement
CALENDAR procedure 81
PRINT procedure 830, 852, 856
SUMBY statement
PRINT procedure 831
summarization procedures
data requirements 1544
SUMMARIZE option
BREAK statement (REPORT) 1025
RBREAK statement (REPORT) 1050
summarizing data
SQL procedure 1286
summary calendars 59, 83
multiple, with atypical work shifts 123
multiple activities per day 88
simple 61
with MEAN values by observation 119
summary-function component 1285
counting rows 1286
remerging data 1288
statistics based on number of arguments 1287
summarizing data 1286
summary lines 981
construction of 1076
SUMMARY procedure 1352
overview 1351
syntax 1352
summary reports 981
summary statistics
COMPARE procedure 233, 256

SUMSIZE= option
  PROC MEANS statement 620
SUMVAR= option
  CHART procedure 172
SUMWGT keyword 1541
superimposing plots 1495
SUPPRESS option
  BREAK statement (REPORT) 1026
survey data
  multiple-choice 1441
  multiple-response 1436
SUSPEND option
  AUDIT statement (DATASETS) 312
Swedish collating sequence 1169
SWEDISH option
  PROC SORT statement 1169
SYMBOL= option
  CHART procedure 172
symbol variables
  TIMEPLOT procedure 1482
symmetric matrices
  Cholesky decomposition for 453
SYSINFO macro variable 227
system failures 309
system fonts 497
system options
  display setting for single option 711
  display settings for a group 703
  list of current settings 701
  loading from registry or data sets 715
  OPTIONS procedure 701
  procedures and 18
  saving current settings 717
  short form listing 710

T

T keyword 1543
table aliases 1240, 1259
table definitions 1227
table-expression component 1292
table expressions 1271
TABLE statement
  TABULATE procedure 1380
tables
  See also PROC SQL tables
  applying style attributes to cells 1400
  cells with missing values 1408
  class variable combinations 1415
  crosstabulation 1451
  customizing headings 1425
  describing for printing 1380
  formatting values in 1393
  multipage 1434
  style precedence 1470
  subtables 1362
  two-dimensional 1413
TABULATE procedure 1356, 1363
  BY-group processing 1394
  class variable combinations 1415
  complex tables 1357
  concepts 1392
  customizing row and column headings 1425
  dimension expressions 1386
  eliminating horizontal separators 1432
  eliminating row headings 1429
  formatting characters 1365
  formatting class variables 1393
  formatting values in tables 1393
  frequency counts and percentages 1451
  headings 1407, 1409, 1411
  indenting row headings 1432
  missing values 1377, 1401
  multilabel formats 1423
  multipage tables 1434
  ODS and 1358
  page dimension 1394
  percentage statistics 1395, 1448
  portability of ODS output 1412
  preloaded formats with class variables 1418
  reporting on multiple-choice survey data 1441
  reporting on multiple-response survey data 1436
  results 1401
  simple tables 1356
  statistics 1392
  style elements 1369, 1398
  style elements for ODS output 1465
  style precedence 1470
  summarizing information 1427
  syntax 1363
  task tables 1363, 1381, 1399
  terminology 1359
  two-dimensional tables 1413
TAGSORT option
  PROC SORT statement 1178
TAPE option
  PROC CIMPORT statement 196
  PROC CPORT statement 275
templates
  for printing numbers 520
  PMENU procedure 793
temporary arrays 472
temporary data sets 18
temporary informats and formats 539
temporary variables 1075
TERMINATE option
  AUDIT statement (DATASETS) 312
text fields 782, 791
TEXT statement
  PMENU procedure 791
threads
  multi-threaded sorting 1181
THREADS option
  PROC MEANS statement 620
  PROC REPORT statement 1019
  PROC SQL statement 1212
  PROC TABULATE statement 1372
  SORT procedure 1178
three-valued logic 1296
tied values 952
TIES= option
  PROC RANK statement 948
TIMEPLOT procedure 1480
  data considerations 1487
  missing values 1488
  ODS table names 1488
  overview 1477
  page layout 1487
  procedure output 1487
  results 1487
  symbol variables 1482
  syntax 1480

task tables 1480, 1483
titles
BY-group information in 21
TRACE option
PROC FCMP statement 424
TRANSLATE= option
PROC CPORT statement 275
translation tables 1170
applying to transport files 282
for exporting catalogs 278
transport files 191
applying translation tables to 282
COPY procedure 262
CPORT procedure 269
identifying content of 199
transporting data sets 328
COPY procedure 262
password-protected 278
TRANSPOSE CALL routine 462
TRANSPOSE option
PROC COMPARE statement 218, 235
TRANSPOSE procedure 1501, 1504
attributes of transposed variables 1511
complex transposition 1503
copying variables without transposing 1508
duplicate ID values 1508
formatted ID values 1508
labeling transposed variables 1509, 1514
listing variables to transpose 1510
missing values 1509
naming transposed variables 1511, 1513, 1518
output data set 1510
output data set variables 1510
results 1510
simple transposition 1502, 1512
syntax 1504
task table 1504
transposing BY groups 1516, 1517
transposing data for statistical analysis 1520
transposition types 1502
transpositions with BY groups 1506
variable names, from numeric values 1509
transposed variables 1502
attributes of 1511
labeling 1509, 1514
naming 1511, 1513, 1518
TRANTAB 278
TRANTAB statement
CPORT procedure 278
TRAP option
PROC TABULATE procedure 1372
TrueType font files
replacing from a directory 508
searching directories for 503
TRUETYPE statement
FONTREG procedure 503
truncated string comparison operators 1280
trust stores 592
two-dimensional tables 1413
two-tailed tests 1566
Type 1 font files 504
Type I error rate 1566
Type II error rate 1567
TYPE= option
CHART procedure 173
MODIFY statement (DATASETS) 345
TYPE1 statement
FONTREG procedure 504
TYPES statement
MEANS procedure 634

U

UCLM keyword 1544
UL option
BREAK statement (REPORT) 1026
RBREAK statement (REPORT) 1051
uncorrected sum of squares 1541
underlining 1024, 1026, 1049, 10
UNDO_POLICY= option
PROC SQL statement 1212
unformatted values
comparing 226
UNIFORM option
PROC PLOT statement 725
PROC TIMEPLOT statement 1480
UNINSTALL= option
PROC REGISTRY statement 967
union joins 1264
UNION operator 1274
UNIQUE keyword 1218
UNIQUE option
CREATE INDEX statement (DATASETS) 339
UNIT= option
PROC PRINTTO statement 891
universe 1544
unsorted data
comparing 220
UPCASE option
INVALUE statement (FORMAT) 518
PROC REGISTRY statement 968
UPCASEALL option
PROC REGISTRY statement 968
UPDATE statement
SQL procedure 1245
UPDATECENTILES= option
CREATE INDEX statement (DATASETS) 340
INDEX CENTILES statement (DATASETS) 338
UPPER function (SQL) 1293
URL option
PROC SOAP statement 1156
USER data library 18
user-defined functions 433
GTL with 493
user input
collecting in dialog boxes 797
USER literal 1278
USER_VAR option
AUDIT statement (DATASETS) 312
USESASHELP option
PROC FONTREG statement 499
PROC PRTDEF statement 923
PROC PRTEXP statement 934
PROC REGISTRY statement 968
USS keyword 1541
utility procedures 4, 8

V

VALIDATE statement
SQL procedure 1246

validation tools
for migrating libraries 684, 693
VALUE option
PROC OPTIONS statement 709
value-range-sets 536
VALUE statement
FORMAT procedure 532
VAR keyword 1541
VAR statement
CALENDAR procedure 81
COMPARE procedure 221
MEANS procedure 635
PRINT procedure 832
RANK procedure 951
STANDARD procedure 1342
SUMMARY procedure 1353
TABULATE procedure 1389
TRANSPOSE procedure 1510
VARDEF= option
PROC MEANS statement 620
PROC REPORT statement 1020
PROC STANDARD statement 1340
PROC TABULATE statement 1373
variability 1550
variable formats
COMPARE procedure 226
variable labels
changing 345
variable names
shortcut notations for 25
variable scope 442
variables
across variables 989, 1037
analysis variables 988, 1037
associating informats and formats with 512, 538
attributes of 342
CHART procedure 173
class variables 1374
computed variables 989, 1038, 1122
copying without transposing 1508
display variables 987, 1039
group variables 988, 1040
ID variables 1040
in reports 987
labels 341
local variables in different routines with same name 443
nested 1362
order of 832
order variables 987, 1041
position and usage in reports 989
renaming 349
report variables 1075
selecting for printing 832, 835
selecting for reports 1087
SQL procedure 1199
standardizing 1335
temporarily associating formats with 29
temporarily dissociating formats from 30
temporary 1075
transposing into observations 1501
variables, comparing
by position 223
comparison summary 229
different data sets 244
different variable names 222
listing for matching 220
matching variables 208
multiple times 245
same data set 222, 247
selected variables 222
value comparison results 232
values comparison summary 231
variance 1541, 1551
VARNUM option
CONTENTS statement (DATASETS) 318
VAXIS= option
PLOT statement (PLOT) 737
VBAR statement
CHART procedure 167
version option
PROC JAVAINFO statement 608
vertical bar charts 156, 167
subdividing bars 179
VEXPAND option
PLOT statement (PLOT) 737
view definitions 1227
ORDER BY clause in 1294
views
copying 325
in-line 1240, 1294, 1319
migrating 687
SQL procedure 1199
VPERCENT= option
PROC PLOT statement 725
VPOS= option
PLOT statement (PLOT) 737
VREF= option
PLOT statement (PLOT) 737
VREFCHAR= option
PLOT statement (PLOT) 737
VREVERSE option
PLOT statement (PLOT) 738
VSPACE= option
PLOT statement (PLOT) 738
VTOH= option
PROC PLOT statement 725
VZERO option
PLOT statement (PLOT) 738

W

WARNING option
PROC COMPARE statement 218
WAYS option
OUTPUT statement (MEANS) 633
WAYS statement
MEANS procedure 636
WBUILD macro 805
Web service
invoking 590
Web services
calling with a proxy 1161
invoking 1153
WEBAUTHDOMAIN option
PROC SOAP statement 1156
WEBDOMAIN option
PROC SOAP statement 1156
WEBPASSWORD option
PROC SOAP statement 1156
WEBUSERNAME option
PROC SOAP statement 1156

WEEKDAYS option
  PROC CALENDAR statement 71
WEIGHT= option
  DEFINE statement (REPORT) 1043
  VAR statement (MEANS) 635
  VAR statement (TABULATE) 1390
WEIGHT statement 42
  calculating weighted statistics 42
  example 43
  MEANS procedure 637
  procedures supporting 42
  REPORT procedure 1051
  STANDARD procedure 1342
  TABULATE procedure 1391
weight values 1010, 1365
weighted statistics 42
weighting observations 637
weights 1544
  analysis variables 42
WHEN statement
  DATA step versus FCMP procedure 437
WHERE ALSO window
  REPORT procedure 1075
WHERE clause
  SQL procedure 1241
WHERE statement 47
  example 47
  procedures supporting 47
WHERE window
  REPORT procedure 1074
WIDTH= option
  CHART procedure 173
  DEFINE statement (REPORT) 1044
  PROC PRINT statement 827
window applications
  menus for 806
windows
  associating with menus 809
WINDOWS option
  PROC REPORT statement 1020
WITH statement
  COMPARE procedure 222
work shifts 91
  defaults 90
  schedule calendars 100, 105
  summary calendars 123
WORKDATA= option
  PROC CALENDAR statement 71
workdays data set 71, 90
  default workshifts instead of 90
  missing values 91
  workshifts 91
WRAP option
  PROC REPORT statement 1021
WRITE= option
  MODIFY statement (DATASETS) 345
WRITE statement
  SCAPROC procedure 1145
WRITE_ARRAY function 441
WS-Security
  client configuration 1157
WSSAUTHDOMAIN option
  PROC SOAP statement 1156
WSSPASSWORD option
  PROC SOAP statement 1157
WSSUSERNAME option
  PROC SOAP statement 1157

X

XML files 854
XSL procedure 1529

Z

ZEROMATRIX CALL routine 462

Your Turn

We want your feedback.

- If you have comments about this book, please send them to **yourturn@sas.com**. Include the full title and page numbers (if applicable).
- If you have comments about the software, please send them to **suggest@sas.com**.

SAS® Publishing Delivers!

Whether you are new to the work force or an experienced professional, you need to distinguish yourself in this rapidly changing and competitive job market. SAS® Publishing provides you with a wide range of resources to help you set yourself apart. Visit us online at support.sas.com/bookstore.

SAS® Press

Need to learn the basics? Struggling with a programming problem? You'll find the expert answers that you need in example-rich books from SAS Press. Written by experienced SAS professionals from around the world, SAS Press books deliver real-world insights on a broad range of topics for all skill levels.

support.sas.com/saspress

SAS® Documentation

To successfully implement applications using SAS software, companies in every industry and on every continent all turn to the one source for accurate, timely, and reliable information: SAS documentation. We currently produce the following types of reference documentation to improve your work experience:

- Online help that is built into the software.
- Tutorials that are integrated into the product.
- Reference documentation delivered in HTML and PDF – **free** on the Web.
- Hard-copy books.

support.sas.com/publishing

SAS® Publishing News

Subscribe to SAS Publishing News to receive up-to-date information about all new SAS titles, author podcasts, and new Web site features via e-mail. Complete instructions on how to subscribe, as well as access to past issues, are available at our Web site.

support.sas.com/spn

sas | THE POWER TO KNOW®